Czech Technical University in Prague

Faculty of Electrical Engineering

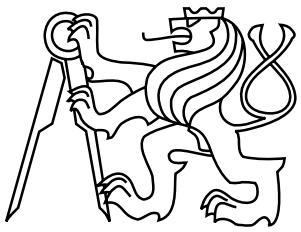
Doctoral Thesis

May 2016

Jan Hrnčíř

Czech Technical University in Prague

Faculty of Electrical Engineering
Department of Computer Science



Models and Algorithms for Sustainable Journey Planning

Doctoral Thesis

Jan Hrnčíř

Prague, May 2016

Ph.D. Programme: Electrical Engineering and Information Technology
Branch of study: Artificial Intelligence and Biocybernetics

Supervisor: doc. Ing. Michal Jakob, Ph.D.

Dedicated to Anna.

Acknowledgements

First of all, I would like to thank my supervisor Michal Jakob for his guidance and support, I have learnt a lot from him. Next, I appreciate the help of my master thesis supervisor Michael Rovatsos with the timetabled transport ridesharing problem. I am also grateful for the collaboration with Pavol Žilecký and Jan Nykl. Lastly and most importantly, I would like to thank my dear girlfriend Anna for her endless support during my studies.

Selected parts of this work have been partially supported by the European Union Seventh Framework Programme (grant agreements no. 289067 and 609023) and by the Czech Technical University (grant no. SGS13/210/OHK3/3T/13). Access to computing and storage facilities owned by parties and projects contributing to the National Grid Infrastructure MetaCentrum, provided under the programme "Projects of Large Infrastructure for Research, Development, and Innovations" (LM2010005), is greatly appreciated.

Abstract

The thesis focuses on models and algorithms for *journey planning for sustainable transport*, i.e., planning journeys from an origin to a destination that respect user preferences and utilise sustainable modes of transport. Our motivation is to provide people with intelligent tools that would help them discover routes that best suit their transport needs and, consequently, to facilitate the much needed shift towards sustainable mobility. In order to achieve our objectives, we first define formal models that enable us to efficiently represent transport networks. On top of these network models, we then develop efficient algorithms that solve three important sustainable journey planning problems. Specifically, we solve the problems of multi-criteria bicycle routing, intermodal journey planning, and ridesharing on timetabled transport services. We evaluate our implemented algorithms using real-world data. We then integrate our algorithms into prototype journey planning systems and validate them in real-world field trials with thousands of users in total. Finally, based on our practical experience with the real-world deployment, we discuss key aspects of engineering real-world journey planning systems, including the quality assurance in journey planning and the efficient implementation of journey planning algorithms.

Contents

Abbreviations

API	Application Programming Interface
DARP	Dial-a-Ride Problem
EAP	Earliest Arrival Problem
GTD	Generalised Time-Dependent
GTFS	General Transit Feed Specification
HMLS	Heuristic-enabled Multi-criteria Label-Setting
IJP	Intermodal Journey Planner
MA-STRIPS	Multiagent STRIPS
MCR	Multimodal Multi-criteria RAPTOR
MLC	Multi-criteria Label-Correcting
MLS	Multi-criteria Label-Setting
NAMOA*	New Approach to Multi-Objective A*
NaPTAN	National Public Transport Access Nodes
NPTDR	National Public Transport Data Repository
OSM	OpenStreetMap
PDDL	Planning Domain Definition Language
PT	Public Transport
REST	Representational State Transfer
SQL	Structured Query Language
SRID	Spatial Reference System Identifier
SRTM	Shuttle Radar Topography Mission
TDD	Time-dependent Dijkstra
TED	Time-expanded Dijkstra
TRL	Technology Readiness Level
WGS 84	World Geodetic System 1984
XML	Extensible Markup Language

Chapter 1

Introduction

Transport and mobility plays an important role in nowadays society. In Europe[1], the transport industry directly employs more than 10 million people that corresponds to 4.5% of total employment. When passenger transport[2] is considered, more than 6000 billion passenger-kilometres (pkm) are travelled every year with more than 70% pkm travelled by car. Such a huge amount of travel, however, takes its toll on the environment. Transport greenhouse gas emissions increased[3] by around 34% between 1990 and 2008. Nowadays, transport is responsible for about a quarter of the EU's greenhouse gas emissions. In larger cities, drivers spend tens of hours a year in road traffic jams. In total, congestion costs Europe about 1% of GDP every year[4].

In order to alleviate the negative impacts of growing transport, we need to make transport and mobility sustainable. The World Business Council for Sustainable Development defines *sustainable mobility* as "the ability to meet the needs of society to move freely, gain access, communicate, trade, and establish relationships without sacrificing other essential human or ecological values today or in the future" [1]. Moving to sustainable transport corresponds to general sustainability goal which is in Europe proclaimed by the Europe 2020[5] growth strategy.

Moreover, in addition to transport becoming more intensive, transport systems are also becoming more complex, offering multitude different ways of travel. Providing intelligent tools that would help citizens make the best use of mobility services on offer is thus needed more than ever [92]. The problem of journey planning in transport networks has been studied for a long time [5]. In principle, majority of techniques find a shortest path using a graph model of the transport network. Traditionally, the problem of routing on road networks has been studied since 1960s. In last decades,

[1] http://ec.europa.eu/transport/strategies/facts-and-figures/transport-matters/index_en.htm

[2] http://www.eea.europa.eu/data-and-maps/figures/passenger-transport-volume-billion-pkm-1

[3] http://ec.europa.eu/transport/strategies/facts-and-figures/putting-sustainability-at-the-heart-of-transport/index_en.htm

[4] See footnote 1.

[5] http://ec.europa.eu/europe2020/index_en.htm

routing research has focused on more challenging problems of multimodal public transport routing, intermodal routing, and realistic network modelling (e.g., taking into account real-time information about public transport vehicles).

Given the proclaimed shift towards sustainable transport and mobility, it is important that journey planning techniques fully support planning journeys using sustainable modes of transport. Therefore, the aim of the thesis is to develop models and algorithms that support *journey planning for sustainable transport*, i.e., planning journeys from an origin to a destination that respect user preferences and utilise a combination of sustainable modes of transport. The sustainable modes of transport include (shared) bike, (shared) electric scooter, shared car, or public transport (PT). To give just one example, bike is an affordable human-powered mode of transport that does not need oil-based fuels. Importantly, riding a bicycle has positive health effects related to regular physical activity and decreased air and noise pollution [115].

1.1 Research Objectives

The high-level objective of the thesis is to design, develop, evaluate, and validate models and algorithms that would maximally support journey planning for sustainable transport. We summarise the research objectives into the following items:

1. **Formal models.** The first objective is to understand the transport network domain and formalise models that will enable solving the chosen problems of sustainable journey planning using efficient algorithms. Each of the models needs to capture the specific features of the chosen problems.

2. **Efficient algorithms.** The second objective is to design and implement efficient algorithms that would solve the chosen problems using the models developed within Research objective 1. The algorithms should have favourable computational properties allowing them to scale to real-world problem sizes. The implementation should use memory-efficient data structures, preprocessing, and other low-level enhancements to keep the runtime per query and memory consumption reasonably low. An inseparable part of this objective is to evaluate the implemented algorithms using real-world map and PT timetables data. The algorithms will be evaluated with respect to runtime and quality of returned plans.

3. **Validation in real-world deployments.** The third and final objective is to validate the implemented algorithms within Research objective 2 in real-world deployments with real users. This objective comprises the full research and development cycle starting from models and continuing with efficient algorithms and their evaluation. Through real-world deployments only, it is possible to discover real issues and the next research directions to achieve a system capable of real-world operation.

1.2 Problem Classification

In this thesis, we explore three related problems of sustainable journey planning that are marked by yellow boxes in Figure 1.1. The problems are classified according to three dimensions: single-agent vs multiagent, single-criteria vs multi-criteria, and unimodal vs multimodal vs intermodal routing.

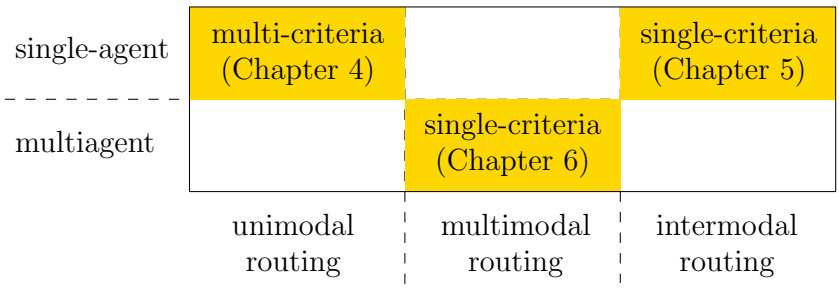


Figure 1.1: Problem classification according to three dimensions. Three researched problems of sustainable journey planning are marked by yellow boxes. References in brackets point to chapters where the problems are presented in detail.

In the first problem classification dimension, within single-agent problems, there is just one actor in an environment (e.g., a user of a multimodal journey planner that requests a journey plan) with no interaction with other actors. In multiagent problems, the actors interacts with each other in an environment (e.g., they collaborate in ridesharing to accomplish their goal of getting to their destinations). The second problem classification dimension concerns the number of criteria considered when solving a journey planning problem. Finally, the third problem classification dimension classifies journey planning problems with respect to number of modes combined in the planner. In *unimodal routing*, only one mode of transport is handled (for example the bike in bike planners). In *multimodal routing*, several public transport modes are combined (e.g., a public transport planner handling combinations of underground, trams, and buses). In the most complex setting of *intermodal routing*[6], planners should be able to plan journeys which use a combination of a full range of transport modes, including scheduled PT (e.g., bus, underground, tram, train, ferry), individual transport (e.g., walk, bike, shared bike and car), and on-demand transport (e.g., taxi). We use the term *intermodal* in order to stress that we consider modes and combinations thereof that go beyond what is supported in existing multimodal journey planners.

[6]In our previous work [63, 71], we referred to this term as *fully multimodal routing*. In recent years, the terminology has evolved and so we now use the term *intermodal routing* since it has become a commonly used term in the routing research community.

1.3 Routing Problems Addressed

More specifically, we study the following related problems since none of them has been satisfyingly solved by the research community (see more details in Chapter 2).

First, we study the problem of *unimodal single-agent multi-criteria routing* where we address the multi-criteria bicycle routing problem with the focus on urban areas, cf. Chapter 4. There are two main issues to be explored. First, the bicycle routing problem has not been properly formalised using well-grounded formal model. Second, the vast majority of existing approaches to bicycle routing do not use multi-criteria search methods and they thus cannot produce diverse sets of suggested routes properly accounting for cyclists' multiple route-choice criteria.

Second, we study the problem of *single-agent single-criteria intermodal routing* where different modes of transport can be combined, cf. Chapter 5. Until recently, little work has been done on solving the intermodal routing problem allowing general combinations of individual and public transport modes. In parallel to our research, several approaches aiming to solve the intermodal routing problem have emerged – this means that this problem has become topical.

Finally, we explore the problem of *multiagent single-criteria multimodal routing*, i.e., multimodal ridesharing, cf. Chapter 6. To the best of our knowledge, no existing work seems to attempt to compute joint travel plans based on public transport timetables, let alone in a way that takes into account the strategic nature of the problem, which comes about through the different (and potentially conflicting) preferences of individual travellers.

All the problems focus on routing in the sustainable transport domain and are therefore interconnected through the transport network models, cf. Chapter 3. Solving these problems enables us to gain an integrated view on the researched area of journey planning for sustainable transport.

1.4 Thesis Structure

The text of the thesis is organised as follows. Chapter 2 presents an overview of related work required to solve the three problems introduced in this chapter. Chapter 3 then defines transport network models for both static and time-dependent transport networks. It also focuses on the real-world data sources needed to instantiate transport network models.

The models are essential for the following three chapters focusing on selected problems of journey planning for sustainable transport: Chapter 4 addresses multi-criteria bicycle routing with the focus on urban areas, Chapter 5 tackles intermodal journey planning, and Chapter 6 focuses on ridesharing on timetabled transport services.

In Chapter 7, we again bring the experience from solving the selected routing problems together and discuss common engineering aspects. Chapter 8 concludes the thesis by a summary of achievements and outlooks for future research.

Chapter 2

Related Work

This chapter reviews the necessary background required to solve the three problems introduced in Chapter 1. The chapter provides information about state-of-the-art algorithms and speedup techniques for single-criteria unimodal routing, multi-criteria unimodal routing, multimodal public transport routing, intermodal journey planning, and multiagent trip planning. An outline of this chapter is shown in Figure 2.1 where related work sections are mapped to three areas researched in the thesis.

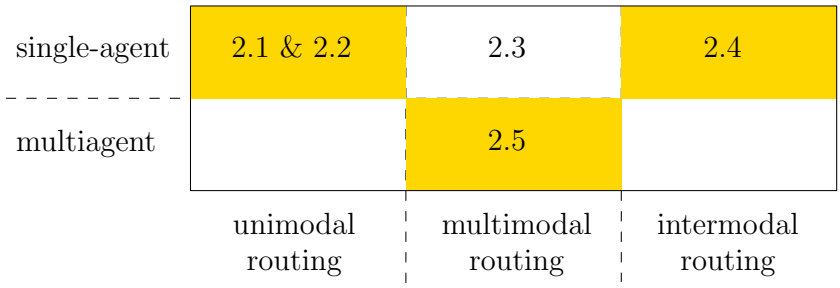


Figure 2.1: An outline of the related work sections with respect to three researched problems of sustainable journey planning denoted by yellow boxes.

2.1 Single-Criteria Unimodal Routing

The aim of a single-criteria unimodal routing algorithm is to find an optimal path given an origin, a destination, and a network graph. We call this problem a shortest path point-to-point problem. This section discusses routing on static road networks which are typically used in this problem.

The road network is typically modelled as a directed graph $G = (V, E)$ where the set of nodes V represents junctions and the set of edges E represent roads. Each edge is assigned a weight capturing the cost (e.g., time) needed to travel across this edge. As an input to a route planning algorithm, the journey origin and destination are represented as nodes $o, d \in V$ of the road graph. As an output, the algorithm returns

an optimal path, i.e., a sequence of edges, from the origin to the destination node. An optimal path is the path with the least total cost.

The basic algorithm for finding a shortest point-to-point path is the Dijkstra's algorithm [37]. When the Fibonacci heap [46] is used for the priority queue of nodes, the computational complexity of the Dijkstra's algorithm is $O(|E| + |V| + \log |V|)$. In practice, the runtimes of the Dijkstra's algorithm are excessive when a large road network is used. In order to speedup road routing, the road network graph can be preprocessed and speedup techniques exploiting the following properties of the road network can be used [98]:

1. A road network is a very sparse and almost planar graph.
2. A road network has a layout, i.e., geographic coordinates of nodes are known.
3. A road network usually has hierarchical properties (there are for example main streets and less important ones).

Depending on the size of the transport network, speedup techniques enable answering shortest path queries in a fraction of a second. The most recent algorithms pushed the average query time to microseconds on the Western Europe graph with 42.5 million directed edges. In the following list, there are examples of specific speedup techniques that use advanced preprocessing strategies and goal-directed search techniques to exploit the above mentioned properties of road networks.

1. **SHARC** [8]: A unidirectional hierarchical approach that runs a strongly goal-directed search on the highest level and automatically level down when getting close to a goal node.
2. **Landmark A* (ALT)** [52, 53]: A combination of A* search and a lower-bounding technique based on landmarks and the triangle inequality.
3. **Highway hierarchies** [94, 95]: An approach exploiting the natural hierarchy of road networks that uses a complete bidirectional search. The graph is constructed in a way that a local area around an origin and a destination contains all edges whereas a sparser highway network shortest paths preserving graph is constructed for the rest of the road network.
4. **Contraction hierarchies** [50]: This approach is based on the contraction of the least important nodes. In the algorithm, the shortest paths using a contracted node are replaced by shortcuts.
5. **Transit-node routing** [6, 7]: Distance table for important transit nodes and all relevant connections between the other nodes and the transit nodes are precomputed. Fast table lookups are then used in the shortest path search.

6. **Customisable Route Planning** [31]: This algorithm uses arc separators to build the overlay graph. It is engineered to handle turn costs and fast updates of the cost function needed by real-world systems operating on road networks. The preprocessing is therefore divided in two parts: metric-independent and customisable.

7. **Hub Labeling** [30]: In the preprocessing, each node in the road graph is assigned a label. The idea is that for any pair of nodes, the mutual distance can be calculated only by retrieving label values of the two nodes. The labels must obey the cover property.

2.2 Multi-Criteria Unimodal Routing

Multi-criteria shortest path problem has been studied in the literature for a long time [5]. The goal is to find a full Pareto set of routes, i.e., all routes non-dominated by any other route. It belongs to the category of NP hard problems [47]. The main parameter that affects the runtime of the algorithm is the size of Pareto set. In general, the Pareto set can be exponentially large in the input graph size even for the case of two optimisation criteria [57, 81]. That leads to runtimes that are not practically usable.

As far as general multi-criteria shortest path algorithms are concerned, the *multi-criteria label-setting* (MLS) algorithm [57, 78] extends Dijkstra's algorithm [37] by operating on labels that have multiple cost values. For each node, the algorithm stores a bag of non-dominated labels. The priority queue stores labels (typically in a lexicographic order) instead of nodes as in Dijkstra's algorithm. A minimum label from the priority queue is processed in every iteration. On the contrary, the *multi-criteria label-correcting* (MLC) algorithm [25, 34] processes the whole bag of nondominated labels associated with a current node at once. Therefore, labels may be scanned multiple times during one run of the algorithm.

To speedup the multi-criteria search, heuristic accelerations have attracted considerable attention, aiming at finding a set of routes that is similar to the optimal Pareto solution. First, the search space can be pruned using the ellipse around the origin and destination [56]. Second, in [89], the authors proposed a near admissible multi-criteria search algorithm to approximate the optimal set of Pareto routes in a state space graph by using the ϵ-dominance approach. It has been proven that the $(1+\epsilon)$-Pareto sets have a polynomial size [88]. Third, in [27], the authors developed several heuristics to weaken the domination rules during the search (e.g., using buckets for the criteria values). Fourth, in [55], the authors proposed a modified label-correcting algorithm with a new label-selection strategy and dominance conditions.

An alternative approach is represented by MOA* [104] and NAMOA* [77] that are multi-criteria extensions of the standard A* algorithm [58]. The extension lies in using a vector of heuristics and a search graph, i.e., a directed acyclic graph to record the set of non-dominated paths to visited nodes. Every time a new path reaches a node, its cost vector is checked for dominance with the set of all stored path cost

vectors at that node. The NAMOA* algorithm with the Tung & Chew heuristic [111] was recently shown to achieve an order of magnitude speedup for bicriteria road routing [76].

Since the thesis deals with the multi-criteria bicycle routing problem, we dedicate the rest of this section to the overview of search algorithms related to *bicycle routing*. In contrast to car and public transport route planning for which advanced algorithms and mature software implementations exist [5], bicycle routing is a relatively underexplored topic. In particular, despite the highly multi-criteria nature of cyclists' route-choice preferences, almost all existing approaches to bicycle routing do not use multi-criteria search methods to properly account for such a multi-criteriality. Instead, these existing bicycle routing approaches transform multi-criteria search to single-criterion search either by optimising each criteria separately [59, 107] or by using a weighted combination of all criteria [66, 112].

Unfortunately, the scalarisation of multi-criteria problems using a linear combination of criteria may miss many Pareto optimal routes [21, 26] and consequently reduce the quality and relevance of suggested routes. Scalarisation also requires the user to weight the importance of individual route criteria a priori, which is difficult for most users.

On the way to the multi-criteria search, there is an approach by Storandt [105] that solves a constrained shortest path problem in the bicycle routing domain. They solve two problems: (1) Find the route from an origin to a destination with the least (positive) height difference (summed over all segments) which has length at most D; (2) Find the route from an origin to a destination which is shortest among all paths which have height difference of at most H.

Only in the last few years, multi-criteria algorithms for bicycle routing started to appear. In [96], the authors showed how to effectively search for a best compromise solution in bicycle routing. However, the method only uses two optimisation criteria and it does not produce multiple solutions approximating the full Pareto set. Recently, in [103], we have explored the use of optimal multi-criteria shortest path algorithms for tri-criteria bicycle routing; however, the proposed algorithm is too slow for interactive route planning.

2.3 Multimodal Public Transport Routing

The aim of an algorithm for planning with scheduled PT services is to find an optimal journey(s) given an origin, a destination, time constraints, and PT timetables. The most important difference to road networks discussed in Section 2.1 is the inherent time dependence of the PT network because PT edges can be traversed only at specific, discrete times. We begin with the models used to model the PT network and continue with problem variants and possible solutions.

There are two major ways to model PT timetables for the planning algorithm as a search graph. On the one hand, in the *time-expanded approach* [100], each event at a PT station, e.g., the departure of a train, is modelled as a node in the graph. The

advantage of this approach is that it allows more-or-less straightforward modelling of model extensions (e.g., vehicle changes). On the other hand, in the *time-dependent approach* [15], the graph contains only one node for each station. Some experimental studies of the two approaches [9, 90] show that the time-dependent approach exhibits better performance than the time-expanded approach.

In this overview, we focus on two variants of the problem. To begin with, the *earliest arrival problem* (EAP) is defined as follows. Given an origin PT stop p_o, a destination PT stop p_d, and a departure time τ, find a journey that departs from p_o no earlier than τ and has the earliest possible arrival to p_d. The EAP problem can be answered in a straightforward way by Dijkstra's algorithm both using the time-dependent model (time-dependent Dijkstra, TDD, [19, 40]) and the time-expanded model (time-expanded Dijkstra, TED, [100]).

To speedup the search, we need to use more advanced techniques but we need to take into account that PT networks have different structural properties than road networks [2]. Because of this fact, some of the speedup techniques for road routing can be modified to solve the EAP while other speedup techniques cannot. In the following list, we present speedup techniques that solve the EAP.

- **SHARC** [8]: A modification of the technique used for road routing, cf. Section 2.1.
- **Contraction Hierarchies** [48]: A modification of the techniques used for road routing, cf. Section 2.1.
- **Core-ALT** [87]: A combination of ALT and core-based routing, where the core contains non-contracted time-dependent scheduled PT services while the roads are contracted.
- **Multi-level graph approach** [99, 101]: A hierarchical decomposition of a planning graph to multiple levels. Based on the node separators, the graph is decomposed to significantly smaller graphs that preserve shortest paths.
- **Access-node routing** [32]: Inspired by transit-node routing, table lookups are used in the multimodal shortest path search.
- **Accelerated Connection Scan Algorithm** [106]: An accelerated version of connection scan algorithm [35] based on the ideas used in customisable route planning [31].

In multimodal PT routing, additional criteria such as number of transfers, walking distance, or price are also important for the travellers. This leads to the *multi-criteria problem* which is defined as follows. Given an origin PT stop p_o, a destination PT stop p_d, and a departure time τ, find a Pareto set of non-dominating journeys with respect to the optimisation criteria that departs from p_o no earlier than τ. In the following list, we present techniques that solves the multimodal multi-criteria routing problem.

- **Layered Dijkstra** [15, 91]: Using the Dijkstra's algorithm [37], it finds a Pareto set of routes optimised with respect to travel time and number of transfers. It uses a timetable graph copied into K layers where K equals to the maximum number of transfers.
- **Multi-criteria Label-setting (MLS) Algorithm** [80, 91]: This algorithm extends Dijkstra's algorithm [37] by operating on labels that have multiple cost values. For more details see Section 2.2.
- **Transfer patterns** [4, 49]: A transfer pattern is the origin stop, the sequence of transfers, and the destination stop. The transfer patterns for pairs of stops are precomputed and then instantiated for a given departure time.
- **RAPTOR (Round-bAsed Public Transit Optimised Router)** [33]: An approach targeted specifically for PT networks which is not based on Dijkstra's algorithm and uses arrays of trips and routes[1] instead of a graph. The algorithm operates in rounds (one for each PT transfer) and computes arrival times by traversing every route at most once per round.
- **Public Transit Labeling** [28]: An approach that reuses ideas from Hub Labeling road network speedup [30] and applies them to the time-expanded model of PT network.

2.4 Intermodal Journey Planning

The aim of the algorithm for intermodal journey planning is given an origin, a destination, and time constraints to find an optimal multi-leg journey(s) which can use any reasonable combination of these modes of transport: fixed-schedule PT modes (e.g., bus, underground, tram, train, ferry), fixed-station free-floating modes (e.g., shared bikes), and unrestricted free-floating modes (e.g., walk, bike, car, taxi). The most important difference to fixed-schedule PT networks discussed in Section 2.3 is that the intermodal network consists of parts with different properties (e.g., static bicycle transport network compared to the time-dependent PT network).

Until recently, very little work has been done to tackle this general problem. One of few exceptions is the planner proposed by Horn [61] which supports combinations of scheduled PT and on-demand transport services. The planner is able to construct a multi-leg journey plan which can combine on-demand and scheduled PT transport. A limitation of the approach is that the on-demand mode in the multi-leg journey plan is restricted to the first or last non-walk leg of a journey, i.e., the on-demand mode serve as a feeder service. The second attempt at solving the intermodal EAP is provided by Yu and Lu [117] who use a genetic algorithm to construct the sequence of transport modes in a journey plan. In their experiments, Yu and Lu permit walk,

[1] Route is a group of trips (vehicle journeys) that are known to public by the same route number identifier.

bus, underground, and taxi modes. However, the individual modes of transport (bike, shared bike and car) are not used.

Very recently, in parallel to our research, several approaches tackling the intermodal journey planning problem have emerged. This is a sign that this problem has become a significant topic of current importance. To start with, Bast et al. [3] have used MLS algorithm and contraction to find meaningful combinations of walking, PT, and car or taxi. They also propose a filtering procedure called "types and thresholds" to produce a small representative subset of the full Pareto set of solutions. Adding bike as supported mode of transport, Delling et al. [27] have proposed multimodal multi-criteria RAPTOR (MCR) algorithm to tackle the problem. To solve the subproblems in PT and road network, the MCR uses RAPTOR and MLS, respectively. Adding shared (rental) bike and shared (rental) car to the network, Kirchler has extended the ALT speedup technique [52, 53] to State Dependent ALT (SDALT) technique [73]. In the preprocessing phase, landmarks are created to bound the search space. Dibbelt et al. [36] focus on the problem of finding intermodal journeys that can be constrained by arbitrary user-specified modal sequences. They build on contraction hierarchies [50] speedup technique and propose user-constrained contraction hierarchies (UCCH) algorithm. Finally, Gundling et al. [54] solve the problem of finding a journey with a PT backbone and private mode (walk, car, bike, and taxi) first and last mile. They address travel time, travel cost, and convenience as optimisation criteria.

2.5 Multiagent Trip Planning

Automated planning technology [51] has developed a variety of scalable heuristic algorithms for tackling hard planning problems, where plans, i.e., sequences of actions that achieve a given goal from a given initial state, are calculated by domain-independent problem solvers. Unlike other approaches to route planning and ridesharing, automated planning techniques permit a fairly straightforward formalisation of travel domains, and allow us to capture the joint action space and complex cost landscape resulting from travellers' concurrent activities. In terms of algorithmic complexity, the kind of multiagent planning needed to compute ridesharing plans for several agents is significantly harder than single-agent planning for two reasons: Firstly, the ability of each agent to execute actions concurrently [11] may result in exponentially large sets of actions available in each step in the worst case. Secondly, whenever individual agents have different (and potentially conflicting) goals [13], a joint solution must satisfy additional requirements, e.g., being compatible with everyone's individual preferences, or not providing any incentive for any individual to deviate from the joint plan. Solving the *general* multiagent planning for problem sizes of the scale we are interested in real-world ridesharing is therefore not currently possible using existing techniques.

Because of the desire to integrate different travellers' individual plans, ridesharing is quite similar to plan merging (e.g., [45, 110], where individual agents' plans are in-

crementally integrated into a joint solution. Compared to these approaches, however, in our domain every agent can always achieve their plan regardless of what others do, and agents do not require others' "help" to achieve their goals. This makes the problem simpler than those of plan merging though, in return, we place much higher scalability demands on the respective solution algorithms.

This explains also why, as will be shown below, we are able to achieve much higher scalability than state-of-the-art multiagent plan synthesis algorithms, e.g., [38, 82, 109]. These algorithms exploit "locality" in different ways in order to be able to plan for parts of a multiagent planning problem while temporarily ignoring others, e.g., by considering non-interacting subplans in isolation from each other. In a sense, our problem involves even more loosely coupled sub-tasks, as these can be essentially solved in a completely independent way, except in terms of cost optimisation.

The relationship between our work and approaches that focus more on decentralised planning, plan co-ordination, and conflict resolution among independent planning agents (e.g., [22, 23]) is similar – as no hard conflicts can arise among individual plans in ridesharing, it is not essential to co-ordinate individual plans with each other, other than for cost optimisation purposes.

Finally, as far as the strategic aspect is concerned, this is obviously also relevant to ridesharing as ultimately each co-traveller wants to achieve an optimal solution for themselves. Various approaches have studied this problem in the past (e.g., [13, 41, 113], yet none of them has been shown to scale to the type of domain we are interested in, with the exception of [72], which makes certain simplifying assumptions to achieve scalability: it does not consider *joint* deviation from equilibrium solutions (i.e., it only safeguards against individual agents opting out of a joint plan, not whole sub-groups of agents), and it assumes that agents will honour their promises when they have agreed on a joint plan. We believe that both these assumptions are reasonable in ridesharing, as we are envisioning a platform on which users would be automatically grouped together whenever a rideshare would be beneficial to each one of them. On such a platform, it is reasonable to assume that agreements could be enforced through a trusted third party, and that collusion among travellers could be avoided by not disclosing their identities to each other until the purchase of all tickets has been completed.

Ridesharing, i.e., purposeful and explicit planning to create groups of people that travel together in a single vehicle for parts of the journey, is a long known and widely studied problem. It solves the problem in a multiagent way compared to approaches that mitigates the negative impacts of the transport on the individual level. Existing work, however, focuses exclusively on ridesharing using vehicles that can move freely on a road transport network, without schedule or route restrictions. The work on such *non-timetabled* ridesharing covers the whole spectrum from formal problem models, through solution algorithms up to practical consumer-oriented services and applications.

On the theoretical side, the vehicle-based ridesharing problem is typically formalised as a *Dial-a-Ride Problem (DARP)*. Different variants of DARPs exist, differ-

ing, for example, in the nature of traveller's constraints, the distribution of pickup and delivery locations, the criteria optimised, or the level of dynamism supported. A comprehensive review of different variants of DARPs, along with a list of algorithmic solution approaches, is given by Cordeau et al. [20]. Most of the existing approaches rely on a centralised coordination entity responsible for collecting requests and producing vehicle assignment and schedules, though more decentralised approaches have also been presented more recently [116]. Bergbelia et al. [10] summarise recent advances in *real-time ridesharing*, which has been gaining prominence with the growing penetration of internet-connected smartphones and GPS-enabled vehicle localisation technologies. Existing work almost exclusively considers a single mode of transport only. One of few exceptions is the work of Horn et al. [60] which considers demand-responsive ridesharing in the context of flexible, multimodal transport systems; the actual ridesharing is, however, only supported for demand-responsive non-timetabled journey legs. On the practical side, there exist various online services for car ridesharing[2,3] as well as services which assist users in negotiating shared journeys[4].

So although both ridesharing using freely moving vehicle and single-agent journey planning for timetabled services have been extensively studied, the combination of both, i.e., ridesharing on timetabled services, has not been – to the best of our knowledge – studied before.

[2] https://liftshare.com/
[3] https://www.enterprisecarclub.co.uk/
[4] http://www.travbuddy.com/

Chapter 3

Transport Network Modelling

In order to be able to solve sustainable journey planning problems, we need to create formal models of real-world transport networks. The purpose of modelling is to create transport network models that reflect the essence of the transport network. On the one hand, the model should capture all important features of the network while on the other hand, the model needs to be compact to enable effective search by routing algorithms and to keep memory requirements low. Let us take bicycle routing as an example. We model all cycleways where cyclists can go together with factors that are important to cyclists (e.g., surface of the cycleway) while we discard all other pieces of information since they are not important for the cyclists (see details in Section 3.1.2).

More specifically, this chapter addresses Research objective 1 and presents the transport network modelling for journey planning for sustainable transport. We model the transport network as a graph. We distinguish two types of networks with respect to the time-dependency of edge links. First, we describe static network models where the cost of each edge does not depend on the departure time. Second, we describe time-dependent network models in order to represent the time-dependent networks (e.g., public transport network). Our important contribution is the *generalised time-dependent graph* which allows representing the combined static and time-dependent transport networks in a single model. Third, we briefly discuss the steps needed to adopt the transport network models for domain-independent planners. We show the relation between models and sustainable journey planning problems researched in the thesis in a comprehensive overview in Figure 3.1 At the end of this chapter, we present an overview of data needed to create instances of transport network models from data.

3.1 Static Transport Networks

In this section, we describe models for the static transport networks where the cost of each edge does not depend on the departure time. As a basis, we use a transport network graph to model the network of roads, cycleways, and pavements. Then we extend this model to a more detailed cycleway graph.

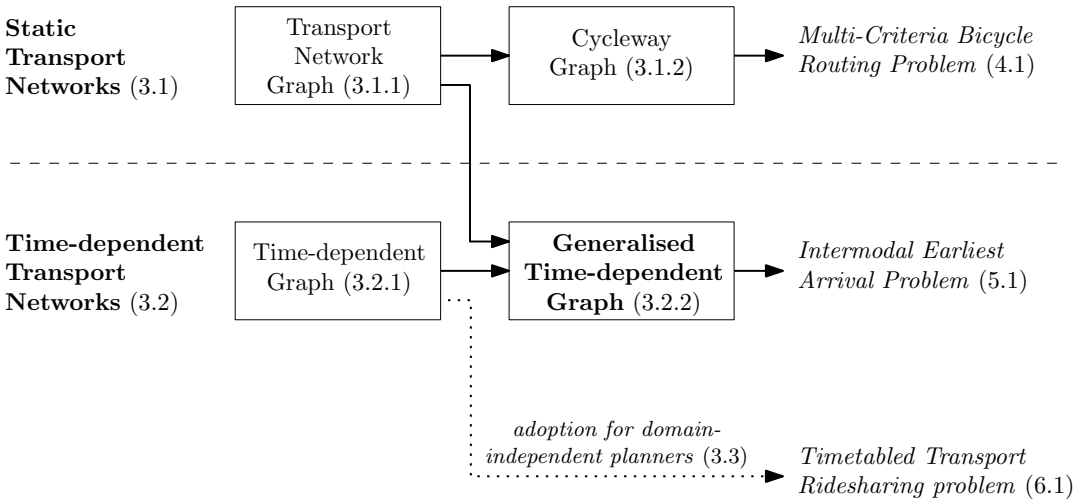


Figure 3.1: The relation between models and sustainable journey planning problems supplemented with pointers to corresponding sections.

3.1.1 Transport Network Graph

To model the network for individual modes of transport (e.g., walk, bike, shared bike, and car) and on-demand modes of transport (e.g., taxi), we use the *transport network graph* $G^N = (V^N, E^N, \rho^N)$ defined as a weighted directed graph, where the set of nodes V^N represents junctions and the set of edges E^N represents roads, pavements, and cycleways. The transport network graph is directed to properly model road links in the map (two edges represent a two-way link; one edge represents a one-way link). The length of each edge $(v, w) \in E^N$ is given by the weight function $\rho^N : E^N \to \mathbb{R}_0^+$.

3.1.2 Cycleway Graph

We extend the transport network graph to model a detailed cycleway network that is then used in the multi-criteria bicycle routing problem in Chapter 4. We represent the cycleway network as a weighted directed *cycleway graph* $G^C = (V^C, E^C, g, h, l, f, \overrightarrow{c})$, where V^C is the set of nodes representing start and end points (e.g., cycleway junctions) of cycleway segments, and $E^C \subseteq \{(u, v) | (u, v \in V^C) \wedge (u \neq v)\}$ is the set of edges representing cycleway segments.

The function $g : V^C \to \mathbb{R}^2$ assigns a latitude and a longitude value to each node $v \in V^C$. An altitude value is assigned to each node by the function $h : V^C \to \mathbb{R}$. The horizontal length of each edge $(u, v) \in E^C$ is given by the function $l : E^C \to \mathbb{R}_0^+$. Let $\wp(F)$ denote a powerset of F. For each edge $(u, v) \in E^C$, the function $f : E^C \to \wp(F)$ returns the *features* associated with the edge, which capture relevant properties of the edge as obtained from the input map data (e.g., the surface of the cycleway segment or the road type). The set of all edge features is denoted by F. Note that an edge can have multiple features assigned to it, thus $f((u, v)) \in \wp(F)$ with the number of

elements $|f((u,v))| \geq 1$.

The cost of each edge is calculated by a k-dimensional vector of cost functions $\overrightarrow{c} = (c_1, c_2, \ldots, c_k)$. The non-negative cost value $c_i((u,v))$ of i-th criterion for a given edge $(u,v) \in E$ is computed by the cost function $c_i : E^C \to \mathbb{R}_0^+$.

3.2 Time-dependent Transport Networks

In this section, we describe models for the time-dependent transport networks where the cost of each edge depends on the departure time. We start with a definition of a time-dependent graph with constant transfer times. Next, we combine this model with the transport network graph in a novel generalised time-dependent graph that is able to capture an intermodal transport network.

3.2.1 Time-dependent Graph

To model the network of scheduled PT services (e.g., bus, tram, underground), we use a *time-dependent graph* $G^T = (V^T, E^T, \rho^T)$ with constant transfer times [91] (the time needed to make a transfer between two lines at a stop is defined as a constant for each stop). We have chosen this model for its better performance than the time-expanded model [90]. Let S be the set of *stop nodes* corresponding to the stops that are physically present in the PT network. A stop node can be served by one or more routes. A *route* is a set of PT vehicle trips that are known to the public under the same route number identifier, e.g., the tram line number 3. Assuming that n is the number of routes using a stop $u \in S$, then n *route nodes* $R_u = \{r_1^u, \ldots, r_n^u\}$, one for each route, are associated with stop u. Route nodes are virtual nodes without corresponding counterparts in the real world and they are used to model constant transfer times. Without route nodes, it would not be possible to model non-zero transfer times between different routes at the same stop. The set of all route nodes is denoted as $R = \cup_{u \in S} R_u$. The set of nodes V^T of the time-dependent graph G^T is then defined as $V^T = S \cup R$.

The set of edges E^T of the time-dependent graph G^T is defined as $E^T = A \cup B \cup C$ where A denotes the set of edges between route and stop nodes, B denotes the set of edges between stop and route nodes, and C denotes the set of *route edges* between route nodes of the same route. Edges $(v,w) \in A \cup B$ are called *transfer edges*. Formally, the sets are defined as follows:

$$
\begin{aligned}
A = &\ \cup_{u \in S} \{(r^u, u) | r^u \in R_u\} \\
B = &\ \cup_{u \in S} \{(u, r^u) | r^u \in R_u\} \\
C = &\ \cup_{u,v \in S} \{(r^u, r^v) | r^u \in R_u \wedge r^v \in R_v\} \text{ where } r^u \\
&\quad \text{and } r^v \text{ are visited successively by the same route}
\end{aligned}
$$

The *link-traversal function* $f'_{(v,w)} : \mathbb{N} \to \mathbb{N}$ is associated with each edge $(v,w) \in C$ and defined as $f'_{(v,w)}(t) := t'$ where t is the departure time from v and $t' \geq t$ is the earliest possible arrival time at stop w. We assume that overtaking of vehicles on

edges of the same route is not permitted. This means that the earliest arrival of a PT vehicle to a route node r_j^w corresponds to the earliest departure from an adjacent departure route node r_i^v.

Let the function g_v return the constant transfer time at stop v. For example in Figure 3.3, the transfer from a route node r_0^v to r_1^v and vice versa takes time g_v. Then the travel duration ${\rho^T}_{(v,w)} : \mathbb{N} \to \mathbb{N}$ of traversing an edge $(v, w) \in E^T$ from v at the departure time t is defined as

$${\rho^T}_{(v,w)}(t) := \begin{cases} 0 & \text{if } (v,w) \in A \\ g_v & \text{if } (v,w) \in B \\ {f'}_{(v,w)}(t) - t & \text{if } (v,w) \in C \end{cases}$$

3.2.2 Generalised Time-dependent Graph

In this section, we define the newly proposed *generalised time-dependent (GTD) graph*, which allows representing the combined road network (for individual and on-demand modes) and PT network (for PT modes) in a single structure. The GTD graph is then used for the intermodal journey planning in Chapter 5.

The GTD graph is a generalisation of the time-dependent graph with constant transfer times defined by Pyrga et al. [91]. The GTD graph G is constructed from the following three structures (see also Figure 3.2):

1. *transport network graph* G^N for the network of pavements, cycleways, and roads (defined in Section 3.1.1)
2. *time-dependent graph* G^T for the PT network (defined in Section 3.2.1)
3. *graph connector* D of the time-dependent graph G^T and the transport network graph G^N (defined below)

Graph Connector. In order to plan multimodal journeys using combinations of individual, on-demand, and PT modes of transport, the time-dependent graph G^T and the transport network graph G^N need to be interconnected. Let $\theta : S \to \mathcal{P}(\mathcal{V}^{\mathcal{N}})$

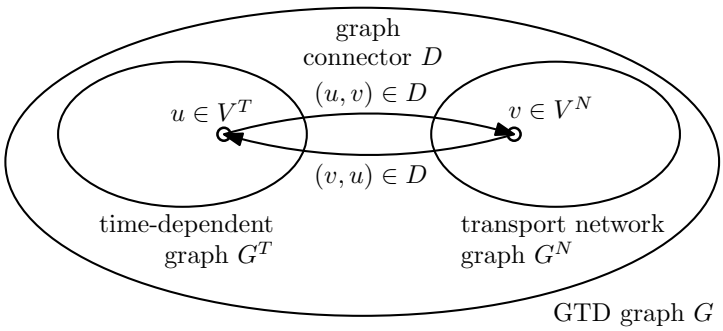


Figure 3.2: The structure of a GTD graph.

be a mapping that associates with each stop $v \in S$ a set of nodes $\theta(v) \in \mathcal{P}(\mathcal{V}^{\mathcal{N}})$ from the transport network graph. For the underground stops and large PT stations, the mapping assigns a stop a set of corresponding entrances from the transport network graph G^N. For the other PT stops, the mapping assigns to a stop a nearest pavement node from the transport network graph G^N. Then the *graph connector* D of graphs G^T and G^N is defined as a set of interconnecting edges:

$$D = \{(v, w) | (v \in S \wedge w \in \theta(v)) \vee (v \in \theta(w) \wedge w \in S)\}$$

A length in metres $\rho_d((v, w)) = |v, w|$ is assigned to each $(v, w) \in D$ (the Euclidean distance between v and w is used).

Once all three components of the GTD graph are defined, we can use the described structures to construct a unified network graph that supports intermodal journeys that use a combination of PT, individual, and on-demand modes of transport. Before defining the GTD graph, we define the edge weight ρ, the permitted modes function μ, and the permitted mode change predicate χ.

Firstly, let t be the departure time from node $v \in V$ and $vel \in \mathbb{R}^+$ the travel speed in metres per second. Then the *edge weight* $\rho_{(v,w)} : \mathbb{N} \times \mathbb{R}^+ \to \mathbb{N}$ returns the travel duration (in seconds) of traversing the edge $(v, w) \in E$ at time t using travel speed vel:

$$\rho_{(v,w)}(t, vel) := \begin{cases} \rho^T((u, w), t) & \text{if } (v, w) \in E^T \\ \rho^N((v, w))/vel & \text{if } (v, w) \in E^N \\ \rho_d((v, w))/vel & \text{if } (v, w) \in D \end{cases}$$

Secondly, assuming $M = \{m_1, \ldots, m_l\}$ is the set of all l supported modes of transport, the function $\mu : E \to \mathcal{P}(\mathcal{M})$ returns the set of permitted modes of transport $\mu((v, w)) \in \mathcal{P}(\mathcal{M})$ at an edge $(v, w) \in E$. In our approach, we currently use the following modes of transport: *walk (W), bike (I), shared bike (S), car (C), taxi (X), bus (B), tram (T)*, and *underground (U)*. Especially in the transport network graph G^N, there are usually several modes of transport permitted to use a given edge, e.g., car, taxi, and bike.

Thirdly, we need to capture the fact that certain changes of mode of transport are possible only at some nodes. For example, changing from walk to shared bike or vice versa is only possible at bike sharing stations. Formally, the *permitted mode change* predicate $\chi_v : M \times M$ is associated with each node $v \in V$ and $\chi_v(m_1, m_2)$ returns true if it is possible to change the mode of transport from m_1 to m_2 at node v.

As an example, let $S^N \subset V^N$ be the set of bike sharing stations and $P^N \subset V^N$ be the set of park and ride (P+R) parking places. For the modes of transport currently used, the predicate $\chi_v(m_1, m_2)$ for each $v \in V$ and $m_1, m_2 \in M$ is defined as follows

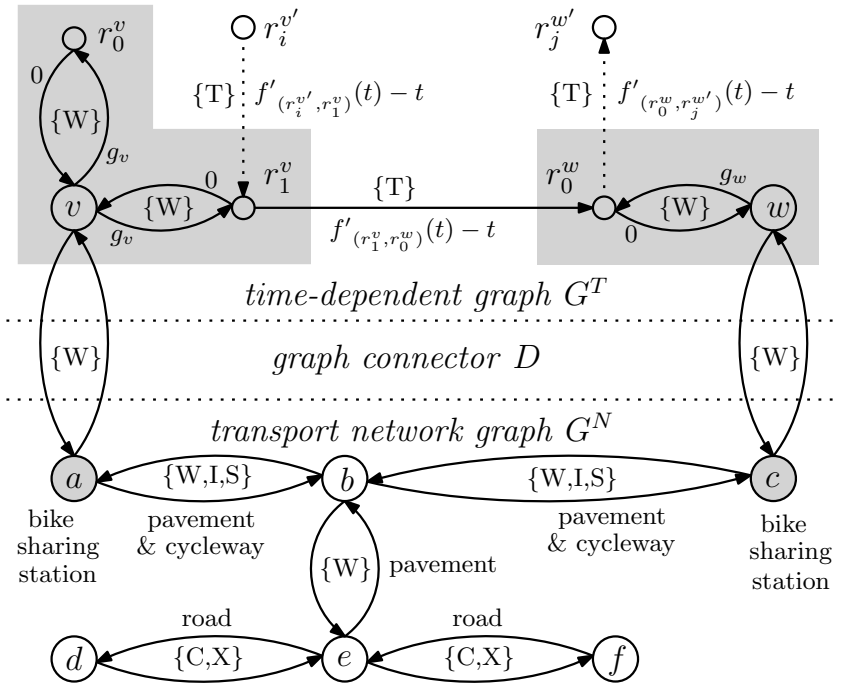


Figure 3.3: An example of the GTD graph. Edges are annotated with the permitted modes of transport. Stop nodes $v, w \in S$ represent two tram stops that are connected by one tram route connecting four route nodes $(r_i^{v'}, r_1^v, r_0^w, r_j^{w'})$. Route nodes $R_v = \{r_0^v, r_1^v\}$ and $R_w = \{r_0^w\}$ associated with the respective stop nodes v and w are highlighted with grey background. Edges from the time-dependent graph G^T are also annotated with their weight (edge traversal time).

(`t` denotes true, `f` denotes false):

$$\chi_v(m_1, m_2) := \begin{cases} \texttt{t} & \text{if } v \in V^T & (1) \\ \texttt{t} & \text{if } v \in S^N \wedge ((m_1 m_2 = \text{WS}) \\ & \qquad \vee (m_1 m_2 = \text{SW})) & (2) \\ \texttt{t} & \text{if } v \in P^N \wedge ((m_1 m_2 = \text{CW}) & (3) \\ \texttt{t} & \text{if } m_1 = m_2 & (4) \\ \texttt{f} & \text{otherwise} & (5) \end{cases}$$

The defined predicate captures the five following rules: (1) change of mode of transport is not restricted for any stop $v \in V^T$; (2) change from walk to shared bike or vice versa is possible only at bike sharing stations $v \in S^N$; (3) change from car to walk is possible only at P+R parking places; (4) change from m_1 to $m_2 = m_1$ is not a change (it is always permitted to continue with the same mode of transport); (5) change of modes is not permitted in all other cases.

Finally, we define the *generalised time-dependent graph* as a weighted directed graph $G = (V, E, M, \rho, \mu, \chi)$ where $V = V^T \cup V^N$ and $E = E^T \cup E^N \cup D$. An example of a GTD graph is shown in Figure 3.3.

3.3 Domain-Independent Planning Models

In Section 3.1 and Section 3.2, we model the transport network as a graph that is than used by domain-specific shortest path algorithms. For the purposes of the timetabled transport ridesharing problem (see Chapter 6), we need to adapt the models for domain-independent planners that are used to solve the problem.

On the one hand, the models for domain-independent planners builds on the graph-based models described above. On the other hand, since the full travel planning domain with a full granularity of timetabled connections is too large for any current state-of-the-art planner to deal with, we distinguish the full transport services domain from the relaxed transport services domain. In Section 6.1.1, the relaxed and full domain are described in detail. In addition, the models needs to be extended from a single-agent domain to a multiagent one, because the ridesharing problem includes interaction of multiple self-interested travellers, cf. Section 6.1.2. Finally, the domains needs to be translated into a Planning Domain Definition Language so they can be used with modern AI planning systems, cf. Section 6.3.

3.4 Instantiating Network Models from Data

In order to instantiate transport network models, various transport network data need to be combined. The data describe a *mobility services universe*, i.e., the information space describing all the aspects of the world required for journey planning. An overview of the used data is given in Table 3.1 where each part of the mobility services universe is linked to a data specification.

To begin with, OpenStreetMap (OSM) data is used to create the walk, cycleway, and road network. OSM data is organised into three entities: nodes, ways, and relations, which are associated with various tags (features). Each map feature is denoted by a key and a value in the form of `entity::key::value`, e.g., `way::highway::primary`. From OSM, it is also possible to acquire information about the locations of bike-sharing stations and parking places. Elevation for all nodes in the OSM data is acquired using the Shuttle Radar Topography Mission (SRTM) project. The Osmosis 0.41 and osmfilter tools has been used to extract areas from the OSM

Table 3.1: Mobility services universe: overview of data.

Part of the mobility services universe	**Data**
Walk, cycleway, and road network maps	OSM
Bike-sharing stations	OSM (`amenity:bicycle_rental`)
Parking places	OSM (`amenity:parking`)
Elevation	SRTM
Scheduled PT services in Helsinki	GTFS
Scheduled PT services in the UK	NaPTAN, NPTDR

Table 3.2: Spatial reference systems used.

Area	Spatial reference system name	SRID
Prague	S-JTSK (Ferro) / Krovak	2 065
Helsinki	KKJ / Finland zone 2	2 392
UK	British National Grid	27 700

data dump.

Data about scheduled PT services is imported in two different data formats. In Helsinki, Helsinki Regional Transport Authority openly provides the timetables in General Transit Feed Specification (GTFS) data format. A GTFS feed contains several CSV-like text files compressed in a ZIP file. Each file models a particular aspect of transit information: stops, routes, trips, and other timetable data. In the UK, PT data is publicly available from the Department for Transport of the British Government. The PT stops are defined in the National Public Transport Access Nodes (NaPTAN), a UK national system for uniquely identifying all the points of access to public transport. Every point of access (bus stop, railway station, etc.) is identified by an ATCO code (a unique identifier for all points of access to public transport in the UK), e.g., *9100YORK* for York Rail Station. Each stop in the NaPTAN XML data is also supplemented by common name, latitude, longitude, address and other pieces of information. This data also contains information about how the stops are grouped together (e.g., several bus bays that are located at the same bus station). The timetables are stored inside the National Public Transport Data Repository (NPTDR). The timetables are represented using TransXChange data format which is an XML-based UK standard for interchange of route and timetable data.

Geographical locations of all nodes in the OSM data and stops in the GTFS and NaPTAN data are represented as their longitude and latitude values using the World Geodetic System (version WGS 84). WGS 84 is a *geographic* coordinate system type identified by SRID 4 326 (Spatial Reference System Identifier). In order to simplify the complex calculation of the Euclidean distance between two nodes expressed in the WGS 84 coordinates (the calculation is for example very frequently used in the A* Euclidean distance heuristic), we use a *projected* coordinate system. The projected coordinate system is regional and projects the location from a spheroid to a plane. Three different projected coordinate systems are used, cf. Table 3.2.

Chapter 4

Multi-Criteria Bicycle Routing

Increasing the *adoption of cycling* is crucial for achieving more sustainable urban mobility since bicycle provides a convenient and affordable form of transport for most segments of the population. It has a range of health, environmental, economical, and societal benefits and is therefore promoted as a modern, sustainable mode of transport [39, 70]. Navigating larger cities on a bicycle is, however, often challenging due to cities' fragmented cycling infrastructure or complex terrain topology. Cyclists would thus benefit from intelligent route planning that would help them discover routes that best suit their transport needs and preferences.

The first problem addressed in this thesis is the unimodal single-agent multi-criteria routing. Following the motivation above, we opted to study the *multi-criteria bicycle routing problem* with the focus on urban areas. In contrast to car drivers, cyclists consider a significantly broader range of factors while deciding on their routes. By employing questionnaires and GPS tracking, researchers have found that besides travel time and distance, cyclists are sensitive to traffic volumes, junction control, turn frequency, slope, noise, pollution, and scenery [14, 114]. Because of the many factors cyclists consider in deciding their routes, employing multi-criteria route search is vital for properly accounting for cyclists' route-choice criteria. Moreover, the relative importance of these factors varies among cyclists and can also be affected by weather and the purpose of the trip [14]. Such a user- and context-dependent multi-criteriality makes bicycle routing a particularly difficult category of routing problems. Bicycle route planner would be particularly useful for inexperienced cyclists with limited knowledge of their surroundings but they would also benefit experienced riders who want to fine-tune their routes [43], in effect making cycling a more attractive and accessible transport option.

The vast majority of existing approaches to bicycle routing, however, do not use multi-criteria search methods and they thus cannot produce diverse sets of suggested routes properly accounting for cyclists' multiple route-choice criteria. In this thesis, we overcome these limitations and present a bicycle routing algorithm that properly considers multiple realistic cyclists' route-choice criteria yet is fast enough for interactive use. Our algorithm extends the well-known multi-criteria label-setting

algorithm [78] with several speedup heuristics in order to generate, in a much shorter time, routes that closely approximate the full set of Pareto optimal routes. In contrast to the majority of existing work, our algorithm employs a formulation of the multi-criteria bicycle routing problem that incorporates realistic route choice factors based on recent studies of cyclists' behaviour [14, 114]. We thoroughly evaluate our algorithm in terms of the speed and quality of suggested routes on a diverse set of real-world urban areas. Finally, we validate the algorithm in real-world deployments using a bicycle routing system with an open API.

4.1 Multi-Criteria Bicycle Routing Problem

The *multi-criteria bicycle routing problem* is defined as a triple $C = (G^C, o, d)$:

- $G^C = (V^C, E^C, g, h, l, f, \overrightarrow{c})$ is the *cycleway graph* (defined in Section 3.1.2)
- $o \in V^C$ is the route origin
- $d \in V^C$ is the route destination

A route π, i.e., a finite path $\pi = ((u_1, v_1), \ldots, (u_n, v_n))$ with a length $|\pi| = n$ from the origin o to the destination d in the cycleway graph G^C has an additive cost value:

$$\overrightarrow{c}(\pi) = \left(\sum_{j=1}^{|\pi|} c_1(u_j, v_j), \ldots, \sum_{j=1}^{|\pi|} c_k(u_j, v_j) \right)$$

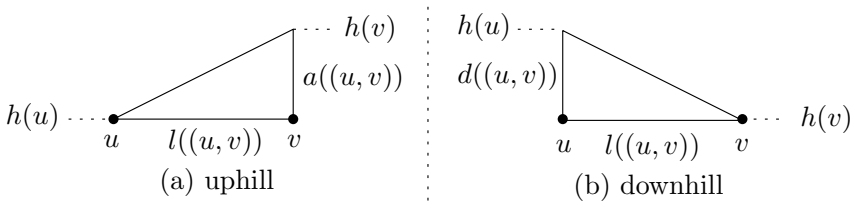
The solution of the multi-criteria bicycle routing problem is a full Pareto set of routes $\Pi \subseteq \{\pi | \pi = ((u_1, v_1), \ldots, (u_n, v_n))\}$, i.e., all routes non-dominated by any other route. A route π_p dominates another route π_q, i.e., $\pi_p \prec \pi_q$, iff $c_i(\pi_p) \leq c_i(\pi_q)$ for all $1 \leq i \leq k$ and $c_j(\pi_p) < c_j(\pi_q)$ for at least one j, $1 \leq j \leq k$.

4.1.1 Tri-Criteria Bicycle Routing Problem

Based on the studies of real-world cycle route choice behaviour [14, 114], we further consider a specific class of multi-criteria bicycle routing problems – a *tri-criteria bicycle routing problem*. Specifically, we consider the travel time criterion c_{time}, the comfort criterion c_{comfort}, and the elevation gain criterion c_{gain} defined in the next subsections. The criteria functions are then instantiated in Section 4.2.1.

Travel Time Criterion

The travel time criterion captures the preference towards routes that can be travelled in a short time. Travel time is a sensitive factor in cyclists' route planning especially for commuting purposes. To model the slowdown caused by obstacle features such as stairs or crossings, we define the slowdown coefficient $r_{\text{slowdown}} : \wp(F) \to \mathbb{R}_0^+$ which returns the slowdown in seconds on the given edge $(u, v) \in E^C$ with a set of features $f((u, v))$.



(a) uphill (b) downhill

Figure 4.1: Positive vertical (a) ascent a and (b) descent d.

Besides, changes in elevation may affect the cyclist's velocity and hence affect travel times. For the case of uphill rides, we define the positive vertical ascent $a : E^C \to \mathbb{R}_0^+$, cf. Figure 4.1.

$$a((u,v)) := \begin{cases} h(v) - h(u) & \text{if } h(v) > h(u) \\ 0 & \text{otherwise} \end{cases}$$

Analogously, for the case of downhill rides, we define the positive vertical descent $d : E^C \to \mathbb{R}_0^+$ (cf. Figure 4.1) and the positive descent grade $d' : E^C \to \mathbb{R}_0^+$ for a given edge $(u, v) \in E^C$ as follows:

$$d((u,v)) := \begin{cases} h(u) - h(v) & \text{if } h(u) > h(v) \\ 0 & \text{otherwise} \end{cases}$$

$$d'((u,v)) := \frac{d((u,v))}{l((u,v))}$$

To model the speed acceleration caused by vertical descent for a given edge $(u, v) \in E^C$, we define the downhill speed multiplier $s_d : E^C \times \mathbb{R}^+ \to \mathbb{R}^+$ as:

$$s_d((u,v), s_{\text{dmax}}) := \begin{cases} s_{\text{dmax}} & \text{if } d'((u,v)) > {d'}_{\text{c}} \\ \frac{(s_{\text{dmax}}-1)d'((u,v))}{{d'}_{\text{c}}} + 1 & \text{otherwise} \end{cases}$$

where $s_{\text{dmax}} \in \mathbb{R}^+$ is the maximum downhill speed multiplier, and ${d'}_{\text{c}} \in \mathbb{R}^+$ is the critical d' value over which a downhill ride would use the multiplier of s_{dmax}. This reflects the fact that the speed acceleration is remarkable for the ride on a steep downhill (compared to a mild one), however, it is limited due to safety concerns, bicycle physical limits and air drag.

Considering the integrated effect of edge length, the change in elevation and its associated features, the travel time criterion is defined as:

$$c_{\text{time}}((u,v)) = \frac{\text{distance}}{\text{speed}} + \text{ slowdown} =$$

$$= \frac{l((u,v)) + a_l \cdot a((u,v))}{s \cdot s_d((u,v), s_{\text{dmax}}) \cdot r_{\text{time}}((u,v))} + r_{\text{slowdown}}((u,v))$$

where s is the average cruising speed of a cyclist. a_l is the penalty coefficient for uphill rides (we use a modification of the Naismith's rule [97]). The criteria coefficient

$r_{\text{time}}((u,v))$ expresses the effect of a set of features $f((u,v))$ assigned to a given edge $(u,v) \in E^C$. Intuitively, $c_{\text{time}}((u,v))$ can model the travel time of flat rides, uphill rides, and downhill rides with $s_d((u,v), s_{\text{dmax}}) = 1$ for uphill and flat scenarios and $a((u,v)) = 0$ for downhill and flat scenarios. Details about the travel time coefficient r_{time} and slowdown coefficient r_{slowdown} are located in Section 4.2.1.

Comfort Criterion

The comfort criterion captures the preference towards comfortable routes with good-quality surfaces and low traffic. The comfort criteria summarises the effect of road surface and traffic volume along the edge. The surface coefficient $r_{\text{surface}}((u,v))$ penalises bad road surfaces, obstacles such as steps, and places where the cyclist needs to dismount his/her bicycle, with small values indicating cycling-friendly surfaces. The traffic coefficient $r_{\text{traffic}}((u,v))$ measures traffic volumes by considering the infrastructure for cyclists (e.g., dedicated cycleways), the type of roads, and the junctions, where low-traffic cycleways are assigned a small coefficient value. The maximum coefficient of the two is used here to avoid the cycleway segments that negatively affect the comfort the most. The comfort criteria coefficients are weighted by edge length $l((u,v))$, i.e., 500 m of cobblestones is worse than 100 m of cobblestones. The criteria function for comfort is then defined as:

$$c_{\text{comfort}}((u,v)) = \max\{r_{\text{surface}}((u,v)), r_{\text{traffic}}((u,v))\} \cdot l((u,v))$$

Elevation Gain Criterion

The elevation gain criterion captures the preference towards flat routes with minimum uphill segments. The criteria function for elevation gain takes into account the average cruising speed s, the positive vertical ascent $a((u,v))$, and the penalty coefficient for uphill rides a_l. The uphill ride over the edge (u,v) is penalised by adding the flat distance $a_l \cdot a((u,v))$. The criteria function for the elevation gain is defined as:

$$c_{\text{gain}}((u,v)) = \frac{\text{distance}}{\text{speed}} = \frac{a_l \cdot a((u,v))}{s}$$

4.2 Bicycle Routing Problem Instantiation from Data

We now describe important implementation details, in particular related to creating instances of the multi-criteria bicycle routing problem (see the definition in Section 4.1) from real-world map data. The instantiation of the routing problem comprises of two steps. In the first step, the cycleway graph is created from map data and its nodes and edges are assigned respective map features. A snapshot showing the cycleway network structure of Prague is illustrated in Figure 4.2. In the second step, the values of the criteria functions (see their definition in Section 4.1.1) are calculated for each edge.

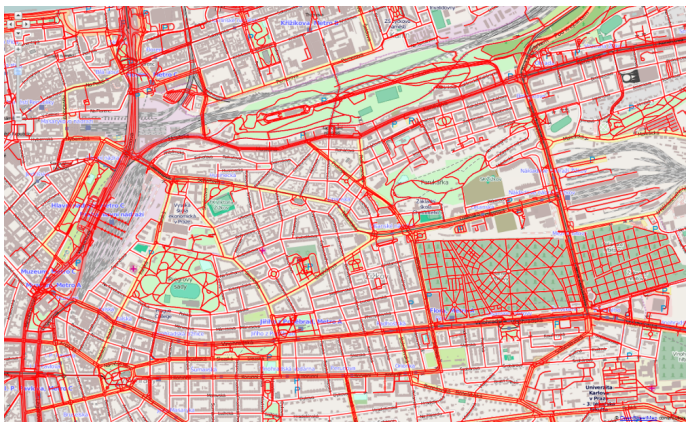


Figure 4.2: A snapshot of Prague cycleway network.

OSM data is used to create the cycleway graph. The following map elements relevant for cyclists are loaded according to the information from OSM tags associated with OSM nodes, ways, and relations. We divide the map features into six categories:

- *Surface:* surface quality in terms of smoothness of the surface and surface material, e.g., asphalt, gravel, or cobblestone.
- *Obstacles:* steps and elevators.
- *Dismount:* places where cyclists need to dismount the bicycle, e.g., pavement or footway crossing.
- *For bicycles:* description of the infrastructure for cyclists, e.g., dedicated cycleway, cycle lane, or shared busway.
- *Motor roads:* category of a road that is also used by cars, e.g., primary, secondary, residential, or living street.
- *Crossings:* crossings, crossroads, or traffic lights on the road.

4.2.1 OSM Tags Mapping

The three criteria c_{time}, c_{comfort}, and c_{gain} are computed using four criteria coefficients r_{time}, r_{slowdown}, r_{surface}, and r_{traffic} that aggregate the effect of map features $p \in F$. Based on the effect of map features on different criteria, we further map the six category of features to the optimisation criteria, as shown in Figure 4.3. To compute the values of criteria functions, we define here the four criteria coefficients – r_{time} and r_{slowdown} for the travel time criterion and r_{surface} and r_{traffic} for the comfort criterion. Their values are assigned using an expert opinion given by a group of chosen cyclists.

The travel time criterion is calculated using the travel time criteria coefficient r_{time} and the slowdown coefficient r_{slowdown}. The travel time criteria coefficient r_{time}

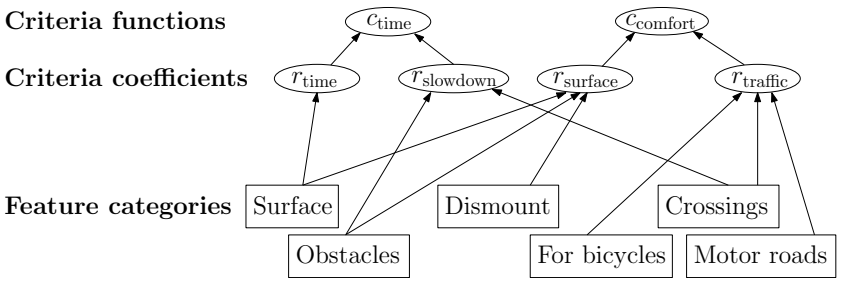


Figure 4.3: Calculating criteria values for c_{time} and c_{comfort} from map feature categories. The arrows show what are the inputs for the criteria and their criteria coefficients.

is computed using the travel time feature value $r'_{\text{time}} : F \to \mathbb{R}^+$ that represents the influence of a certain feature $p \in F$ on the travel time. The total effect depends on all features associated with an edge that contribute to the travel time criterion. Suppose function $f((u,v))$ returns the features associated with an edge $(u,v) \in E^C$, then the criteria coefficient r_{time} is given by:

$$r_{\text{time}}((u,v)) = \min\{r'_{\text{time}}(p) \mid p \in f((u,v))\}$$

For the travel time criterion, the minimum feature value is used since we are interested in a feature that reduces the cyclist's speed the most. Specific values of r'_{time} for the surface category of tags are shown in Table C.1 in Appendix C.

The slowdown coefficient r_{slowdown} is computed using the slowdown feature value $r'_{\text{slowdown}} : F \to \mathbb{N}_0^+$ caused by each relevant feature $p \in F$. Similarly, the total effect depends on all features associated with the edge $(u,v) \in E^C$ that may cause a slowdown in seconds, hence the slowdown coefficient $r_{\text{slowdown}} : \wp(F) \to \mathbb{N}_0^+$ is given by:

$$r_{\text{slowdown}}((u,v)) = \max\{r'_{\text{slowdown}}(p) \mid p \in f((u,v))\}$$

We take the maximum value of r'_{slowdown} since we are interested in a feature that slows down the cyclist the most. Specific values of r'_{slowdown} for an example subset of features $p \in F$ are shown in Table C.1 in Appendix C.

The comfort criterion is calculated using the surface criteria coefficient r_{surface} and traffic criteria coefficient r_{traffic} that are computed by surface feature value r'_{surface} and traffic feature value r'_{traffic} respectively. The total effect depends on all comfort related features associated with an edge:

$$r_{\text{surface}}((u,v)) = \max\{r'_{\text{surface}}(p) \mid p \in f((u,v))\}$$

$$r_{\text{traffic}}((u,v)) = \max\{r'_{\text{traffic}}(p) \mid p \in f((u,v))\}$$

Again, we take the maximum values of r'_{surface} and r'_{traffic} because we need to take into account the features that negatively affect the comfort the most. Specific values of r'_{surface} and specific values of r'_{traffic} for an example subset of features from *For bicycles* and *Motor roads* categories are shown in Table C.1 in Appendix C.

4.3 Heuristic-Enabled Multi-Criteria Label-Setting Algorithm

Our newly proposed *heuristic-enabled multi-criteria label-setting* (HMLS) algorithm extends the standard multi-criteria label-setting (MLS) algorithm [78] with several points for inserting speedup heuristic logic. The algorithm uses the following data structures: for each node $u \in V^C$, $L(u) := (u, (l_1(u), l_2(u), \ldots, l_k(u)), L^P(u))$ represents one of the *labels* at a node u, which is composed of the node u, the cost vector $l(u) = (l_1(u), l_2(u), \ldots, l_k(u))$ indicating the current cost values from the origin to the node u, and the predecessor label $L^P(u)$, which precedes $L(u)$ in an optimal route from an origin. A priority queue Q is used to maintain all labels created during the search. Since each node may be scanned multiple times, we define the bag structure $Bag(u)$ for each node u to maintain the non-dominated labels at u.

The pseudocode of the heuristic-enabled MLS algorithm is given in Algorithm 1. The speedups of the MLS algorithm can be implemented using up to three speedup-specific functions provided by the HEURISTIC interface: HEURISTIC.TERMINATIONCONDITION, HEURISTIC.SKIPEDGE, and HEURISTIC.CHECKDOMINANCE. The logic of the speedup functions is described in Section 4.3.1. Here we describe the overall heuristic-independent logic which consists of the following steps:

Step 1 – Initialisation: For a k-criteria optimisation problem, the algorithm first initialises the priority queue Q and Bag for each $v \in V^C$. Then it initialises the label at the origin to $L(o) := (o, (l_1(o), l_2(o), \ldots, l_k(o)), null)$, where $l_i(o) = 0$ for $i = 1, 2, \ldots, k$. Finally, it inserts the initial label $L(o)$ into the queue Q and the $Bag(o)$.

Step 2 – Label expansion: The algorithm extracts a minimum label $current := (u, (l_1(u), l_2(u), \ldots, l_k(u)), L^P(u))$ from the priority queue Q (in a lexicographic order of a cost vector). For each outgoing edge (u, v), the algorithm compute new cost vector $(l_1(v), l_2(v), \ldots, l_k(v))$ by adding the costs of the edge (u, v) to the current cost values $(l_1(u), l_2(u), \ldots, l_k(u))$. Then, it creates a new label $next$ using the node v, the cost vector $(l_1(v), l_2(v), \ldots, l_k(v))$ and the predecessor label $current$.

Function MLS.SKIPEDGE (cf. line 17 of Algorithm 1) prevents looping the path by checking the predecessor label in the label data structure, i.e., if previous node $L^P(u).getNode()$ is equal to the node v then the edge (u, v) is skipped. We have also experimented with checking the whole path for cycles. However, this significantly degraded the runtime of the algorithm. Therefore, we only check the predecessor label and cycles with larger size are eliminated through dominance checks.

Function MLS.CHECKDOMINANCE (cf. lines 20–23 of Algorithm 1 and Algorithm 6 in Appendix B), by default, controls dominance between the label $next$ and all labels inside $Bag(v)$. If $next$ is not dominated, the algorithm inserts it into $Bag(v)$ and Q. Also, if some label inside $Bag(v)$ is dominated by $next$, it is eliminated from the bag structure $Bag(v)$ and the priority queue Q, i.e., it will not be considered in future search.

Algorithm 1: Heuristic-enabled MLS algorithm.

Input: cycleway graph $G^C = (V^C, E^C, g, h, l, f, \overrightarrow{c})$,
origin node o, destination node d
Output: full Pareto set of labels

```
1  Q := empty priority queue
2  Bag(∀v ∈ V^C) := empty set
3  L(o) := (o, (0, 0, ..., 0), null)
4  Q.insert(L(o))
5  Bag(o).insert(L(o))
6  while Q is not empty do
7      current := Q.pop()
8      u := current.getNode()
9      (l_1(u), l_2(u), ..., l_k(u)) := current.getCost()
10     L^P(u) := current.getPredecessorLabel()
11     if Heuristic.terminationCondition(current) then
12         break
13     end
14     foreach edge (u, v) do
15         l_i(v) := l_i(u) + c_i(u, v) for i = 1, 2, ..., k
16         next := (v, (l_1(v), l_2(v), ..., l_k(v)), current)
17         if Heuristic.skipEdge(next) then
18             continue
19         end
20         if Heuristic.checkDominance(next) then
21             Bag(v).insert(next)
22             Q.insert(next)
23         end
24     end
25 end
26 return Bag(d)
```

Step 3 – Pruning condition: The algorithm exits if the queue Q becomes empty. Otherwise, it continues with *Step 2*.

After the algorithm has finished, the optimal Pareto set of routes Π^* is extracted. Let $|Bag(d)| = |\Pi| = m$. Then, from labels $L_1, \ldots, L_m$ in the destination Pareto set of labels $Bag(d)$, the routes $\pi_1, \ldots, \pi_m$ are extracted using the predecessor labels $L^P(\cdot)$ (see the pseudocode of EXTRACTROUTES method in Algorithm 5 of Appendix B). These routes comprise the set $\Pi^* = \{\pi_1, \pi_2, \ldots, \pi_m\}$ of optimal Pareto routes.

4.3.1 Speedups for the HMLS Algorithm

A significant drawback of the standard MLS algorithm is that it is very slow. The main parameter that affects the runtime of the algorithm is the size of the Pareto set. In general, the Pareto set can be exponentially large in the input graph size [81]. Furthermore, the MLS algorithm always explores the whole cycleway graph.

To accelerate the multi-criteria shortest path search, we introduce five speedup heuristics. Two of the heuristics are newly proposed by us: *ratio-based pruning* and *cost-based pruning*, while three are existing heuristics: *ellipse pruning*, *ϵ-dominance* and *buckets*. Implementation-wise, the heuristics are incorporated into the heuristic-enabled MLS algorithm by defining the respective three heuristic-specific functions used in Algorithm 1.

Ellipse Pruning

The first speedup heuristic taken from [56] prevents the MLS algorithm from always searching the whole cycleway graph, even for a short origin-destination distance[1]. The heuristic permits visiting only the nodes that are within a predefined ellipse. The focal points of the ellipse correspond to the journey origin o and the destination d. We maintain a constant ratio $\frac{a}{b}$ between semimajor axis a and semiminor axis b. In addition, to improve the ellipse performance for short origin-destination distances, we keep a minimum value $d'_{\min}$ for the length of $d' = |op|$ which is the distance between the origin o and a peripheral point p on the main axis of the ellipse, cf. Figure 4.4. During the search, in the ELLIPSE.SKIPEDGE function (cf. Algorithm 1, line 17), it is checked whether an edge (u, v) has its target node v inside the ellipse by checking the inequality $|ov| + |vd| \leq 2a$.

Ratio-Based Pruning

The ratio-based pruning terminates the search (long) before the priority queue gets empty (which means that the whole search space has been explored). A pruning ratio $\alpha \in \mathbb{R}^+$ is defined and the search is terminated when the chosen criterion cost value $l_i(u)$ in the *current* label exceeds α times the best so far value of the same criterion for a route that has already reached the destination (this is checked in the

[1] Note that in contrast with single-criterion Dijkstra's algorithm, the MLS algorithm does not stop when the destination node is first reached.

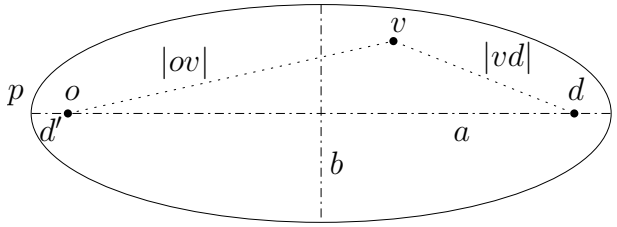


Figure 4.4: Geometry of the ellipse pruning condition.

RatioPruning.terminationCondition function, cf. Algorithm 7 in Appendix B and Algorithm 1, line 11). In this work, we choose the travel time criterion $l_1(u)$ since we do not want to consider plans with an excessive duration.

Cost-Based Pruning

The third heuristic does not include a label $L(v)$ in the bag of the node v in the situation when $L(v)$ has cost values similar to some label already located in the bag. To be specific, the CostPruning.checkDominance function (cf. Algorithm 8 in Appendix B and Algorithm 1, line 20) returns false when $L(v)$ is dominated by a label inside the bag or its cost-space Euclidean distance to some label inside the bag is lower than $\gamma \in \mathbb{R}^+$. Therefore, the search process is accelerated since fewer labels are inserted into the priority queue and the bag. We do not normalise the criteria values because it would be computationally expensive during the search process (without noticeable benefit).

ϵ-dominance

The fourth strategy we consider is to weaken the dominance checks so as to reduce the number of labels pushed through the network. We use the notion of ϵ-dominance defined in [89]. A newly generated label with a cost vector $\overrightarrow{c}$ is already dominated by the existing label $L(v) \in Bag(v)$ with a cost vector $\overrightarrow{l}$ if $\overrightarrow{l} \prec (1+\epsilon)\overrightarrow{c}$ and $\epsilon \in \mathbb{R}^+$ (see pseudocode of EpsilonDominance.checkDominance in Algorithm 9 of Appendix B). Similarly, any existing label satisfying $\overrightarrow{c} \prec (1+\epsilon)\overrightarrow{l}$ will be removed from the priority queue and the bag of node v.

Buckets

The last heuristic defined in [27] discretises the cost space using buckets for the criteria values. The heuristic is executed in the Buckets.checkDominance function (cf. pseudocode Buckets.checkDominance in Algorithm 10 of Appendix B and Algorithm 1, line 20). A function $bucketValue : \mathbb{R}_0^+ \times \cdots \times \mathbb{R}_0^+ \rightarrow \mathbb{N} \times \cdots \times \mathbb{N}$ is used to assign a real cost vector $\overrightarrow{l}$ an integer bucket vector $bucketValue(\overrightarrow{l})$. The function is implemented as $bucketValue(\overrightarrow{l}) := (l_1 - (l_1 \,\%\, bucketSize), \ldots, l_k - (l_k \,\%\, bucketSize))$ where % is the modulo operation.

4.4 HMLS Algorithm Evaluation

To evaluate our approach, we consider the real cycleway network of Prague. Prague is a challenging experiment location due to its complex geography and fragmented cycling infrastructure, which raises the importance of proper multi-criteria routing.

4.4.1 Experiment Settings

We evaluate our solution in two phases. In the first phase, we use three different cycleway graphs corresponding to three distinct neighbourhood-scale areas of the city of Prague (each of the areas covers approximately 10 km^2). We have chosen parts Prague A, Prague B, and Prague C to be different in terms of network density, nature of the cycling network, and terrain topology so as to evaluate the performance of heuristics across a range of conditions. The sizes of the evaluation graphs are described in Table 4.1. The specifics of each evaluation area are the following:

- Prague A: This graph covers a flat city centre area of the Old Town with many narrow cobblestone streets and Vinohrady with the grid layout of streets.
- Prague B: This graph covers a very hilly area of Strahov and Brevnov with many parks.
- Prague C: This graph covers residential areas of Liben and Vysocany further from the city centre. There are many good cycleways in this area.

All evaluation cycleway graphs are strongly connected. The size of the evaluation graphs allows us to run the standard MLS algorithm without any speedups – this is necessary for being able to compare the quality of heuristic and optimal solutions.

In the second scale-up phase, we evaluate the best performing heuristics from the first phase on the large city-scale graph of the whole Prague that has 69 544 nodes and 188 868 edges and covers the area of approximately 200 km^2. In this phase, we impose a 15 minutes limit for the search runtime for each origin-destination pair.

For each graph evaluation area, a set of origin-destination pairs generated randomly with a uniform spatial distribution, was used in the evaluation. First, we generated 100 origin-destination pairs for each of graphs Prague A, B, and C. The

Table 4.1: Graph sizes for the experiments.

Graph	**Nodes**	**Edges**	**Area**
Prague A	4 708	13 129	Old Town, Vinohrady
Prague B	3 343	8 915	Strahov, Brevnov
Prague C	5 164	14 072	Liben, Vysocany
Whole Prague	69 544	188 868	The whole city of Prague

minimum origin-destination distance is set to 500 m. The longest routes have approximately 4.5 km. We executed the MLS algorithm and the HMLS with all 15 heuristic combinations using the same generated 100 origin-destination pairs for each graph Prague A, B, and C. Therefore, each heuristic combination is evaluated on 300 origin-destination pairs. Then we generated a total of 100 route requests for the whole Prague graph where the origin-destination distance is set to be in the interval from 500 to 10 000 m.

The parameters in the cost functions were set as follows. The average cruising speed is $s = 14$ km/h, the penalty coefficient for uphill is $a_l = 13$ (according to the route choice model developed in the user study [14]), the maximum downhill speed multiplier is $s_{\mathrm{dmax}} = 2.5$, and the critical grade value is $d'_{\mathrm{c}} = 0.1$.

Configuration parameters for the heuristics were optimised so as to maximise the ratio between the algorithm runtime and the quality of the solution (see the next section), as measured on the three graphs Prague A, B, and C. Specifically, the following values were chosen: $\frac{a}{b} = 1.25$ for ellipse pruning, $\alpha = 1.6$ for ratio-based pruning, $\gamma = \frac{c_1}{5}$ for cost-based pruning (c_1 corresponds to a criteria value of the first criterion), $\epsilon = 0.05$ for ϵ-dominance, and $(15, 2500, 4)$ for buckets (the triple defines the buckets sizes for the three used criteria).

The results obtained in the experimental evaluation section are based on running the algorithm on a single core of a 2.4 GHz Intel Xeon E5-2665 processor of a Linux server. The source code of the HMLS algorithm and speedups for the multi-criteria bicycle routing is openly available in a repository[2] under LGPL license[3].

4.4.2 Evaluation Metrics

We consider two categories of evaluation metrics: *speed* and *quality*. We use the following to measure the algorithm speed:

- Average runtime in milliseconds for each origin-destination pair together with its standard deviation $\sigma_{\mathrm{runtime}}$.
- Average speedup over the standard, optimal MLS algorithm in terms of algorithm runtime.

For a multi-criteria optimisation problem, solution quality cannot be simply defined in terms of closeness to an optimal solution – instead, we define solution quality in set terms as the closeness to the full Pareto set. To our best knowledge, there is not a universal way to evaluate the quality of multi-criteria solutions. Therefore, we use the following metrics to measure the quality of returned routes in the multi-criteria bicycle routing problem:

- Average distance $d_c(\Pi^*, \Pi)$ of the heuristic Pareto set Π from the optimal Pareto set Π^* in the cost space. Distance $d_c(\pi^*, \pi)$ between two routes π^* and π is measured as the Euclidean distance in the unit three-dimensional space of criteria

[2] https://github.com/agents4its/cycleplanner/tree/mcspeedups
[3] http://www.gnu.org/licenses/lgpl.html

values normalised to the $[0, 1]$ range (min and max values of each criterion are calculated for all plans in the optimal Pareto set Π^* and the heuristic Pareto set Π; the min value is mapped to 0 and the max to 1).

$$d_c(\Pi^*, \Pi) := \frac{1}{|\Pi^*|} \sum_{\pi^* \in \Pi^*} \min_{\pi \in \Pi} d_c(\pi^*, \pi)$$

Intuitively, $d_c(\pi^*, \pi) = 0.1$ corresponds to a 6% optimality loss in each criterion, assuming the difference to optimum is distributed equally across all three criteria.

- Average distance $d_J(\Pi^*, \Pi)$ of the heuristic Pareto set Π from the optimal Pareto set Π^* in the physical space. Jaccard distance $d_J(\pi^*, \pi)$ [74] is used to measure the dissimilarity between Pareto routes. For routes π^* and π, i.e., sequences of edges, the physical distance is computed by dividing the difference of the sizes of the union and the intersection of the two route sets by the size of their union. Reasonably, the physical distance definition for routes obeys the triangle inequality.

$$d_J(\pi^*, \pi) := \frac{|\ (\pi^* \cup \pi)\ | - |\ (\pi^* \cap \pi)\ |}{|\ (\pi^* \cup \pi)\ |}$$

$$d_J(\Pi^*, \Pi) := \frac{1}{|\Pi^*|} \sum_{\pi^* \in \Pi^*} \min_{\pi \in \Pi} d_J(\pi^*, \pi)$$

- Average number of routes $|\Pi|$ in the Pareto set Π together with its standard deviation $\sigma_{|\Pi|}$.

- The percentage of Pareto routes $\Pi_{\%}$ in heuristic Pareto set Π that are equal to routes in the optimal Pareto set Π^*. For instance, if there are 10 routes in the heuristic Pareto set Π and 6 routes are optimal, the percentage $\Pi_{\%} = 60\%$.

4.4.3 Results for Graphs Prague A, B, and C

Table 4.2 summarises the evaluation of the HMLS algorithm and its heuristics using the neighbourhood-scale graphs Prague A, B, and C. The MLS algorithm is used as a baseline for the evaluation of the proposed heuristics and their combinations. Columns d_c, d_J, and $\Pi_{\%}$ are calculated with respect to the optimal Pareto set Π^* returned by the MLS algorithm. The MLS algorithm returns optimal solutions (1 647 routes in the Pareto set on average) at the expense of a prohibitively high runtime.

As anticipated, all heuristic methods are significantly faster than the MLS algorithm. First, we have compared the 15 evaluated methods using the following metrics: the average runtime (from the speed category) and the average distance $d_c(\Pi^*, \Pi)$ in the cost space (from the quality category). From the perspective of these two metrics, there are *nine non-dominated combinations* of heuristics, cf. filled-in bars in

Table 4.2: Evaluation of the heuristic performance on three graphs Prague A, B, and C. Primary speed and quality metrics are marked by bold column headings. Non-dominated heuristic combinations with respect to primary speed and quality metrics are denoted by bold font.

Heuristic	Speedup	**Runtime [ms]**	σ_{runtime}	$\lvert\Pi\rvert$	$\sigma_{\lvert\Pi\rvert}$	$\mathbf{d_c}$	d_J	$\Pi_{\%}$
MLS	-	898 101	1 103 097	1 647	2 390	-	-	100.00
HMLS+Buckets	585	**1 536**	1 296	40	44	**0.138**	0.323	55.25
HMLS+Cost	129	6 946	5 421	93	63	0.178	0.333	54.66
HMLS+Epsilon	4490	**200**	131	9	7	**0.195**	0.420	54.61
HMLS+Ratio	3	**294 458**	538 357	1 173	1 690	**0.054**	0.076	99.89
HMLS+Ratio+Buckets	1 415	635	818	35	37	0.173	0.347	58.81
HMLS+Ratio+Cost	284	3 161	3 044	80	51	0.211	0.363	58.41
HMLS+Ratio+Epsilon	7 302	123	98	8	5	0.233	0.449	56.32
HMLS+Ellipse	2	**434 170**	956 062	1 637	2 390	**0.005**	0.008	99.98
HMLS+Ellipse+Buckets	1 801	**499**	853	40	44	**0.142**	0.327	55.27
HMLS+Ellipse+Cost	293	3 069	3 897	93	63	0.179	0.334	54.79
HMLS+Ellipse+Epsilon	9 657	**93**	104	9	7	**0.199**	0.423	54.64
HMLS+Ellipse+Ratio	4	**241 675**	598 290	1 171	1 691	**0.056**	0.080	99.86
HMLS+Ellipse+Ratio+Buckets	2 183	**411**	715	35	37	**0.175**	0.349	58.77
HMLS+Ellipse+Ratio+Cost	427	2 103	2 830	80	51	0.211	0.363	58.52
HMLS+Ellipse+Ratio+Epsilon	11 087	**81**	84	8	5	**0.232**	0.448	56.27

Figure 4.5 and bold values in Table 4.2. In the following, we only discuss the non-dominated combinations of heuristics.

The *HMLS+Ellipse* heuristic performs best in terms of the quality of the solution. It successfully prunes the search space with $d_c(\Pi^*, \Pi) = 0.005$, i.e., approximately 0.3% optimality loss. The average runtime of this heuristic is around seven minutes. This heuristic is very good for combining with other heuristics, it offers double speedup over the MLS algorithm with a negligible quality loss (99.98% of the routes in the heuristic Pareto set Π are equal to the ones in the optimal Pareto set Π^*).

The *HMLS+Ratio* and *HMLS+Ellipse+Ratio* heuristics offer very good quality with $d_c(\Pi^*, \Pi) = 0.054$ and $d_c(\Pi^*, \Pi) = 0.056$ respectively. Adding the ellipse pruning heuristic cuts the runtime from approximately 5 to 4 minutes while keeping the quality of the solution very similar. The search space is pruned geographically by the ellipse pruning and the search is also terminated sooner by the ratio-based pruning method.

With a small decrease of the solution quality to $d_c(\Pi^*, \Pi) = 0.138$, *HMLS+Buckets* heuristic offers a significant additional speedup in average runtime to approximately 1.5 seconds. This makes this heuristic (and also the following ones) usable for real time applications, e.g., a web-based bicycle journey planner. When the ellipse pruning method is combined with the buckets, the average runtime of *HMLS+Ellipse+Buckets* is lowered to approximately 500 ms while keeping very similar quality $d_c(\Pi^*, \Pi) = 0.142$. When the ratio pruning method is added, the average runtime of *HMLS+Ellipse+Ratio+Buckets* is lowered to approximately 400 ms while the quality is decreased to $d_c(\Pi^*, \Pi) = 0.175$.

With an additional worsening of the solution quality to $d_c(\Pi^*, \Pi) = 0.195$, *HMLS+Epsilon* heuristic offers serious speedup in average runtime to approximately 200 milliseconds. When the ellipse pruning method is added to the ϵ-dominance heuristic, the average runtime of *HMLS+Ellipse+Epsilon* is lowered to approximately 93 ms while keeping almost the same quality $d_c(\Pi^*, \Pi) = 0.199$.

The last combination *HMLS+Ellipse+Ratio+Epsilon* performs best in terms of average runtime which is approximately 81 ms, i.e., it has four orders of magnitude speedup over the MLS algorithm. The quality of this combination is reflected by higher $d_c(\Pi^*, \Pi) = 0.232$, still over 56% of the routes in the heuristic Pareto set Π are equal to the ones in the optimal Pareto set Π^*.

To provide a deeper insight in search runtimes, we show in Figure 4.6 how the runtime of the *HMLS+Ellipse+Epsilon* heuristic depends on the direct origin-destination distance. The runtime increases with the origin-destination distance since larger search space needs to be explored. This also explains the high σ_{runtime} in Table 4.2. Despite the increase, our scale-up experiments in Section 4.4.4 resulted in less than 10 second response times even for 20 times larger cycleway graph covering the whole city of Prague (approximately 200 km^2).

To get further insight in the sizes of generated Pareto sets, we observe that the average number of Pareto routes increases notably with the direct origin-destination distance. In addition, the network structure around the origin and destination also affects the number of Pareto routes. The number of routes in the optimal Pareto set

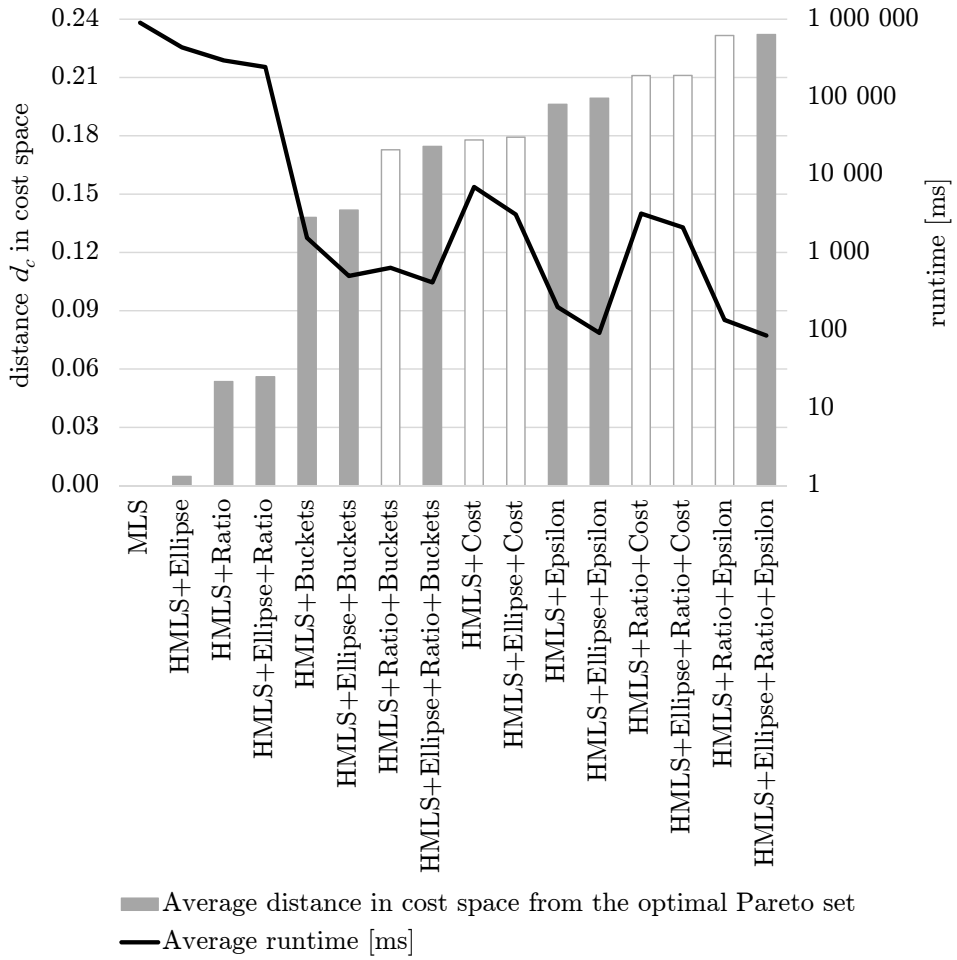


Figure 4.5: Speed and quality for the HMLS algorithm and all heuristic combinations sorted by the quality from the best (MLS on the left hand side) to the worst. Non-dominated heuristic combinations have grey filled-in bars.

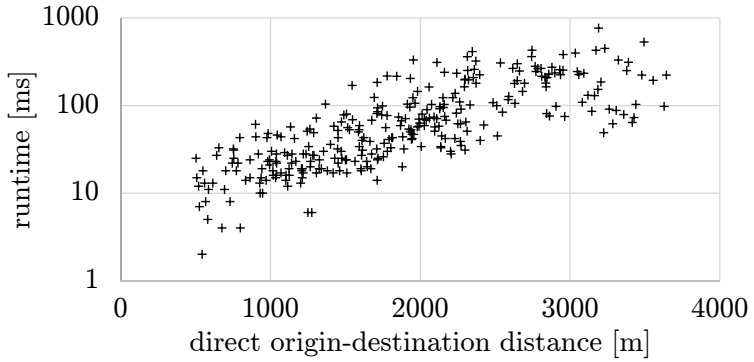


Figure 4.6: The runtime of HMLS+Ellipse+Epsilon in milliseconds in dependency on the direct origin-destination distance.

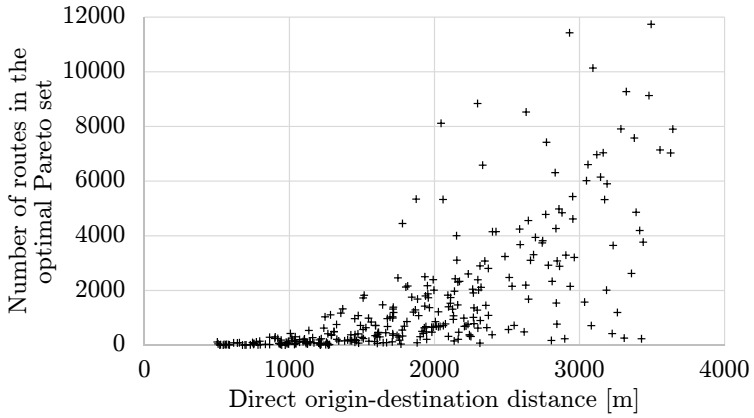
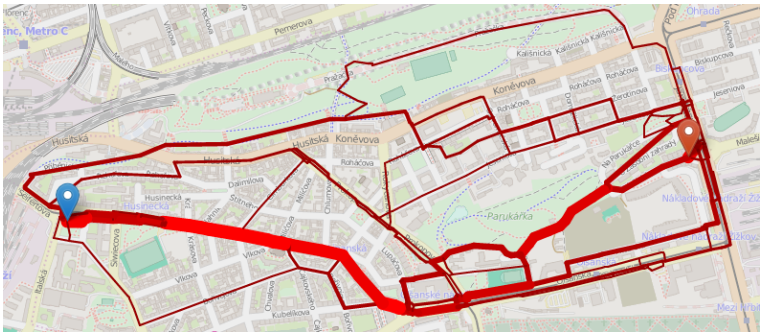


Figure 4.7: The number of routes in the optimal Pareto set $|\Pi^*|$ in dependency on the direct origin-destination distance.

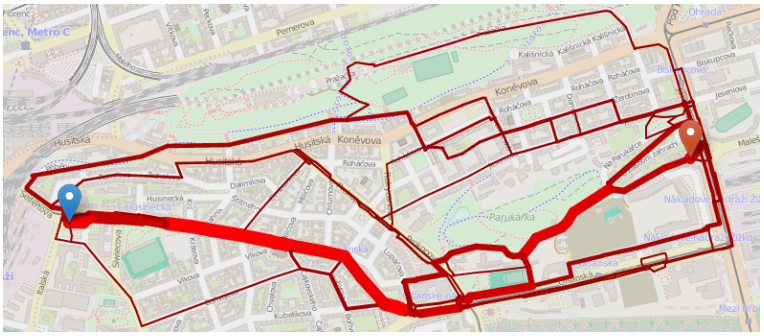
$|\Pi^*|$ in dependency on the direct origin-destination distance is shown in Figure 4.7.

The Pareto set can get very large. Therefore, a route selection method needs to be implemented. It is an independent step for which various methods can be used. In fact, we have already proposed a method based on clustering [103]. Regardless of the selection method, the quality of the unfiltered results is essential as high-quality solutions can be selected only if they are included in the unfiltered results set.

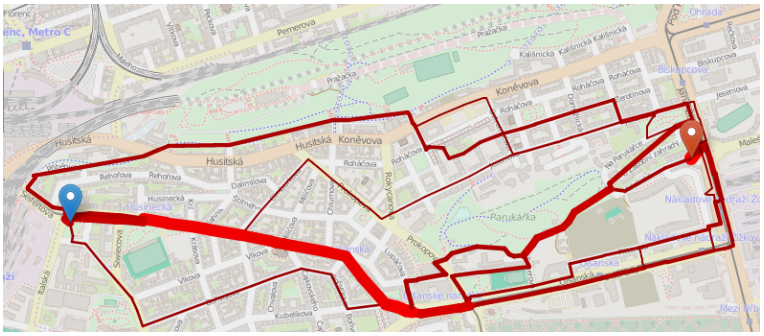
Finally, we have chosen a hilly area in Zizkov to illustrate the Pareto set of routes in the physical space. In Figure 4.8, we illustrate the route distribution of the Pareto sets returned by the MLS and HMLS algorithm with three different heuristic combinations. Each subfigure is provided with the name of the heuristic combination, the size of the Pareto set (ranges from 532 to 6 routes) and the algorithm runtime (ranges from 90 seconds to 372 ms). The more routes use a given cycleway network segment, the wider is the depicted line. It can be observed that the heuristics return a Pareto set of routes that very well corresponds to the optimal Pareto set.



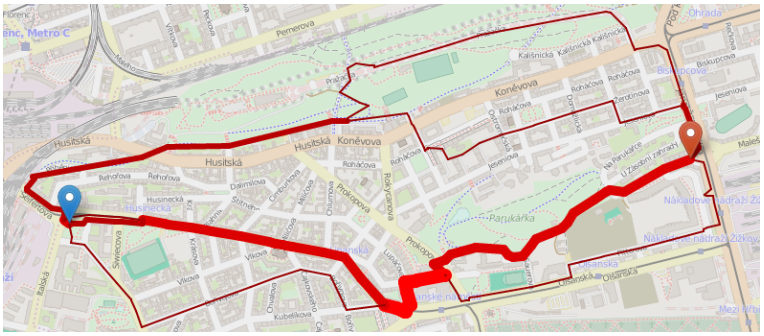
(a) MLS, optimal Pareto set with 532 routes, 90 s



(b) HMLS+Ellipse+Ratio, 500 routes, 32 s



(c) HMLS+Ellipse+Buckets, 26 routes, 623 ms



(d) HMLS+Ellipse+Ratio+Epsilon, 6 routes, 372 ms

Figure 4.8: Pareto sets for the MLS and HMLS algorithms.

To summarise, we have evaluated 15 different combinations of heuristics from which 9 combinations dominated the others in terms of quality and speed. The heuristics offer significant *one to four orders of magnitude speedup* over the MLS algorithm in terms of average runtime. The speedup is achieved by lowering the number of iterations and also the number of dominance checks in each iteration. *HMLS+Ellipse* is the best heuristic in terms of quality of the produced Pareto set while *HMLS+Ellipse+Ratio+Epsilon* is the best heuristic in terms of average runtime. Taking into the account the trade-off between the quality of a solution and the provided speedup, we consider *HMLS+Ellipse+Epsilon* heuristic to have the best ratio between the quality and speed.

4.4.4 Scale-up Results for the Whole Prague Graph

In order to show the scalability of the proposed heuristics to city-scale routing, we evaluate the heuristics on the graph of the whole Prague city which is approximately 20 times bigger than each of the graphs Prague A, B, and C. The results are summarised in Table 4.3. We present only a subset of statistics (compared to Table 4.2) since the graph is too large to get an optimal Pareto set by the MLS algorithm. We show the total number of requests finished in the imposed 15-minutes limit, average runtime in milliseconds, and average size of the Pareto set.

We have evaluated all combinations of heuristics; they offer average runtimes between 5 and 260 seconds. Eight evaluated heuristic combinations successfully return

Table 4.3: Evaluation of the heuristic performance on the whole Prague graph.

Heuristic	15 min	Runtime [ms]	σ_{runtime}	$\lvert\Pi\rvert$
HMLS+Buckets	0	-	-	-
HMLS+Cost	100	70 734	25 311	72
HMLS+Epsilon	100	15 759	4 859	12
HMLS+Ratio	14	168 302	250 902	402
HMLS+Ratio+Buckets	51	176 810	225 504	126
HMLS+Ratio+Cost	100	49 948	29 421	69
HMLS+Ratio+Epsilon	100	8 847	6 512	11
HMLS+Ellipse	18	260 278	313 214	918
HMLS+Ellipse+Buckets	72	128 096	194 328	244
HMLS+Ellipse+Cost	100	33 474	25 008	72
HMLS+Ellipse+Epsilon	100	**5 306**	5 667	11
HMLS+Ellipse+Ratio	20	112 333	159 589	768
HMLS+Ellipse+Ratio+Buckets	76	130 738	206 001	242
HMLS+Ellipse+Ratio+Cost	100	32 799	25 235	69
HMLS+Ellipse+Ratio+Epsilon	100	**4 834**	5 159	11
NAMOA*+TC	100	106 436	192 884	2 984

solutions for all 100 origin-destination pairs in the 15-minutes time limit. We can observe the behaviour of the ellipse pruning heuristic that does not significantly decrease the size of the Pareto set yet offers solid speedup consistent with the results on the graphs Prague A, B, and C. The best performing combinations of heuristics are *HMLS+Ellipse+Epsilon* and *HMLS+Ellipse+Ratio+Epsilon* with practically usable average runtime of 5 seconds. This result is consistent with our evaluation of the heuristics on the graphs Prague A, B, C where the two heuristics also performs best in terms of average runtime.

Finally, we compared the HMLS algorithm with heuristic speedups to the optimal NAMOA* algorithm [76] with the Tung & Chew (TC) heuristic [111]. The TC heuristic works as a preprocessing step where for each criterion a backward Dijkstra's algorithm is executed from the destination to all nodes in the graph. The preprocessing calculates the values of a perfect heuristic function, i.e., true cost values from each node to the destination, for the NAMOA* algorithm. Using the whole Prague graph, the preprocessing takes 444 ms on average. The NAMOA* algorithm with the TC heuristic successfully returned solutions for all 100 origin-destination pairs; the total runtime of the algorithm was 106 seconds on average. On the runtime side, the algorithm is better than 6 combinations of heuristics. However the best performing heuristic combination *HMLS+Ellipse+Epsilon* is 50 times faster than the NAMOA* algorithm with TC heuristic. On the quality side, the algorithm outperformed HMLS algorithm with each heuristic combination since it delivered optimal Pareto sets of routes (2 984 on average). Altogether, for the problems where there is not necessary to deliver the whole Pareto set of solutions (e.g., multi-criteria bicycle routing problem) the *HMLS+Ellipse+Epsilon* and *HMLS+Ellipse+Ratio+Epsilon* deliver 50 times better runtime than the NAMOA* algorithm with TC heuristic.

To summarise, we are able to solve multi-criteria bicycle routing problem on the large graph instance with tens of thousands of nodes and edges while achieving practically usable runtimes of 5 seconds.

4.5 Validation in Real-World Deployments

To fulfil Research objective 3, we have performed validation using real-world deployments. We think that deploying our approach in the operational environment (TRL 7[4]) is the way to find out the issues that needs to be solved to use the approach in practise. These important issues cannot be found at the stage of algorithm design/evaluation. Furthermore, it is a way to discover open research questions and next research steps. This is a place where two disciplines – research and engineering – meets. Based on real-world deployments, we have acquired useful experience which is common with other problems solved in this thesis so we have concentrated the lessons learned in Chapter 7. In the remaining part of this section, we describe details that are specific for the deployment of bicycle routing system.

[4] http://ec.europa.eu/research/participants/data/ref/h2020/wp/2014_2015/annexes/h2020-wp1415-annex-g-trl_en.pdf

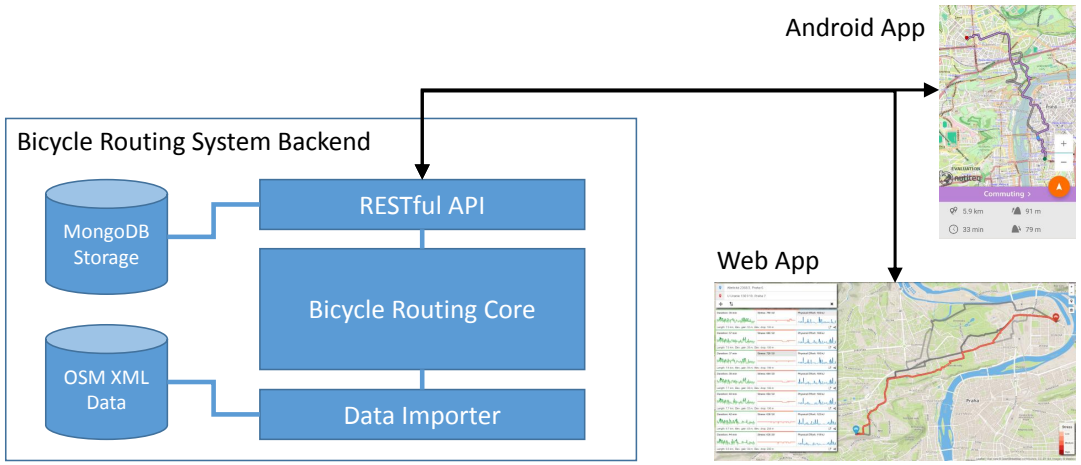


Figure 4.9: Architecture of the bicycle routing system.

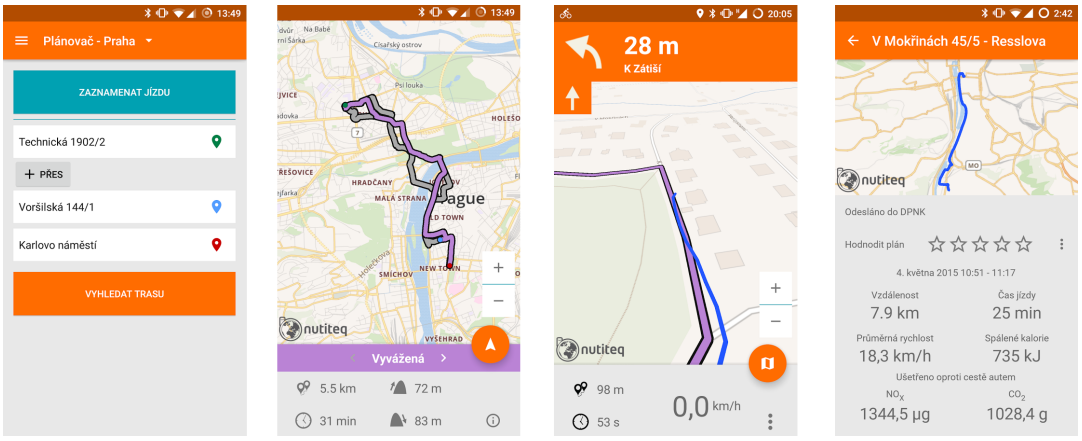


Figure 4.10: Android App frontend for the bicycle routing system allows finding bicycle routes, navigate to a destination, and track journeys (developed by Jan Linka).

To begin with, we have deployed a simplified version of the bicycle routing algorithm as a *bicycle routing system* with an open RESTful API. Architecture of the system is presented in Figure 4.9. The system consists of two main parts. The first part is the *bicycle routing system backend* that is primarily responsible for finding the Pareto set of routes based on the origin-destination pairs. At the backend, Data Importer builds the cycleway graphs from OSM XML files. The graphs are then used by the Bicycle Routing Core component that performs the routing. Finally, the communication with the backend is managed through the open RESTful API that uses a NoSQL MongoDB database to store found routes, feedback from cyclists, and tracked bicycle journeys. An efficient implementation of the bicycle routing system backend has been carried out by Pavol Žilecký in his master thesis (supervised by the author of the thesis) [118]. More details about backend internal workings and about

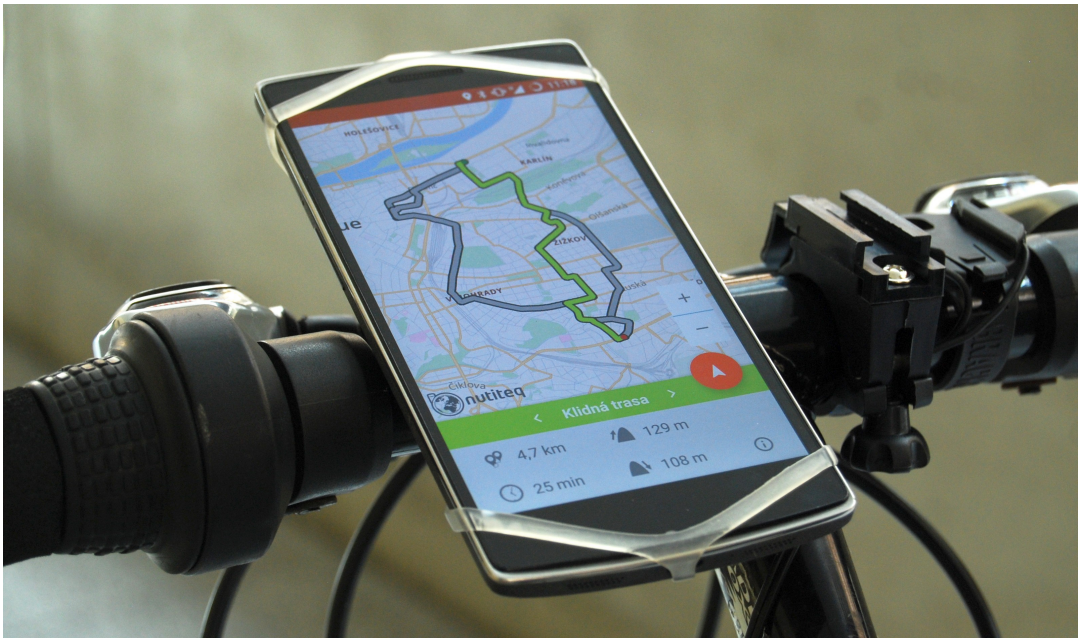


Figure 4.11: Smartphone with our app mounted on bicycle handlebars helps users with the navigation in cities which is often challenging due to fragmented cycling infrastructure.

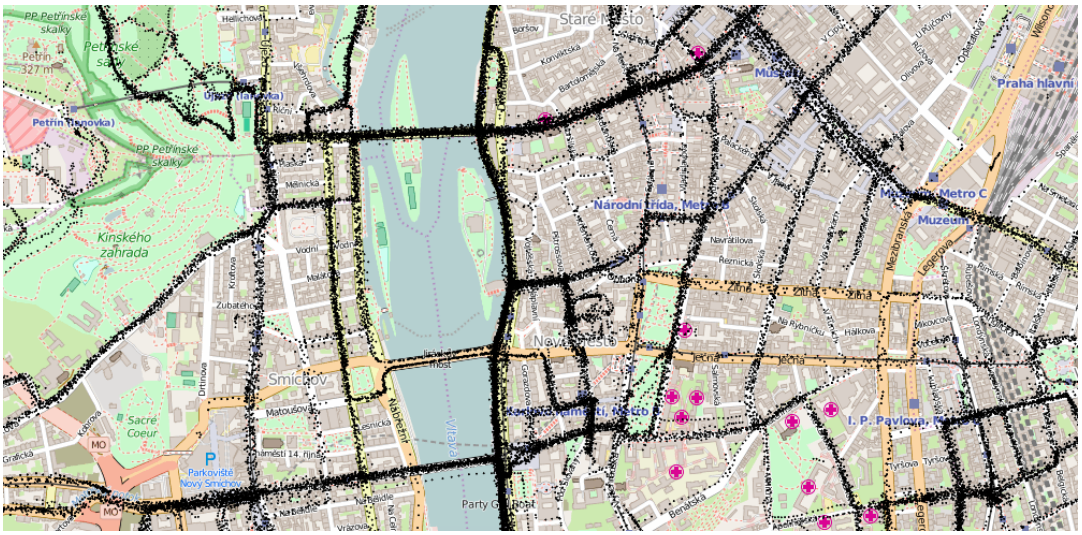


Figure 4.12: Tracked GPS points in the centre of Prague.

the API can be found in his thesis.

The second part of the system is composed from the *Android App* and *Web App* frontends. On the one hand, the Android App has been developed by Jan Linka in his bachelor thesis (supervised by the author of the thesis) [75] with a valuable

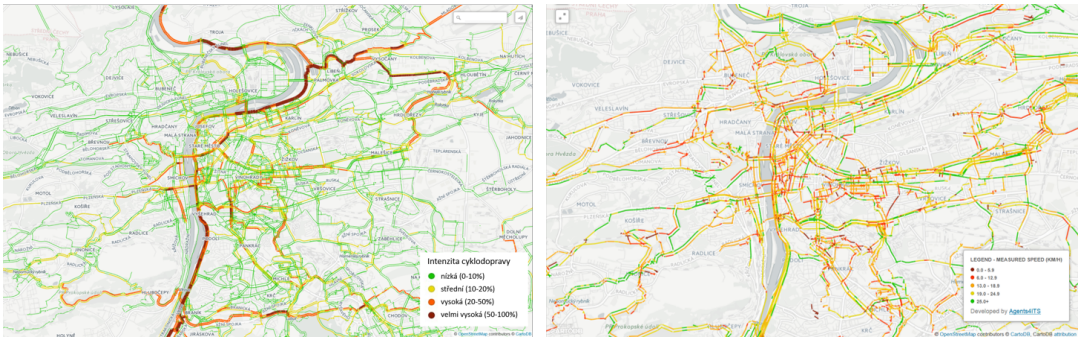


Figure 4.13: Cycling traffic intensity map on the left and cycling speed map on the right (developed by Filip Langr).

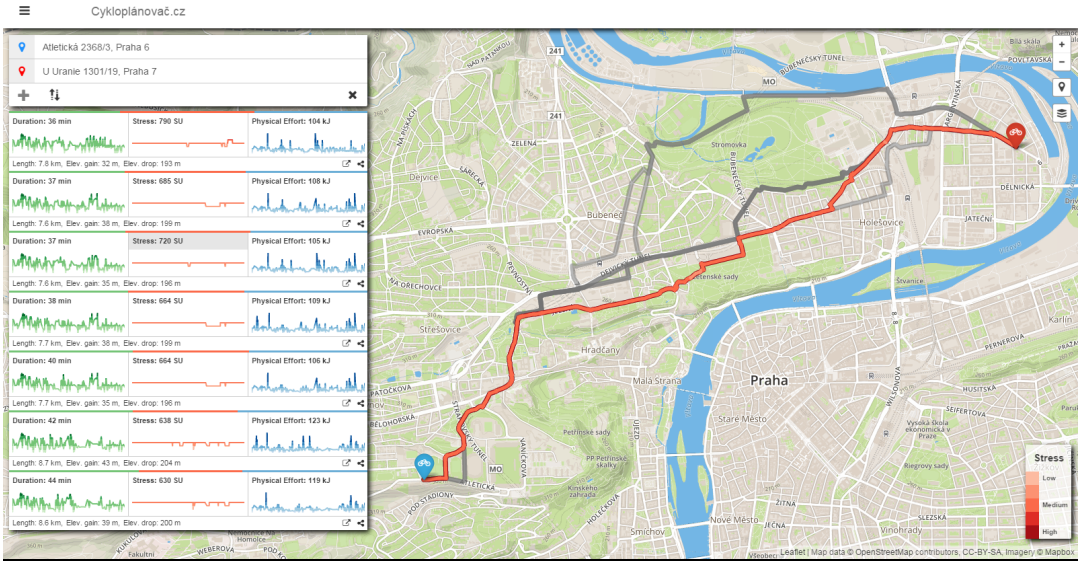


Figure 4.14: Web App frontend for the bicycle routing system (developed by Tomáš Fišer).

feedback from experts on cycling from AUTO*MAT NGO and is freely available on Play Store[5] where it has more than 5 000 installations. The application is able to find bicycle routes for a cyclist based on her origin and destination, cf. Figure 4.10. When a cyclist uses a holder to attach a smartphone to the handlebar of a bicycle, the app is able to navigate the cyclist from an origin to a destination, cf. Figure 4.11. When a cyclist detours from his or her route, the app automatically replans a route to the destination. Finally, the app is able to track journeys of cyclists. This has been successfully used during the "Do práce na kole" (Bike2Work) competition in May 2015 when around 2 000 tracks has been collected in Prague by the participants of

[5] https://play.google.com/store/apps/details?id=cz.agents.cycleplanner

the competition. The tracked data are depicted in Figure 4.12. The GPS tracked data has been mapped to the cycling network by Daniel Slunečko in his bachelor thesis (supervised by the author of the thesis) [102] and then a map of cycling traffic intensity and cycling speed has been created by Filip Langr, see Figure 4.13. Importantly, we plan to incorporate the tracked journeys into the routing process (see Section 8.1) to improve the quality of route suggestions. Furthermore, the tracked journeys can be used by the municipality to improve the quality of the infrastructure for cyclists such as adding new cycle lanes in streets that are frequently used by the cyclists. On the other hand, the Web App has been developed by Tomáš Fišer and is freely available[6]. It is able to show the found set of routes and also their quality in terms of criteria, cf. Figure 4.14.

4.6 Contributions and Summary

In this chapter, we have investigated a multi-criteria approach to urban bicycle routing. Our contributions with respect to state-of-the-art techniques described in Section 2.2 are summarised as follows.

In contrast to existing work, we have provided a well-grounded formal model of multi-criteria bicycle routing and have applied a novel heuristic-enabled multi-criteria shortest path algorithm to find a diverse set of cycling routes. As a result, we have made bicycle routing that properly considers multiple realistic route choice criteria fast enough for practical, interactive use. Our method produces routes closely approximating the full Pareto set in hundreds of milliseconds when routing on a neighbourhood scale and in seconds when routing on a city-wide scale. Further speedups are possible through low-level optimisation of data structures and the algorithmic logic.

We have achieved these results by employing five heuristic speedup techniques for multi-criteria shortest path search. The speedup heuristics provide a variable trade-off between the search time and the completeness and quality of the suggested routes and they enable fast response times without severely compromising the quality of the results. Although the heuristics are relatively simple, their application in the context of bicycle routing is novel and their effect is very significant. Because real-world speedup performance may differ significantly from the performance on general multi-criteria shortest path problems, our work is important for properly understanding realistic performance trade-offs in developing efficient bicycle routing algorithms.

We have evaluated our approach extensively in the challenging conditions of the city of Prague, which features complex geography and fragmented cycling infrastructure. The evaluation has confirmed the usefulness of the multi-criteria approach to bicycle routing. To conclude, the bicycle routing system has been validated in real-world deployments via a RESTful API used by Android App and Web App frontends. The system has been discussed with the experts on cycling from Auto*Mat NGO and has been used by hundreds of cyclists in the Bike2Work competition in May 2015.

[6] http://its.felk.cvut.cz/cycleplanner2/

The multi-criteria bicycle routing algorithm has been devised in collaboration with Pavol Žilecký and Qing Song. Pavol has efficiently implemented the HMLS algorithm including speedups and executed the experiments. He has also created a simple visualisation tool used in Figure 4.8 and contributed to the description of the HMLS algorithm in Section 4.3 and pseudocodes of speedups in Appendix B. Qing has contributed to related work regarding multi-criteria bicycle routing in Section 2.2 and definition of the bicycle routing problem in Section 4.1.1. She has also collaborated on the design of ratio-based pruning and cost-based pruning heuristics. Finally, this topic is covered by the following articles [66, 67, 68, 103].

Chapter 5

Intermodal Journey Planning

The advent of *new types of mobility services*, such as bike, scooter, and car sharing, real-time carpooling or next-generation taxi, has further expanded the already rich portfolio of mobility means available in modern cities. Providing intelligent tools that would help citizens make the best use of mobility services on offer is thus needed more than ever [92]. Despite recent algorithmic advances [5], existing planners available in practise, such as Google Maps[1] or Here.com[2], address this need only partially. In particular, they only consider a limited subset of transport modes and their combinations, and they only provide limited ways for users to express their travel preferences.

Based on the motivation above, we progress to study the single-agent single-criteria *intermodal routing problem*. The problem is most often formalised as the earliest arrival problem (EAP), i.e., the problem of finding the earliest arrival at a destination given a departure date and time from an origin. The earliest arrival problem has been widely studied and numerous algorithms and speedup techniques exist for solving it on road network graphs and networks of public transport (PT) services. Until recently, little work has been done on solving the earliest arrival problem for journey plans allowing general combinations of individual and public transport modes, the work of Horn [60], and Yu and Lu [117] being notable exceptions. However, in parallel to our research, several approaches aiming to solve the intermodal routing problem have emerged (Delling et al. [27], Gundling et al. [54], and Kirchler [73]). This means that this problem has become topical.

In our work, we focus on solving the *intermodal* variant of the EAP. We use the term intermodal in order to stress that we consider modes and combinations thereof that go beyond what is supported in existing multimodal journey planners. Our main contribution with respect to state-of-the-art techniques described in Section 2.4 lies in a representation-centric approach to solving the intermodal EAP. Thus, instead of providing complex, purpose-specific journey planning algorithms, we have introduced a *generalised time-dependent graph* (see Section 3.2.2) that allows representing the intermodal EAP as a standard graph search problem and consequently use general

[1] http://maps.google.com
[2] http://here.com

shortest path algorithms to solve it. What is more, studying this approach further on lead to related work on PT network analysis [84, 85] and on integration of multiple journey planners [83].

We treat the problem in a deterministic setting assuming no uncertainty in any of the attributes of the planning graph. In addition to the generalised time-dependent graph representation, we also introduce the concept of journey plan templates. Journey plan templates provide a powerful way of parameterising the operation of the planner and allow the user or the administrator of the journey planner to obtain plans that best meet their constraints and preferences. Furthermore, the templates constrain the search space and therefore speed up journey planning.

5.1 Intermodal Earliest Arrival Problem

In this section, we provide the formalisation of the intermodal earliest arrival problem. The intermodal journey planner (IJP) should be able to plan journeys which use any combination of a full range of transport modes, including scheduled PT (e.g., bus, underground, tram, train, ferry), individual transport (e.g., walk, bike, shared bike and car), and on-demand transport (e.g., taxi). The term *intermodal* emphasises the fact that the planner should be able to plan a journey using a combination of a full range of transport modes, as compared to existing planners that are only able to handle a subset of transport modes and their combinations (e.g., a cycle planner, a scheduled PT planner).

Our approach to solving the intermodal EAP relies on the newly proposed generalised time-dependent (GTD) graph defined in Section 3.2.2, which allows representing the combined road network (for individual and on-demand modes) and PT network (for PT modes) in a single structure. Below, we first describe the notions of a journey leg and a journey plan. Then, we define two variants of the intermodal earliest arrival problem.

5.1.1 Journey Leg and Journey Plan

Let the journey leg be a part of a journey plan that is either covered by the traveller on foot or by a movement by one and only one vehicle from one location to another. Formally, the *journey leg* $L = ((v_1, w_1), \ldots, (v_k, w_k))$ is defined as a sequence of $|L| = k$ edges $(v_j, w_j) \in E$. Edges are a finer-grained decomposition of a journey leg and represent the lowest-level, atomic parts of any journey plan. Then the *journey plan* is a quadruple $\pi = (P, \sigma, \phi, \psi)$:

- $P = (L_1, \ldots, L_n)$ is a sequence of $|P| = n$ journey legs L_i.
- Function σ denotes the mode of transport $\sigma(L_i) \in M$ that is used for journey leg L_i.
- Function $\phi : E \to \mathbb{N}$ returns the departure time from v for each edge $(v, w) \in E$.

- Function $\psi : E \to \mathbb{N}$ returns the arrival time at w for each edge $(v, w) \in E$.

Let $L[j]$ be the j-th element of a sequence of elements L and $|L|$ be the number of elements in L. Let $\xi(P)$ be the *flattened plan* constructed as the concatenation of all edges in all journey legs $L_i \in P$:

$$\xi(P) = (L_1[1], \ldots, L_1[|L_1|], \ldots, L_n[1], \ldots, L_n[|L_n|])$$

5.1.2 Intermodal Earliest Arrival Problem

The *intermodal earliest arrival problem* is a pair $J = (G, r)$, where:

- $G = (V, E, \rho, \mu, \chi)$ is a *GTD graph*
- $r = (o, d, t)$ is a *journey request* specifying an origin $o \in V$, a destination $d \in V$, and a time of departure $t \in \mathbb{N}$

A *journey plan* $\pi = (P, \sigma, \phi, \psi)$, where $P = (L_1, \ldots, L_n)$, is then a solution of the intermodal earliest arrival problem $J = (G, r)$ if and only if all the following conditions hold:

1. Journey plan starts at the origin:
   $o = v$ where $(v, w) = L_1[1]$
2. Journey plan ends at the destination:
   $d = w$ where $(v, w) = L_n[|L_n|]$
3. All edges are present in the GTD graph:
   $\forall (v, w) \in \xi(P) : (v, w) \in E$
4. Edges form a path in the GTD graph:
   $\forall j \in \{1, \ldots, |\xi(P)| - 1\} :$
   $(v, w) = \xi(P)[j] \wedge (w, x) = \xi(P)[j + 1]$

5.1.3 Intermodal EAP with Templates

For the intermodal earliest arrival problem with templates, we introduce the notion of a *journey plan template*. As mentioned in the introduction, journey plan templates give users and journey planner administrators a powerful way of parameterising the journey planner to obtain plans that best meet their constraints and preferences. For instance, a journey plan template that prefers environmentally friendly modes of transport can be designed by a journey planner administrator (e.g., a combination of walk and shared bike).

A journey plan template constrain the journey plan in the permitted combination of modes on the level of journey legs. A *journey plan template* τ is defined as a regular expression over the transport modes alphabet M. As an example, we list three templates[3]:

[3]POSIX Extended Regular Expression syntax is used.

- Taxi only: `^X$`
- Walk and PT: `^W((B|T|U)W)*$`
- Walk and shared bike: `^W(SW)?$`

We define several notions related to the journey plan templates. Let the word $\sigma(L_1)\dots\sigma(L_n)$ be the *mode sequence* $\kappa(P)$ of a sequence of journey legs $P = (L_1, \dots, L_n)$. Empty mode sequence $\kappa(\emptyset) = \epsilon$. We say that a sequence of journey legs P match a journey plan template τ if and only if the mode sequence $\kappa(P)$ matches the regular expression τ. Next, let $modes(\tau)$ be the set of modes of transport that are present in a template τ. Finally, the binary operator $\|$ over a mode sequence $m_1 \dots m_n$ and a mode of transport $m \in M$ is defined as follows:

$$m_1 \dots m_n \| m := \begin{cases} m_1 \dots m_n & \text{if } m = m_n \\ m_1 \dots m_n m & \text{otherwise} \end{cases}$$

The intermodal EAP with templates adds the notion of journey plan template to the intermodal EAP. Thus, the *intermodal earliest arrival problem with templates* is a triple $J = (G, r, \tau)$, where:

- $G = (V, E, \rho, \mu, \chi)$ is a *GTD graph*
- $r = (o, d, t)$ is a *journey request*
- τ is a *journey plan template*

A *journey plan* $\pi = (P, \sigma, \phi, \psi)$ is then a solution of the intermodal EAP with templates $J = (G, r, \tau)$ if and only if all the following conditions hold:

1. Journey plan π is a solution of the intermodal earliest arrival problem $J = (G, r)$.
2. Journey legs $P = (L_1, \dots, L_n)$ match the journey plan template, i.e., $\sigma(L_1) \dots \sigma(L_n)$ matches τ.

5.2 Intermodal Planning Algorithm

We present a solution method to solve the intermodal earliest arrival problem with templates using the GTD graph representation (defined in Section 3.2.2). The method uses a contextual view over the underlying GTD graph in order to use general shortest path algorithms to find the journey plans in the search space. This is enabled by storing the node context, i.e., the time of arrival and the modes of transport sequence used, in the contextual GTD graph.

5.2.1 Contextual GTD Graph

The contextual GTD graph is a *view* over an underlying GTD graph. The contextual GTD graph serves two main purposes. First, it allows filtering the available edges in the GTD graph with respect to the permitted modes of transport specified by a given journey plan template τ. Second, it allows checking that the current partial journey plan matches a given journey plan template τ during the search process.

Let us define the graph formally. The *contextual GTD graph* G_τ over a GTD graph $G = (V, E, M, \rho, \mu, \chi)$ using a journey plan template τ is defined as $G_\tau = (V_\tau, E_\tau, M, \rho, \mu, \chi, \lambda)$. V_τ is a set of *contextual nodes* defined as triples (v, t_a, m_s) where:

- $v \in E$ is a node in the GTD graph
- $t_a \in \mathbb{N}$ is the arrival time at v
- m_s is a mode sequence $\kappa(P')$ of a sequence of journey legs P' from origin o (taken from the input journey request r) to node v

The context of contextual nodes corresponds to a GTD graph traversal at certain time using specific modes of transport, cf. Figure 5.1.

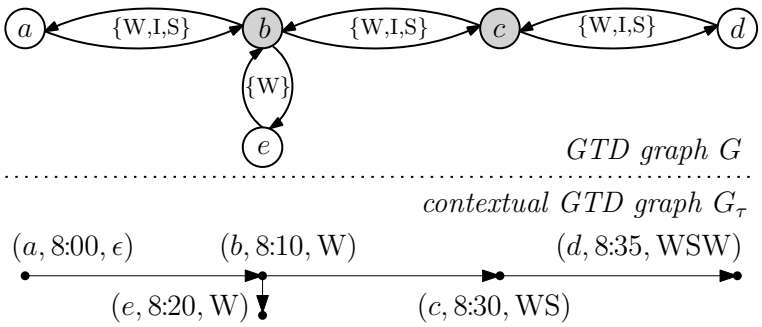


Figure 5.1: An example of a GTD graph and its corresponding contextual GTD graph searched using the walk and shared bike template. Origin is set to a at 8:00; destination is set to d. Grey nodes b and c represent bike sharing stations. The bottom part of the figure shows how the contextual information is represented using the contextual nodes in the contextual GTD graph G_τ.

Let the function $\lambda : M \to \mathbb{R}^+$ returns the travel speed $\lambda(m)$ for a mode of transport $m \in M$. Then the set of contextual nodes V_τ and the set of edges E_τ is constructed using the origin contextual node (o, t, ϵ) and the function $\textsc{out}((v, t_a, m_s), \tau)$ (cf. Algorithm 2) that returns the outgoing edges for a contextual node (v, t_a, m_s) and a template τ. At line 10 of Algorithm 2, it is checked that a mode of transport m is present in the template τ, that a mode change from m_{prev} to m is permitted and that the current mode sequence m'_s matches the journey plan template τ.

Algorithm 2: Outgoing edges of a contextual node.

Input: A contextual node (v, t_a, m_s) and a template τ
Output: A set of outgoing edges from (v, t_a, m_s) given τ

function OUT$((v, t_a, m_s), \tau)$
begin
  $O := \emptyset$
  forall the $(v, w) \in E$ **do**
    forall the $m \in \mu((v, w))$ **do**
      $m'_s := m_s \| m$
      $m_{\text{prev}} := m_n$ where $m_s = m_1 \dots m_n$
      $a := \chi_v(m_{\text{prev}}, m)$
      $b := m'_s$ matches τ
      if $m \in modes(\tau) \wedge a \wedge b$ **then**
        $t' := t_a + \rho_{(v,w)}(t_a, \lambda(m))$
        $O := O \cup ((v, t_a, m_s), (w, t', m'_s))$
      end
    end
  end
  return O
end

The advantage of the contextual GTD graph is that unmodified general shortest path algorithms (e.g., A* or Dijkstra's algorithm) can be used to find journey plans. This is enabled by embedding the domain information (e.g., permitted modes of transport and checking against a journey plan template) in the contextual GTD graph.

From the implementation point of view, the contextual GTD graph can be constructed *on request*. The nodes and edges are created on request only when they are needed during the search process of the respective shortest path algorithm.

5.2.2 Intermodal Planning Algorithm Specification

Now we present how the contextual GTD graph is used to solve the intermodal EAP with templates $J = (G, r, \tau)$. The input of the algorithm is an instance of the problem $J = (G, r, \tau)$ and the output is a journey plan $\pi = (P, \sigma, \phi, \psi)$ that solves the problem $J = (G, r, \tau)$. The algorithm works in two phases:

1. Shortest path algorithm on contextual GTD graph
2. Journey plan derivation

In the first phase, a general shortest path algorithm (e.g., A* or Dijkstra's algorithm) is used to find a path $K = ((x_1, x_2), (x_2, x_3), \dots, (x_k, x_{k+1}))$ of length $|K| = k$

Algorithm 3: Path to journey plan transformation.

Input: A path K in G_τ
Output: A journey plan $\pi = (P, \sigma, \phi, \psi)$

```
1  function DeriveJourneyPlan(K)
2  begin
3      i := 0
4      forall the ((v, t_a, m_s), (v', t'_a, m'_s)) ∈ K do
5          if m_s ≠ m'_s then
6              i := i + 1
7              L_i := ()
8              σ(L_i) := m_j where m'_s = m_1 ... m_j
9          end
10         L_i := L_i ∘ (v, v')
11         φ((v, v')) := t_a
12         ψ((v, v')) := t'_a
13     end
14     P := (L_1, ..., L_i)
15     return (P, σ, φ, ψ)
16 end
```

in the contextual GTD graph $G_\tau = (V_\tau, E_\tau, \rho, \mu, \lambda)$ from the origin contextual node (o, t, ϵ) to the destination contextual node $(d, \cdot, \cdot)$. The edge weight function $\rho_{(v,w)}$ at line 11 of Algorithm 2 returns the duration of traversing an edge $(v, w) \in E$, therefore the journey plan is optimised with respect to its duration (i.e., the earliest arrival problem is solved).

In the second phase, the path K found in the contextual GTD graph G_τ is transformed into a journey plan $\pi = (P, \sigma, \phi, \psi)$. This is done using the DeriveJourney-Plan(K) function, cf. Algorithm 3. The function iterates over edges $((v, t_a, m_s), (v', t'_a, m'_s)) \in K$. Every time the mode sequence is changed, a new journey leg L_i is created and its mode $\sigma(L_i)$ set. The edge $(v, v') \in E$ is then added to the current journey leg L_i using the operator $\circ$ that appends an element to a sequence and the departure $\phi(v, v')$ and arrival $\psi(v, v')$ is set.

It is important to note that if the shortest path algorithm used in the first phase of the algorithm is optimal, then the solution of the intermodal EAP with templates is optimal with respect to journey plan duration and the journey plan template τ.

5.3 Intermodal Planning Algorithm Evaluation

In this section, our proposed approach has been evaluated on real-world PT and road network data for Helsinki. The main purpose of the empirical evaluation was to confirm that the GTD graph representation is flexible enough to allow successfully

Table 5.1: Size of the Helsinki GTD graph.

Graph name	Graph	Nodes	Edges
Time-dependent graph	G^T	50 320	112 127
Network graph	G^N	207 240	585 937
Graph connector	D	-	14 980
GTD graph	G	257 560	713 044

Table 5.2: Journey plan templates used in the evaluation, along with the best performing algorithm for each template.

Template name	Template regexp	Algorithm
Walk only	`^W$`	A*
Bike only	`^I$`	A*
Taxi only	`^X$`	A*
Walk and PT	`^W((B|T|U)W)*$`	Dijkstra's algorithm
Car, walk and PT	`^CW((B|T|U)W)*$`	Dijkstra's algorithm
Walk and shared bike	`^W(SW)?$`	A*
Empty template	N/A	A*

planning intermodal journeys with a variety of mode combinations. We were also interested in measuring how fast the GTD graph can be searched using standard algorithms without any speedup techniques or other optimisation methods.

5.3.1 Experiment Settings

For the evaluation, we use the Helsinki metropolitan area which covers approximately 600 square kilometres. Basic statistics about the size of the Helsinki GTD graph and its components are given in Table 5.1. A fragment of the GTD graph is visualised in Figure 5.2. Note that in Helsinki, there are currently no bike sharing stations. For experimentation purposes, 150 bike sharing stations have therefore been added – the locations of the stations were chosen randomly with the uniform distribution over the nodes V^N of the network graph G^N. In addition, P+R parking places are not properly set in the OSM data. For experiment purposes, 10 P+R parking places were manually inserted into the map at the border of the Helsinki city centre.

We used seven journey plan templates $\tau \in T_7$ for the evaluation, cf. Table 5.2. The templates have been chosen to reflect the typical combinations of modes used in modern multimodal transport systems. To allow a unified description of the results, we treat the intermodal EAP (without templates) as equivalent to the intermodal EAP with templates using the *empty template* permitting any combination of modes.

A* and Dijkstra's algorithms have been used to find a journey plan π given $J = (G, r, \tau)$. A* uses a duration heuristic $h(v)$ calculated as $h(v) = |v, d|/vel_{max}$ where $|v, d|$ is the Euclidean distance between current node v and destination node d, vel_{max} is the speed of the underground set to 120 km/h.

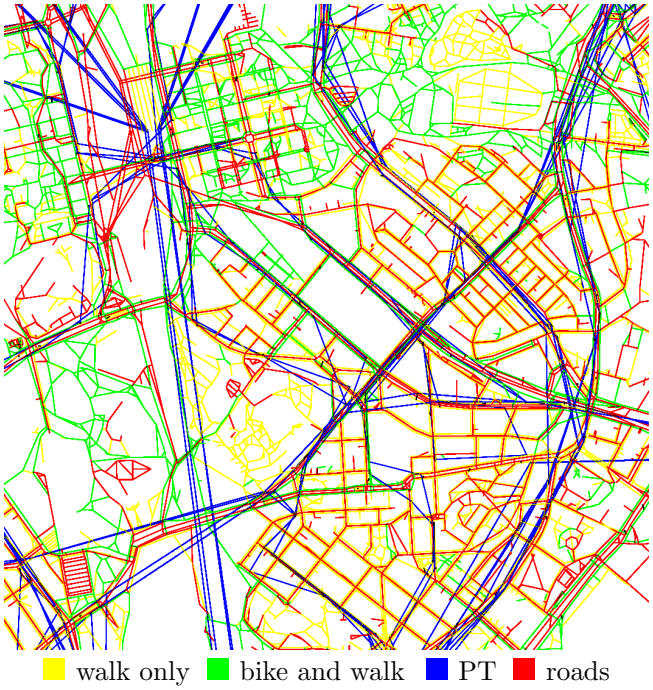


Figure 5.2: Visualisation of a 2.4 km by 2.4 km fragment of the Helsinki GTD graph. Edge colours denote the modes of transport permitted at each edge, cf. legend. All other combinations of modes (e.g., car and taxi, bike only) are marked red. There are approximately 9 500 nodes and 27 700 edges in the visualisation.

Following initial experiments, the better algorithm of the two has been chosen for each template $\tau \in T_7$. The templates $\tau \in T_7$ and their corresponding chosen algorithms are listed in Table 5.2. Consequently, all templates use A* except the walk and PT template and car, walk and PT template where the Euclidean distance heuristic slows-down the A* search process [52] so the Dijkstra's algorithm is used.

The set of instances of the intermodal EAP with templates Q for the experiment were created in the following way. First, $n = 10\,000$ origin-destination-departure triples $Q_t = ((o_1, d_1, t_1), \ldots, (o_n, d_n, t_n))$ were sampled using the uniform distribution over the coordinates of Helsinki area and the uniform distribution over the time interval from 8:00 to 18:00 on 17 Jan 2013. The maximum origin-destination distance was set to 40 km to exclude long trips that are not usual in the urban setting.

Then the origin and destination coordinates were converted to origin and destination nodes from graph G. Let $\delta(c, m)$ be a function that returns the nearest node in the GTD graph G given a coordinate c and a mode of transport m. For example, for the walk mode, the nearest node on a pavement is returned. Then the set of $|Q| = 70\,000$ instances of the intermodal EAP with templates is constructed. Each of the origin-destination-departure triples Q_t is combined with all journey templates

as follows:

$$Q = \{(G, (\delta(o, m_1), \delta(d, m_n), t), \tau) | \\ (o, d, t) \in Q_t \wedge \tau = m_1 \dots m_n \in T_7\}$$

The results presented in the next chapter are based on running the algorithm on one core of a 3.2 GHz Intel Core i7 processor of a Linux desktop computer.

5.3.2 Results

A solution for each problem instance $J \in Q$ has been computed. All instances are divided into three sets based on the distance of their origin and destination location: short (below 10 km), medium (10–20 km), and long (20–40 km). Average runtimes in milliseconds for each journey plan template and origin-destination distance interval are shown in Table 5.3.

Runtimes for all journey plan templates (except templates containing PT) are better than the runtimes for the empty template. This empirically confirms that the journey plan templates constrain the search space of the planner, which results in lower runtimes than when the empty template, which permits any combination of modes, is used. Runtimes for the empty template are better than the runtimes for templates containing PT because the heuristic of the A* algorithm leads the planner well into the destination using the taxi mode (for the majority of requests, taxi is the fastest mode of transport with the lowest journey plan duration).

In general, the templates containing more than one mode of transport are more difficult for the planner (higher branching factor and a larger contextual GTD graph) resulting in higher runtimes than the single-mode templates. Template with the lowest runtimes is the bike only template where the average runtimes range from 15 ms for the short requests up to 178 ms for the long requests. Template with the highest average runtimes is the walk and PT template where the average runtimes ranges from 488 ms for the short requests up to 939 ms for the long requests. The runtimes of car, walk and PT template are lower than the runtimes of the walk and PT template because a significant part of the journey is covered by car and only the last part from the P+R parking place to the destination by walk and PT modes.

Table 5.3: Average runtimes in milliseconds.

Template name	**Short**	**Medium**	**Long**
Walk only	24	135	417
Bike only	15	60	178
Taxi only	31	103	239
Walk and PT	488	817	939
Car, walk and PT	384	477	504
Walk and shared bike	87	223	440
Empty template	376	758	891

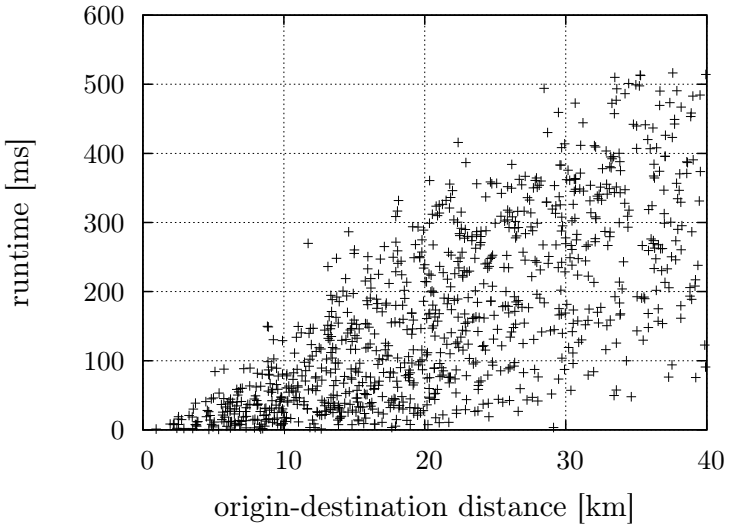


Figure 5.3: Runtime against origin-destination distance (taxi only template, 1 000 randomly selected requests).

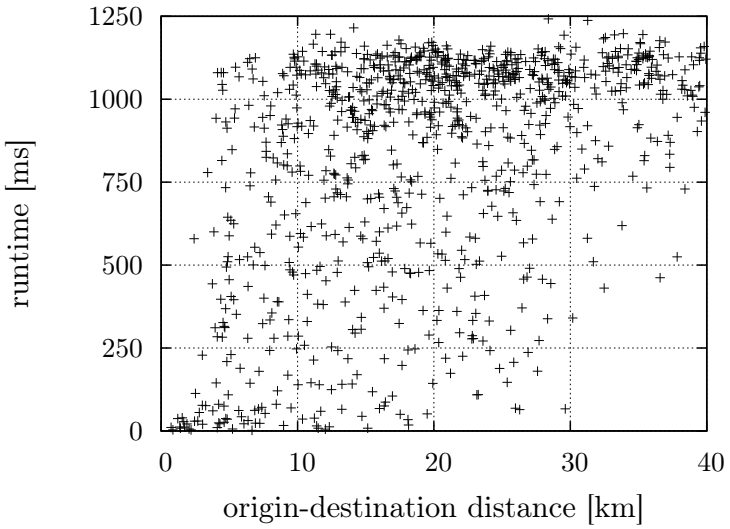


Figure 5.4: Runtime against origin-destination distance (walk and PT template, 1000 randomly selected requests).

Figures 5.3 and 5.4 show scatter plots of the search runtime versus the origin-destination distance for 1 000 randomly selected requests. It can be observed that runtimes for the taxi only template in Figure 5.3 are more strongly correlated on the origin-destination distance than the runtimes of the walk and PT template in Figure 5.4.

5.3.3 Discussion

Compared to the algorithms employing state-of-the-art speedup techniques specifically designed for road network and public transport network variant of EAP, the search times of our method are high. There are several reasons for such a behaviour. First and most importantly, the GTD graph representation is significantly more expressive and flexible, enabling searching for plans from a much richer family of journey plans, which necessarily increases the method's computational cost. Second, no speedup techniques have yet been applied to accelerate the search of the contextual GTD graph. That said, even without the use of speedup techniques and other optimisations, our method achieves practically usable runtimes.

So far, seven journey plan templates have been used in the evaluation. In the future, we plan to add the following useful plan templates:

- Taxi and PT: `^X?W((B|T|U)W)*X?$`
  A taxi can be used for covering the first, the last, or both first and last journey legs.
- Bike and PT: `^IW(UW(IW)?)*I?$`
  A traveller uses his or her own bike to get from an origin to a destination. Where possible and beneficial, PT mode of transport that permits taking bike along is used (in this example only the underground permits it).

5.4 Validation in Real-World Deployments

To fulfil Research objective 3, we have performed validation using real-world deployments. Deploying our approach in the operational environment (TRL 7) is the way to find out the issues that needs to be solved to use the approach in practise. These important issues cannot be found at the stage of algorithm design/evaluation. The validation of the intermodal journey planner in the real-world environment has been done in the scope of the SUPERHUB project[4] that I coinvestigated and where I developed the intermodal planning core. Throughout the deployment, we have acquired useful experience which is common with other problems solved in this thesis so we have concentrated the lessons learned in Chapter 7. In the remaining part of this section, we describe details that are specific for the deployment of the intermodal journey planning system.

The architecture of the intermodal journey planning system is organised in two main parts. The first part is the *intermodal journey planning system backend* deployed on a server. The backend part consists of three layers: (1) the *data layer* responsible for importing, processing and storing relevant data describing the transport network,

[4] https://ec.europa.eu/digital-single-market/en/content/superhub-tailor-made-mobility: The SUPERHUB project (SUstainable and PERsuasive Human Users moBility in future cities; co-financed by the European Commission; grant agreement no.: 289067; 2011–2014) aimed at developing a new services mobility framework supporting an integrated and eco-efficient use of multimodal mobility systems in an urban setting.

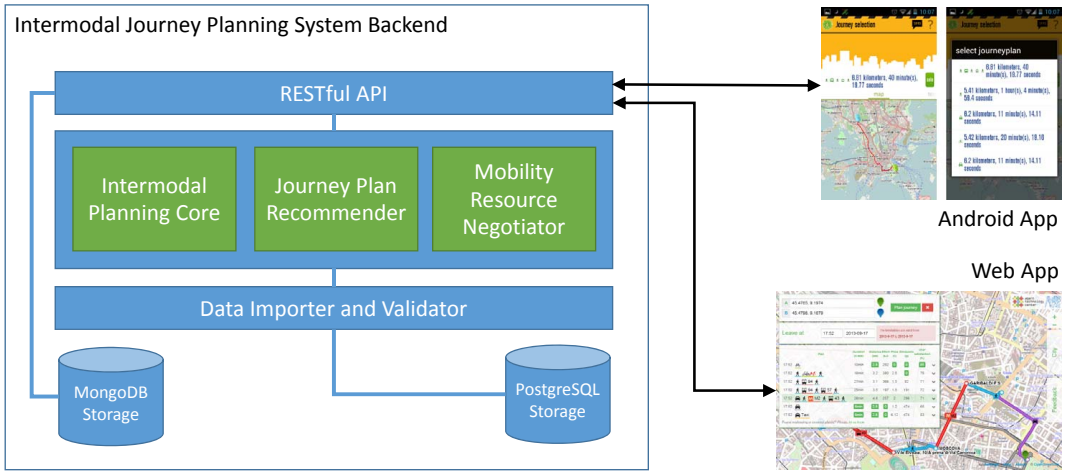


Figure 5.5: Architecture of the intermodal journey planning system.

(2) the *business logic layer* implementing the journey planning functionality, and (3) the *API layer* providing the backend functionality to the frontends. The second part of the system is composed from the *Android App* and *Web App* frontends. See Figure 5.5 for the overview of the architecture. Individual backend components are described below in more details.

Data Importer and Validator: This component was created in collaboration with Zdeněk Moler, Radek Holý, and Daniel Stahr and is responsible for creating the GTD graph (defined in Section 3.2.2) that is then used by the *Intermodal Planning Core* component. The network graph (roads, cycleways, and pavements) is loaded from the XML OSM file. The time-dependent graph for the PT network is loaded from a PostgreSQL database with a GTFS schema. The planner uses a range of additional data, including the information about vehicle and bicycle sharing stations and parking facilities. Upon import, information from different data sources is translated, cross-referenced and integrated into the GTD graph. Resulting GTD graph for Milan is shown in Figure 5.6. This component is also responsible for data validation, e.g., to check whether the timetables do not contain connections with zero runtimes.

Intermodal Planning Core: This component uses intermodal planning algorithm described in Section 5.2 and is responsible for planning multi-leg journeys utilizing the full range of transport modes. To promote modularity and extensibility, the planner employs a novel multi-critics architecture that relies on the notion of *critics*, specialised modules for evaluating candidate journey plans from a certain perspective, e.g., price, emissions or user convenience. The multi-critics architecture allows introducing new concerns to journey planning without modifying the core planning logic. At the moment, the planner evaluates duration, distance, price, CO_2 emissions, physical effort, and user satisfaction of each journey plan.

Journey Plan Recommender: This component developed by Codina et al. [17] is responsible for making personalised trip recommendations best reflecting the user's

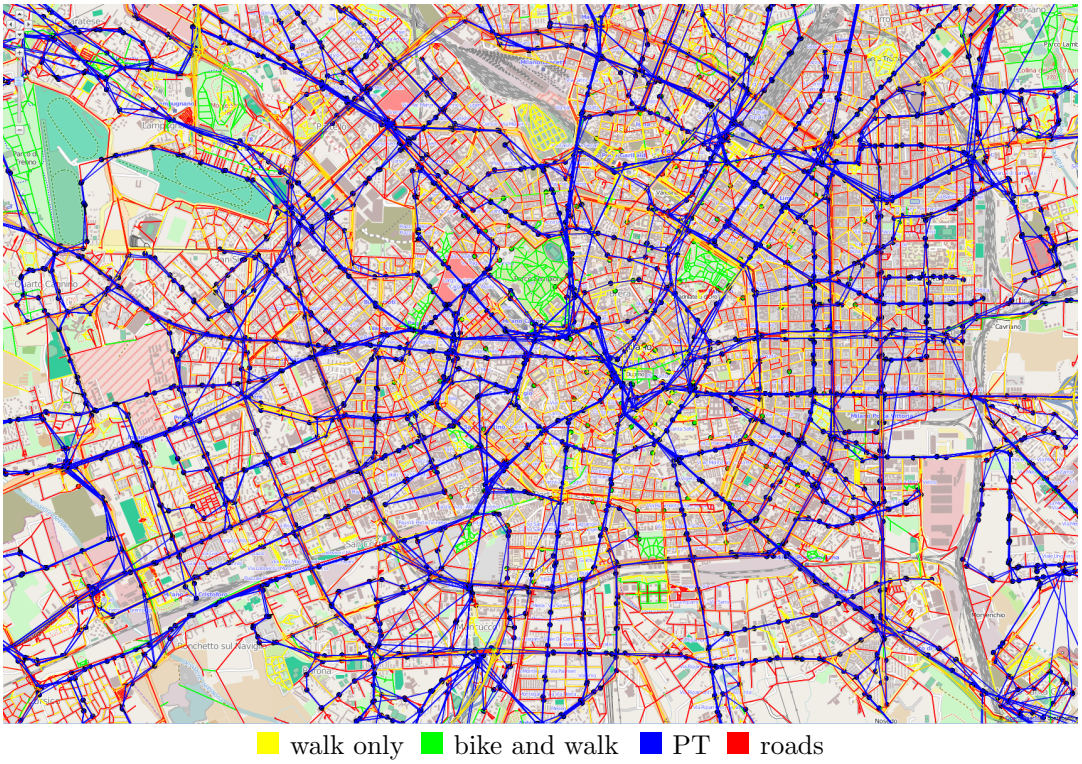


Figure 5.6: GTD graph of Milan (221 707 nodes and 601 695 edges).

situation, needs and preferences. Technically, the recommender uses the latest developments in hybrid recommendation, combining content-based approaches and collaborative approaches. It forecasts user satisfaction with respect to given journey plans and returns it to the intermodal planner core through the critic interface. In particular, user satisfaction with respect to journey plans and journey legs is expressed by an Integer value ranging from 0 (definitely dislike) to 100 (definitely like).

Mobility Resource Negotiator: This component developed by Finnegan et al. [44] interacts with the transport service providers on behalf of the user and allows price calculation together with selecting and reserving services that best match the requirements of the journey plan under consideration. Mobility resources required for carrying out recommended journeys are allocated through a (semi-)automated negotiation mechanisms using an open tendering process [108]. The component is integrated with the intermodal planner core through the critic interface which provides the planner with the feedback on the expected availability and price of requested mobility resources (e.g., Helsinki Region Transport SMS public transport ticket and taxi booking in Milan).

The backend part of the system is used via a RESTful API which allows additional, third party applications and services to be developed on top of the planner. Currently, there are three frontends using the API. To begin with, a Web App and Android App

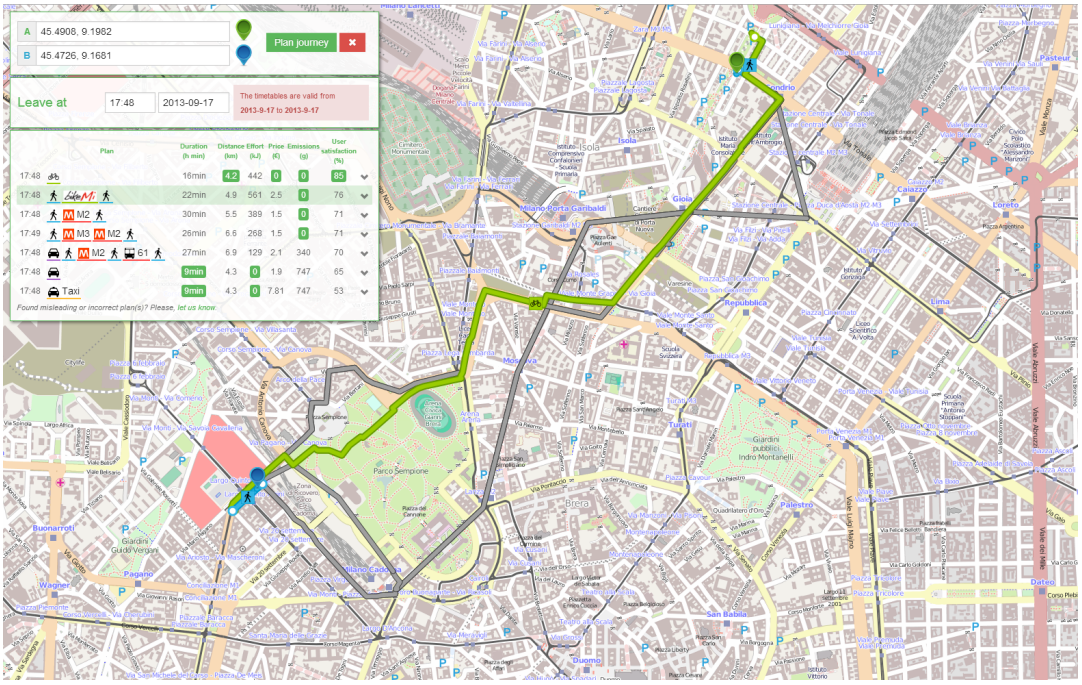


Figure 5.7: Simplified Web App frontend for the planner. A set of intermodal journey plans returned by the planner for the city of Milan is shown together with the values of six supported journey criteria for each plan (developed by Pavol Žilecký).

developed by the SUPERHUB project is available as part of the full SUPERHUB software platform [16]. Second, a simplified Web App frontend which allows users to submit journey planning requests and interactively explore recommended journey plans has been developed by Pavol Žilecký. The Web App is depicted in Figure 5.7 and also available online[5].

We deployed the presented planner, along with additional components, as part of the SUPERHUB project's field trials in three large European cities (Barcelona, Helsinki, and Milan). Several hundred trial participants used the planner for a period of six weeks in autumn 2014. The feedback was largely positive – the users commended the quality of plans and the responsiveness of the system, which was able to return plans within one second in most circumstances.

5.4.1 Journey Planning Quality

Journey planning in a city represented by a graph with several hundred thousands of nodes and edges, cf. Table 5.1, is a very complex task where it is impossible for a human to find an optimal solution manually. This means that there is no other possibility to check the quality of journey planning than to compare journey plans with a different planner. To automate the comparison process, it is needed that the

[5] http://transport.felk.cvut.cz/journeyplanner/

Table 5.4: Results of intermodal planner comparison for individual modes (cities are abbreviated to B = Barcelona, H = Helsinki, and M = Milan).

	Duration difference [%]			**Distance difference [%]**			**Intermodal planner dominated plans [%]**		
Template	B	H	M	B	H	M	B	H	M
Walk only	-13.1	-5.9	-3.2	-8.1	-2.3	-0.8	82.0	91.9	87.0
Bike only	N/A	4.4	N/A	N/A	-10.5	N/A	N/A	39.1	N/A
Car only	-20.0	-17.3	-35.4	-14.1	-3.9	-9.1	85.9	86.8	99.0

second planner has an open API that can be queried automatically. As the second planner, we chose Google Directions API[6] that uses state-of-the-art journey planning technology (also used in Google Maps) and provides an open API.

We compared the quality of the *intermodal planner* (that uses intermodal planning algorithm defined in Section 5.2) for the following four journey plan templates that are also supported by Google Directions API: "Walk only", "Bike only", "Car only", and "Walk and PT". For each city, we created 100 random origin-destination-departure triples that were used for the comparison. The results of the comparison for individual templates "Walk only", "Bike only", and "Car only" are shown in Table 5.4. The results of the comparison for "Walk and PT" template are shown in Table 5.5. In both tables holds that when the duration/distance/number of transfers difference in % is smaller than zero, it means that intermodal planner have found journey plans with lower duration/distance/number of transfers (the difference is calculated over the 100 random queries).

In this paragraph, we discuss the results of individual journey plan templates shown in Table 5.4. Intermodal planner dominated plans column provides a percentage of plans that have strictly better duration than plans found by Google Directions API. It is important to say that the differences is partially caused by a fact that intermodal planner uses OSM maps whereas Google Directions API uses their own maps. Regarding walk, the intermodal planner is slightly better in all three cities (we use the same 5 km/h walking speed as Google does). Bike journey planning was tested only in Helsinki since Google Directions API did not support it in Barcelona and Milan (denoted by N/A in the table). Intermodal planner provides bike journey plans with similar duration and slightly lower distance than Google Directions API. With the car, intermodal planner finds plans with slightly shorter distance than Google Directions API. However, intermodal planner is underestimating the duration of car journey plans.

In the following paragraphs, we discuss the results of "Walk and PT" template shown in Table 5.5. Both planners return multiple "Walk and PT" plans, we than compare the ones that have the minimum number of transfers.

Regarding plan duration, the intermodal planner finds slightly shorter plans in Barcelona and Helsinki. In Milan, the duration difference is much larger (-29.1%).

[6] https://developers.google.com/maps/documentation/directions/

Table 5.5: Results of intermodal planner comparison for "Walk and PT" template.

	Barcelona	**Helsinki**	**Milan**
Duration difference [%]	-7.1	-2.4	-29.1
Number of transfers difference [%]	-15.4	20.0	75.0
Walk distance difference [%]	32.3	24.1	-55.4
Intermodal planner dominated plans [%]	70.7	37.9	44.2
Google planner dominated plans [%]	22.0	43.2	18.6
Non-dominated plans [%]	7.3	18.9	37.2

The difference is caused by different GTFS data used by the two planners. Intermodal planner is missing Trenord trains[7] whereas Google planner is missing ATM operator[8] that have quite a large transport network (i.e., Google then finds walk journey plans instead of using, e.g., underground from the ATM transport network).

Regarding the number of transfers, the intermodal planner is better in Barcelona but provides plans with more transfers in Helsinki. In Milan, the high difference (75%) is caused by the missing ATM data in Google Directions API; the Google planner uses often walk instead of scheduled PT services resulting in significantly lower number of transfers.

Regarding walk distance, the intermodal planner is much better in Milan due to the data issue mentioned above. In Barcelona and Helsinki, intermodal planner offers plans with more walking than Google. This can be easily improved by tuning the maximum walking distance parameter.

Finally, we discuss the quality of plans when two criteria – duration and number of transfers – are taken into account. The intermodal planner dominated plans means a percentage of plans that have strictly better duration and better or the same number of transfers than plans found by Google Directions API. By this overall comparison, we can observe that intermodal planner is better in Barcelona and Milan. In Helsinki, both planners return plans with similar quality.

To summarize, despite having a broader set of features, the quality of journey plans returned by the intermodal planner compares very well (in terms of journey duration and number of transfers) with state-of-the-art commercial journey planners that only support a limited number of transport modes and mode combinations.

5.5 Contributions and Summary

In this chapter, we have presented an advanced intermodal journey planner designed to help travellers to take the full advantage of the increasingly richer and consequently more complex offer of mobility services available in modern cities. Our main contribution with respect to state-of-the-art techniques presented in Section 2.4 is

[7] http://www.trenord.it/en/
[8] http://www.atm.it/en/

a representation-centric approach to solving the intermodal journey planning problem. Thus, instead of providing complex, purpose-specific journey planning algorithms, we have introduced a generalised time-dependent graph that allows representing the intermodal journey planning problem as a standard graph search problem and consequently use general shortest path algorithms to solve it.

Experiments on realistic network data about the Helsinki transport system confirmed the viability of the approach – the planner was able to find a diverse set of journey plans and achieve runtimes which, although noticeably higher compared to algorithms optimised for basic variants of the earliest arrival problem, are generally usable. Finally, the presented planner has been validated in real-world deployments in an intermodal journey planning system with a RESTful API. The system has been deployed along with additional components as part of the SUPERHUB project's field trials in three large European cities (Barcelona, Helsinki, and Milan). In 2014, several hundreds of trial participants were using the planner through Android App and Web App for a period of one month. The feedback was mostly positive – the users commended the quality of plans and the responsiveness of the system, which was able to return plans within one second in most circumstances.

The research presented in this chapter was published in the following articles [63, 71]. The research on this topic has led to related work on PT network analysis [84, 85] and on the integration of multiple journey planners [83].

Chapter 6

Ridesharing on Timetabled Transport Services

Lastly, travelling is an important and frequent activity, yet people willing to travel have to face problems with rising fuel prices, carbon footprint, and traffic jams. One way to tackle these problems is through *ridesharing*, i.e., purposeful and explicit planning to create groups of people that travel together in a single vehicle for parts of the journey. Participants in such schemes can benefit from ridesharing in several ways: sharing parts of a journey may reduce cost (e.g., through group tickets), carbon footprint (e.g., when sharing a private car), and travellers can enjoy the company of others on a long journey. In more advanced scenarios one could even imagine this being combined with working together or holding meetings while travelling.

Following the motivation above, we finally explore the field of multiagent single-criteria multimodal routing, i.e., *multimodal ridesharing*. The potential of improving travel sharing technology has great application value due to its ability to reduce the environmental impact of travelling while providing benefits to travellers at the same time. In general, ridesharing is a widely studied problem – existing work, however, focuses exclusively on ridesharing using vehicles that can move freely on a road transport network. This overlooks the potential for innovative future transport schemes that might exploit ridesharing using *timetabled public transport*. Here, customised group discount schemes could be devised to balance the load across different times of the day, or to make more efficient use of the capacity of public modes of transport. Also, joint travel can be used to increase the comfort and safety of individuals, e.g., for female travellers using night buses, or groups of schoolchildren. To the best of our knowledge, no existing work seems to attempt to compute joint travel plans based on public transport timetables and geographical stop locations, let alone in a way that takes into account the *strategic* nature of the problem, which comes about through the different (and potentially conflicting) preferences of individual travellers.

From the point of view of (multiagent) planning [24], i.e., the problem of synthesising sequences of actions to reach a certain goal – in this case, arrival at a destination from a given point of departure – for several travellers in parallel, ridesharing on

timetabled services presents itself as a very complex application scenario: To begin with, even if one restricted oneself to centralised planning, the domain is huge – public transport data for the UK alone currently involves 240 590 timetable connections for trains and coaches (even excluding local city buses), which would have to be translated to a quarter of a million planning actions, at least in a naive formalisation of the domain. This is the case even if we assume a *non-strategic* setting, where individuals' preferences are not taken into account, and we are simply looking for a set of itinerary that gets everybody to their destination, without any regard for how costly this might be for the individual, or how the joint plan might favour some agents while putting others at a disadvantage. Moreover, considering a *strategic* setting where we are looking for a plan for multiple self-interested agents that are willing to cooperate only if it is beneficial for them is known to be exponentially harder than planning for each agent individually [12]. Yet any automated service that proposes joint journeys would have to guarantee such strategic properties in order to be acceptable for human users.

We proposed a solution employing strategic multiagent planning that guarantees that for any shared journey plan found, each individual is better off taking the shared ride rather than travelling alone, thus providing a clear incentive to participate in it. The core of our algorithm is based on a domain-independent *best-response planning* [72] approach which is a planner that can solve strategic multiagent planning problems of the scale required, and whose properties and assumptions combine particularly well with the ridesharing problem in hand.

6.1 Timetabled Transport Ridesharing Problem

Informally, the problem we are trying to solve is the following: Assume a (potentially very large) set of agents who represent individual travellers, with their individual trips specified in terms of origin and target location. Assume also that the agents want to optimise the individual utility accrued from a trip, and this utility may depend on the travel cost and number of people travelling along each leg of the journey (generally, we will assume that group travel has a positive effect on utility, as we want to study the impact of this very aspect on travel behaviour). Based on this information, we are looking for an algorithm that can identify appropriate groups of travellers who could share parts of their journeys using the full timetabling information of public transport systems, and determine a precise joint travel plan for each group. Also, we want to be sure that if we propose a plan to a group, none of the individual agents will have an incentive to improve on the proposed solution by deviating from it, i.e., we only want to suggest rideshares from which *all* travellers involved will benefit.

This section provides the formalisation of the *timetabled transport ridesharing problem*, which is then used by the ridesharing planning algorithm described in the next section. This formalisation builds on a representation of timetabled transport services captured at two different levels of granularity, which we call the *relaxed* and *full* transport services domain. From a planning perspective, problem formulation

builds on the definition of a multiagent planning problem, which is essentially the combination of several individual planning problems involving an initial and goal state, as well as a set of actions that can be performed by the agent, i.e., the public transport services it can use.

We employ the multiagent paradigm because we want to account for every individual traveller's preferences, and to satisfy certain game-theoretic properties for solutions we calculate (i.e., joint travel plans, parts of which are shared among more than one agent). To our knowledge, such modelling of strategic interaction situations cannot be done without an agent-based model.

6.1.1 Timetabled Transport Services Representation

Since the full travel planning domain with a full granularity of timetabled connections is too large for any current state-of-the-art planner to deal with, we distinguish the *full transport services domain* from what we call the *relaxed transport services domain*, which we will use to come up with an initial plan before mapping it to the full timetable information in our algorithm below. Roughly speaking, the relaxed domain contains information about all travel connections in the transport network with their respective shortest travel times, and ignores any concrete service timetables and information about which passengers are using which services, which are only included in the full domain (the relaxed domain also ignores direct connections among locations with intermediate stops, for reasons that will be explained below). Since only trips that are possible in the relaxed domain are possible in the full domain, this gives us a sound relaxation of the problem we can work with. This relaxation is of course incomplete in the general case, as many trips that are possible in theory cannot be performed in practice due to timetabling constraints, both regarding transport services and participating travellers' requirements.

The *relaxed domain* is a single-agent planning domain represented as a weighted directed graph $T = (V, E, w)$ where the set of nodes V represents the stops and the set of edges E represents the connections provided by a service. The graph must be directed because there exist stops that can only be used in one direction. There is an edge $e = (A, B) \in E$ from stop A to B in this graph if there is at least one connection from A to B in the timetable. The weight $w(e)$ of this edge is given by the weight function $w : E \to \mathbb{R}_0^+$ which returns the minimal time needed for travelling from A to B. A plan $P_i = \langle A_1 \to A_2, A_2 \to A_3, \ldots, A_{k-1} \to A_k \rangle$ found in the relaxed domain for the agent i is a sequence of $k-1$ connections to travel from its origin A_1 to its destination A_k.

A small example of the relaxed domain is shown in Figure 6.1. An example plan for an agent travelling from C to F is $P_1 = \langle C \to D, D \to E, E \to F \rangle$. To give an idea of the difference between the relaxed domain and the full timetable in terms of domain complexity, there are 497 connections in the relaxed domain for trains and coaches in the Yorkshire area compared to 10 295 timetable connections.

Direct trains that do not stop at every stop are filtered out from the relaxed domain for the following reason: Assume that in Figure 6.1, there is only one agent

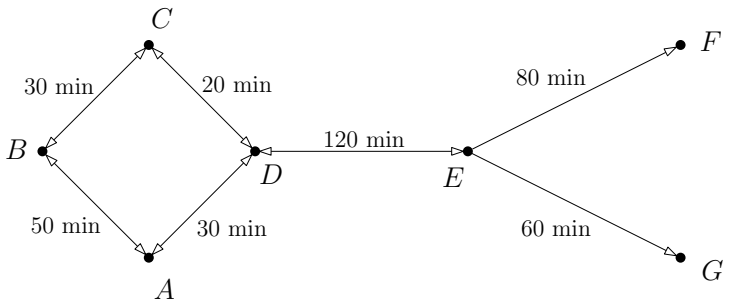


Figure 6.1: An example of the relaxed domain showing basic connection times (e.g., it takes 50 minutes to travel from A to B).

travelling from C to F and that its plan in the relaxed domain is to use a direct train from C to F. In this case, it is only possible to match its plan to direct train connections from C to F, and not to trains that stop at C, D, E, and F. Therefore, the agent's plan cannot be matched against all possible trains between C and F which is problematic especially in the case where the majority of trains stop at every stop and only a few trains are direct. On the other hand, it is possible to match a plan with a train stopping in every stop to a direct train, as explained later in Section 6.2.4.

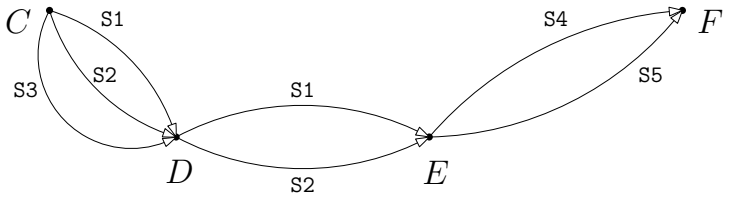


Figure 6.2: An example of the full domain with stops C, D, E and F for the merged plan of two single-agent plans $P = \{C \xrightarrow{\{1\}} D \xrightarrow{\{1,2\}} E \xrightarrow{\{1\}} F\}$.

Assume a set $N = \{1, \ldots, n\}$ of agents in the full domain, where each agent i has plan P_i from the relaxed domain. Then the *full domain* is a multiagent planning domain constructed using a *merged plan* P of single-agent plans $P_1, \ldots, P_n$ defined by formula

$$P = \bigcup_{i=1}^{n} P_i = (V', E', l')$$

where we interpret $\bigcup$ as the union of graphs that would result from interpreting each plan as a set of edges connecting stops. More specifically, given a set of single-agent plans, the plan merging operator $\bigcup$ computes its result in three steps: First, it transforms every single-agent plan P_i to a directed graph $T_i = (V_i, E_i)$ where the nodes V_i are the stops from the single-agent plan P_i and the edges E_i represent the atomic travel actions of P_i (for instance, a plan $P_1 = \langle C \to D, D \to E, E \to F \rangle$ is transformed to a directed graph $T_1 = \{C \to D \to E \to F\}$). Second, the merging operator performs a graph union operation $\bigcup_{i=1}^{n} T_i = (V', E', l')$ over the directed

graphs and sets $V' = \bigcup_{i=1}^{n} V_i$, $E' = \bigcup_{i=1}^{n} E_i$, and labels every edge $e = (A, B) \in E'$ with the numbers of agents that are using the edge by a labelling function $l' : V' \times V' \to 2^N$. As an example, following Figure 6.1, the merged plan of plans of agent 1 travelling from C to F and sharing a journey from D to E with agent 2 would be computed as

$$\langle C \to D, D \to E, E \to F \rangle \cup \langle D \to E \rangle = \{C \xrightarrow{\{1\}} D \xrightarrow{\{1,2\}} E \xrightarrow{\{1\}} F\}$$

With this, the *full domain* is represented as a labelled directed multigraph $T' = (V', E_t, l, l')$ where the set of nodes V' represents the stops that are present in the merged plan P of plans from the relaxed domain. A set of edges E_t represents the journey services from the timetable. The labelling function $l : E_t \to \langle s, t_A, \tau \rangle$ returns a triple of a unique service name s, a departure time t_A from stop A, and a duration τ of the service journey between stops A and B for each edge $e = (A, B) \in E_t$. The labelling function $l' : V' \times V' \to 2^N$ labels every edge $e \in E'$ with the number of agents using it.

A joint plan π with a timetable is a sequence $\pi = \langle a^1 \dots a^k \rangle$ of joint actions. Each joint action a^j from π represents a subset $N_j \subseteq N$ of agents travelling together using a specific service s_j.

In the example of the full domain in Figure 6.2, the agents can travel using some subset of five different services `S1` to `S5`. The full domain example is based on the group of agent 1 (travelling from C to F) and agent 2 (travelling from D to E) where initial single-agent plans have been found in the relaxed domain shown in Figure 6.1. In order to travel from C to D using service `S1`, an agent must be present at stop C before the departure of service `S1` to D.

6.1.2 Multiagent Planning Problem

To model the ridesharing problem, we use a multiagent planning formalism which is based on MA-STRIPS [12] and coalition-planning games [13]. States are represented by sets of ground fluents, actions are tuples $a = \langle pre(a), \mathit{eff}(a) \rangle$. These fluents are logical propositions describing aspects of the current state that may change over time, e.g., $at(1, l_1)$ to express that agent 1 is at location l_1. After the execution of action a, positive fluents p from $\mathit{eff}(a)$ are added to the state and negative fluents $\neg p$ are deleted from the state. For example, an action $travel(A, X, Y)$, when applied to the case of $A = 1$ travelling from $X = l_1$ to $Y = l_2$ would make $at(1, l_1)$ false and $at(1, l_2)$ true. Each agent has individual goals and actions with associated costs. There is no extra reward for achieving the goal, the total utility received by an agent is simply the inverse of the cost incurred by the plan executed to achieve the goal. In the ridesharing domain, the agents are the travellers, located in their origin locations in the initial state, and attempting to achieve goal states where they are at their destination locations. Agents specify the initial state and the goal state but their journey plans are computed for them centrally.

More formally, following the notation of [72], a *multiagent planning problem* is a tuple

$$\Pi = \langle N, F, I, \{G_i\}_{i=1}^n, \{A_i\}_{i=1}^n, \Psi, \{c_i\}_{i=1}^n \rangle$$

where

- $N = \{1, \ldots, n\}$ is the set of agents,
- F is the set of fluents,
- $I \subseteq F$ is the initial state,
- $G_i \subseteq F$ is agent i's goal,
- A_i is agent i's action set,
- $\Psi : A \to \{0, 1\}$ is an admissibility function,
- $c_i : \times_{i=1}^n A_i \to \mathbb{R}$ is the cost function of agent i.

$A = A_1 \times \ldots \times A_n$ is the joint action set assuming a concurrent, synchronous execution model, and $G = \wedge_i G_i$ is the conjunction of all agents' individual goals. The assumption of synchronous action among agents here is an important simplification to make the problem more tractable. We will see below how it is possible to determine specific synchronisation points for jointly travelling agents when mapping the problem to the full timetabling information. A multiagent planning problem typically imposes concurrency constraints regarding actions that cannot or have to be performed concurrently by different agents to succeed which the authors of [72] encode using an admissibility function Ψ, with $\Psi(a) = 1$ if the joint action a is executable, and $\Psi(a) = 0$ otherwise.

A *plan* $\pi = \langle a^1, \ldots, a^k \rangle$ is a sequence of joint actions $a^j \in A$ such that a^1 is applicable in the initial state I (i.e., $pre(a^1) \subseteq I$), and a^j is applicable following the application of $a^1, \ldots, a^{j-1}$. We say that π *solves* the multiagent planning problem Π if the goal state G is satisfied following the application of all actions in π in sequence. The cost of a plan π to agent i is given by $C_i(\pi) = \sum_{j=1}^k c_i(a^j)$. Each agent's contribution to a plan π is denoted by π_i (a sequence of $a_i \in A_i$).

6.1.3 Timetabled Transport Ridesharing Problem Definition

The real-world ridesharing domain used in this work is based on the large and complex public transport network in the UK. An agent representing a passenger is able to use different modes of transport during its journey: walking, trains, and coaches. The aim of each agent is to get from its starting location to its final destination at the lowest possible cost. The cost of an agent's journey can be based on the weighted sum of several criteria such as journey duration, ticket price, mode of transport, and number of agents travelling together.

For the purposes of this work, we will make the assumption that sharing a part of a journey with other agents is cheaper than travelling alone. While this may not

currently hold in many public transport systems, defining hypothetical cost functions that reflect this would help assess the potential benefit of introducing such pricing schemes. This means that our cost functions reflect *synergies* occurring from the joint use of a resource, and this can be easily accommodated within the framework of best-response planning, where these positive effects on cost are simply treated as "negative contention", i.e., the cost to each agent when sharing a resource simply decreases instead of increasing. Note that this does not imply that every time an agent decreases her local cost this will benefit everybody else. For example, agent A might abandon the plan to share with B in order to reduce her overall cost, and join agent C instead, thus increasing B's cost, who will now travel alone. Thereupon B will try to improve on this result (and so on), the important property of BRP being that this process is guaranteed to terminate, and will result in a joint plan in which no individual agent can improve further on. Also, it is worth pointing out that a joint plan will not necessarily be globally optimal – its quality will depend on the initial plan computed before the best-response process.

The ridesharing problem is then, for a given travel demand expressed as a set of origin-destination pairs, one for each agent, finding groups of agents and corresponding shared journey plans. We define the ridesharing problem more formally by presenting definitions for problem instances and our formal solution concept: A *timetabled transport ridesharing problem* is a triple $P = \langle T, T', G \rangle$, where

- $T = (V, E, w)$ is the *relaxed domain* containing a set V of *public transport stops*,
- $T' = (V', E_t, l, l')$ is the *full domain* over the subset $V' \subseteq V$ of public transport stops, and
- $G = \{(o_1, d_1), \ldots, (o_c, d_c)\}$ is a *set of agent trips* (an agent's goal is to travel from an origin to a destination), where each agent's trip $g \in G$ is represented by a tuple $g = (o, d)$ denoting the agent's origin $o \in V$ and destination $d \in V$.

A solution to this problem is a *joint plan* $\pi = \langle a^1, \ldots, a^k \rangle$ specifying fully the shared journeys of agents in terms of connections from the timetable and fulfilling all agent trips $g \in G$. From the many joint plans possible, we are looking for such a joint plan that correspond to a Nash equilibrium, i.e., where no agent/traveller can unilaterally improve its individual journey cost.

6.2 Ridesharing Planning Algorithm

Once we have formalised the problem, we can proceed to a detailed description of the ridesharing planning algorithm. The algorithm takes as an input the timetabled transport ridesharing problem $P = \langle T, T', G \rangle$, a maximum travel group size $n_{\max}$, and a maximum bearing difference $\Delta\varphi$. A bearing $\varphi(t)$ for a trip $t = (o, d)$ is defined as an angle in degrees, measured in the clockwise direction, between the north reference ray and the origin-destination ray. Bearing of a trip is used to identify trips with a similar direction as these are more suitable for ridesharing than trips with opposite

bearing. The output of the algorithm is a joint plan $\pi = \langle a^1, \ldots, a^k \rangle$ that fulfils all agent trips $g \in G$.

The main problem when planning for an identified group of agents with a centralised multiagent planner is the exponential blowup in the action space which is caused by using concurrent, independent actions [72]. Using a naive PDDL translation has proven that a direct application of a centralised multiagent planner to this problem does not scale well. As mentioned above, we tackle the complexity of the domain by breaking the planning process down into different phases that avoid dealing with the full fine-grained timetable data from the outset. The overall algorithm, which is shown in Figure 4, is designed to work in four phases, which we will now describe in detail.

6.2.1 The Trip Grouping Phase

The algorithm starts with the *trip grouping phase* where the trips $G = \{(o_1, d_1), \ldots, (o_c, d_c)\}$ are grouped into groups of at most $n_{\max}$ agents. Groups are created incrementally from G, until G becomes empty, in the following way: First, pick a trip $g' \in G$ at random. Then, create a set of candidate trips $G' = \{g \in G | \mathrm{bd}(g, g') \leq \Delta\varphi\}$ that have a similar bearing as g' (function $\mathrm{bd}(g, g')$ calculates the bearing difference between trips g and g'). Next, create a group $G_j \subseteq G'$ by selecting at most $n_{\max}$ trips with minimum spatial difference $\mathrm{sd}(\cdot, g')$ to g'. Here, the spatial difference $\mathrm{sd}(g, g')$ of two trips g and g' is defined as

$$\mathrm{sd}(g, g') = |o, o'| + |d, d'|,$$

where $|o, o'|$ denotes the direct distance between the origins of the two trips, and $|d, d'|$ the direct distance between their destinations. Once a group G_j is created, the trips $g \in G_j$ are deleted from the set of all trips G.

For each group G_j, a joint journey plan π with a timetable is found by applying the next three phases of the algorithm.

6.2.2 The Trip Planning Phase

In the *trip planning phase*, an initial journey is found for each agent i from the set of agents G_j using the relaxed domain $T = (V, E, w)$ where the action set is identical for every agent and contains all transport services available in the transport network. A journey for each agent is calculated independently of other agents in the scenario using a single-agent planner. As a result, each agent is assigned a single-agent plan P_i which will be further optimised in the next phase. This approach makes sense in our domain because the agents do not need each other to achieve their goals and they cannot invalidate each other's plans. A PDDL specification for the relaxed domain is shown in Section 6.3.

Algorithm 4: Four-phase algorithm for finding shared journeys for agents.

Input: Timetabled transport ridesharing problem $P = \langle T, T', G \rangle$
Maximum travel group size $n_{\max}$
Maximum bearing difference $\Delta\varphi$
Output: Joint plan $\pi = \langle a^1, \ldots, a^k \rangle$ that fulfils all agent trips $g \in G$

```
// 1. The trip grouping phase
```
$j := 0$
while $G \neq \emptyset$ **do**
  Pick a trip $g' \in G$ at random
  Create a set of candidate trips $G' = \{g \in G | \mathrm{bd}(g, g') \leq \Delta\varphi\}$
  Create a group $G_j \subseteq G'$ by selecting at most $n_{\max}$ trips
    with minimum spatial difference $\mathrm{sd}(\cdot, g')$ to g'
  Delete trips $t \in G_j$ from G
  $j := j + 1$
end
for each created group $G_j = \{1, \ldots, n\}$ **do** the next three phases

```
// 2. The trip planning phase
```
for $i = 1, \ldots, n$ **do**
  Find an initial journey for agent i using a single-agent planner
end

```
// 3. The best-response phase
```
repeat
  for $i = 1, \ldots, n$ **do**
    Create a simpler best-response planning problem
      from the point of view of agent i
    Minimise the cost of i's plan without changing the plans of others
  end
until *no change in the cost of the joint plan*

```
// 4. The timetabling phase
```
Identify independent groups of agents $I = \{u_1, \ldots, u_m\}$, where $u_i \in 2^N$
for $i = 1, \ldots, m$ **do**
  Find the relevant timetable for group u_i
  Match the joint plan of u_i to timetable using a temporal
    single-agent planner in the full domain with the relevant timetable
end

6.2.3 The Best-response Phase

The *best-response phase* is based on the relaxed domain. Again, the action set is identical for every agent and contains all transport services available in the transport network. The algorithm uses the best-response planning algorithm as described below. It iteratively creates and solves simpler best-response planning problems from the point of view of each individual agent. In the case of the relaxed domain, the best-response planning problem looks almost the same as a problem of finding a single-agent journey. The difference is that, as we have explained in Section 6.1.3, we make the assumption that the cost of travelling is smaller when an agent uses a connection which is used by one or more other agents. A specific cost function used for the evaluation of the algorithm is defined in Section 6.4.1.

Iterations over agents continue until there is no change in the cost of the joint plan between two successive iterations. This means that the joint plan cannot be further improved using the best-response approach. The purpose of this is not only to exploit local, "greedy" optimisations for single agents in an overall schedule of plans. It also ensures that the proposed joint solution is compatible with the incentives of individual agents, i.e., they could do no better on their own by deviating from it. The fact that the first iteration of the best-response optimisation starts from initial plans that agents can perform on their own ensures this (any subsequent plan generated will be cheaper to them). The output of the best-response phase is a merged plan P of the single-agent plans in the relaxed domain (defined in Section 6.1.1) that specifies which connections the agents use for their journeys and which segments of their journeys are shared. The merged plan P will be matched to the timetable in the final phase of the algorithm.

Best-response Planning

The *best-response planning* algorithm proposed in [72] is an algorithm which, given a solution π^k to a multiagent planning problem Π, finds a solution π^{k+1} to a *transformed planning problem* Π_i with minimum cost $C_i(\pi^{k+1})$ for agent i among all possible solutions, while considering all other agents' plans to be fixed:

$$\pi^{k+1} = \arg\min\{C_i(\pi) | \pi \text{ identical to } \pi^k \text{ for all } j \neq i\}$$

The transformed planning problem Π_i is obtained by rewriting the original problem Π so that all other agents' actions are fixed, and agent i can only choose its own actions in such a way that all other agents still can perform their original actions. Since Π_i is a single-agent planning problem, any cost-optimal planner can be used as a best-response planner.

In [72], the authors show how for a class of congestion planning problems, where all fluents are *private*, the transformation they propose allows the algorithm to converge to a Nash equilibrium if agents iteratively perform best-response steps using an optimal planner. This requires that every agent can perform its actions without requiring another agent, and hence can achieve its goal in principle on its own,

and conversely, that no agent can invalidate other agents' plans. Assuming infinite capacity of vehicles (or, more realistically, large enough capacities to accommodate at least the number of agents for whom we are trying to find a plan), the relaxed domain is an instance of a congestion planning problem: following the definition of a congestion planning problem in [72], all actions are private, as every agent can use modes of transport on their own and the other agents' concurrently taken actions only affect action cost. The convergence of the best-response phase derives from the theorem presented in [72] which states that for any congestion planning problem, best-response planning converges to a pure-strategy Nash equilibrium.

The best-response planner works in two phases: In the first phase, an initial plan for each agent is computed (e.g., each agent plans independently or a centralised multiagent planner is used). In the second phase, the planner solves simpler best-response planning problems from the point of view of each individual agent. The goal of the planner in a best-response planning problem is to minimise the cost of an agent's plan without changing the plans of others (though the cost of their plans might change as explained in Section 6.1.3). Consequently, it optimises a plan of each agent with respect to the current joint plan.

This approach has several advantages. It supports full concurrency of actions and the best-response phase avoids the exponential blowup in the action space resulting in much improved scalability. For the class of potential games [79], it guarantees convergence to a Nash equilibrium. On the other hand, it does not guarantee the optimality of a solution, i.e., the quality of the equilibrium in terms of overall efficiency is not guaranteed (it depends on which initial plan the agents start off with). However, experiments have proven that it can be successfully used for improving general multiagent plans [72].

6.2.4 The Timetabling Phase

In the final *timetabling phase*, the optimised shared journeys are matched against timetables using a temporal single-agent planner which assumes the full domain. For this, in a first step, independent groups of agents with respect to journey sharing are identified. An independent group of agents is defined as an edge disjoint subgraph of the merged plan P. This means that actions of independent groups do not affect each other so it is possible to find a timetable for each independent group separately.

Then, for every independent group, *parts* of the group journey are identified. A *part* of the group journey is defined as a maximal continuous segment of the group journey which is performed by the same set of agents. As an example, there is a group of two agents that share a segment of their journeys in Figure 6.3: Agent 1 travels from A to G while agent 2 travels from B to H. Their group journey has five parts, with the shared part (part 3) of their journey occurring between stops C and F.

In order to use both direct and stopping trains when the group journey is matched to the timetable, the *relevant timetable* for a group journey is composed in the following way: for every part of the group journey, return all timetable services in the direction of agents' journeys which connect the stops in that part. An example of

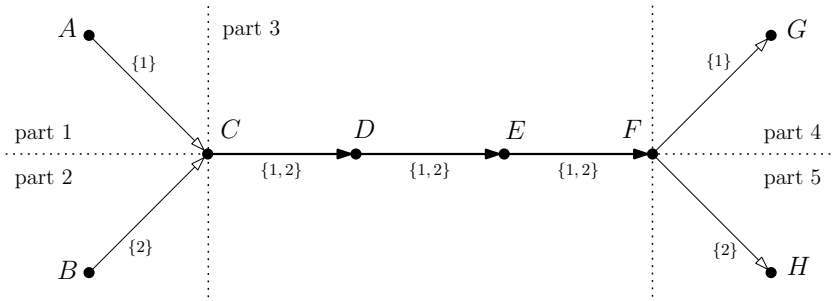


Figure 6.3: Parts of the group journey of two agents.

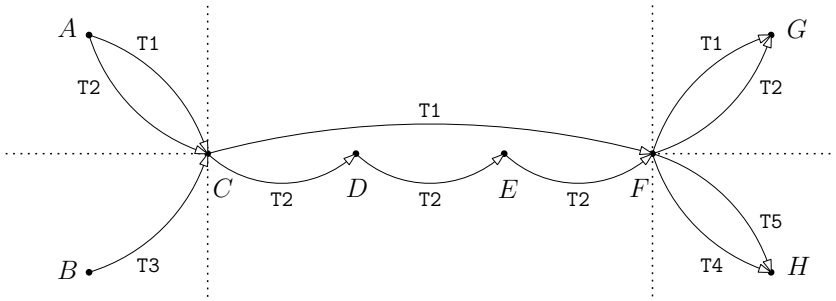


Figure 6.4: The full domain with services from the relevant timetable. There are five different trains `T1` to `T5`, and train `T1` is a direct train.

the relevant timetable for a group of agents from the previous example is shown in Figure 6.4. Now, the agents can travel using the direct train `T1` or using train `T2` with intermediate stops.

The relevant timetable for the group journey is used with the aim to cut down the amount of data that will be given to a temporal single-agent planner. For instance, there are 9881 timetabled connections for trains in the Yorkshire area. For an example journey of 4 agents, there are only 634 services in the relevant timetable which is approximately 6% of the data. As a result, the temporal single-agent planner gets only the necessary amount of data as input, to prevent the time-consuming exploration of irrelevant regions of the state space.

In the timetabling phase, every agent in a group of agents tries to spend the shortest possible time on its journey. When matching the plan to the timetable, the temporal planner tries to minimise the sum of durations of agents' journeys including waiting times between services. A PDDL specification for the full domain is shown in Section 6.3.

Once the timetabling phase finishes, the algorithm adds a joint plan $\pi^j = \langle a^1, \ldots, a^k \rangle$ for the identified group of agents $G_j \in G$ to the final joint plan $\pi = \pi \bigcup \pi^j$. The algorithm then proceeds to the next group $G_j \in G$.

6.3 Ridesharing Planning Algorithm Implementation

This section describes two important aspects of the ridesharing planning algorithm implementation. It deals with the conversion of public transport timetables data to the Planning Domain Definition Language and with the choice of planners for implementing the individual phases of the algorithm.

To be able to use timetables data of public transport services with modern AI planning systems, it has to be converted to the Planning Domain Definition Language (PDDL). We transformed the data in three subsequent stages. First, we transformed the NPTDR and NaPTAN XML data to a spatially-enabled PostgreSQL database. Second, we automatically processed and optimised the data in the database. The data processing by SQL functions in the procedural PL/pgSQL language included the following steps: merging bus bays at bus stations and parts of train stations, introducing walking connections to enable multimodal journeys, and eliminating duplicates from the timetable. Finally, we created a script for generating PDDL specifications based on the data in the database. More details about the data processing and PDDL specifications can be found in [62].

```
(define (domain travelplanner)
    (:requirements :typing :action-costs)
    (:types location)
    (:predicates
        (connection ?origin - location ?destination - location)
        (at ?loc - location)
    )
    (:functions
        (time ?origin - location ?destination - location)
        (total-cost)
    )
    (:action go
        :parameters (?o ?d - location)
        :precondition (and (at ?o) (connection ?o ?d) )
        :effect (and
            (at ?d) (not (at ?o))
            (increase (total-cost) (time ?o ?d)) )
    )
)
```

Figure 6.5: The domain file for the relaxed domain.

In the relaxed domain used in the trip planning and best-response phase, a single agent aims to travel from its origin to its destination. The domain file contains two predicates, two functions and only one action, cf. Figure 6.5. The predicate `connection` is true when there is an edge from `?origin` to `?destination` (there are separate edges for walking, travel by bus or train), the predicate `at` denotes the current location of the agent. The function `time` returns the cost of travelling from the location `?origin` to `?destination`. The action `go` moves the agent from the location `?o` to `?d` and it increases the total cost of the plan which is stored by the `total-cost` function. The problem file then contains the origin and destination of

```
(:durative-action go-agent-1-2_C-D
 :parameters (?s - service)
 :duration (= ?duration (+
     (- (+ (departure C D ?s) (runtime C D ?s)) (agent-time agent1))
     (- (+ (departure C D ?s) (runtime C D ?s)) (agent-time agent2))
 ))
 :condition (and
     (at start (connection C D ?s))
     (at start (at agent1 C))
     (at start (<= (agent-time agent1) (departure C D ?s)))
     (at start (at agent2 C))
     (at start (<= (agent-time agent2) (departure C D ?s)))
 )
 :effect (and
     (at end (at agent1 D))
     (at start (not (at agent1 C)))
     (at end (assign (agent-time agent1)
                     (+ (departure C D ?s) (runtime C D ?s))))
     (at end (at agent2 D))
     (at start (not (at agent2 C)))
     (at end (assign (agent-time agent2)
                     (+ (departure C D ?s) (runtime C D ?s))))
))
```

Figure 6.6: A durative action *go-agent-1-2_C-D* in the domain file for the full domain.

the agent, the list of stops, and the list of connections between the stops and their costs.

In the full domain used in the timetabling phase, multiple agents aim to travel from their origins to their destinations. The full domain is a multiagent planning domain constructed using a merged plan P of single-agent plans $P_1, \ldots, P_n$. Therefore, it contains only the stops that are present in the union of these plans, with the shared parts of the journeys and "who shares which part of the journey" already specified. In the process of finding a plan for the full domain, a joint plan of the group of agents is instantiated with concrete timetabled services.

The domain file contains a list of partially instantiated durative actions for travelling from one stop to another, where origin, destination, and agents using this action are instantiated, and the only free variable is the name of the service the agents are going to use. The function `(agent-time ?a - agent)` is used to store the current time of the agent `?a`. An example of a durative action is shown in Figure 6.6. The durative action *go-agent-1-2_C-D* enables agent 1 and 2 to travel together from stop C to stop D. If the travel from C to D is shared by three agents, the domain file would contain an action *go-agent-1-2-3_C-D*.

Let N be the number of agents travelling together, at_i the current time of agent i, $d_{CD}(s)$ the departure time of service s from the stop C to D and $r_{CD}(s)$ its duration. Then, the duration D_{CD} of the action to travel from the stop C to D is computed as $D_{CD} = \sum_{i=1}^{N} (d_{CD}(s) + r_{CD}(s) - at_i)$. The temporal planner tries to minimise the sum of the durations of agents' journeys. In other words, it tries to find a journey with minimal waiting times between services.

The conditions of the action are the following: there must be a connection by the service s between the stops C, D and the agents must be present at the stop C before the departure of service s. Once the action is executed, the agents are located at stop D and their current time is set to the arrival of the service s at stop D. The problem file contains origins and destinations of the agents and the list of services and their departures and durations.

6.3.1 Planners

All three single-agent planners used for the implementation were taken from recent International Planning Competitions from 2008 and 2011. We use LAMA [93] in the trip planning phase and for each of the individual single-agent best-response iterations in the best-response phase. LAMA is a sequential *satisficing* (as opposed to cost-optimal) planner which searches for any plan that solves a given problem and does not guarantee optimality of the plans computed. LAMA is a propositional planning system based on heuristic state-space search. Its core feature is the usage of landmarks, i.e., propositions that must be true in every solution of a planning problem.

SGPlan_6 [69] and POPF2 [18] are the two temporal satisficing planners used in the timetabling phase. Such temporal planners take the duration of actions into account and try to minimise makespan (i.e., total duration) of a plan but do not guarantee optimality. The two planners use different search strategies and usually produce different results. This allows us to run them in sequence on every problem and to pick the plan with the shortest duration. It is not strictly necessary to run both planners, one could save computation effort by trusting one of them.

In many of the experiments, the SGPlan_6 and POPF2 used in the timetabling phase returned some plans in the first minute but then they continued exploration of the search space without returning any better plan. To account for this, we imposed a time limit for each planner in the temporal planning stage to 2 minutes for a group of up to 4 agents and 4 minutes otherwise.

6.4 Ridesharing Planning Algorithm Evaluation

We have evaluated the proposed ridesharing algorithm on realistic scenarios based on real-world public transport timetables and travel demand data for the Yorkshire area of the United Kingdom. The size of the area was dictated solely by our need to evaluate the algorithm on the whole travel demand (approximately 100 000 train trips per day). The ridesharing planning algorithm itself scales up well up to the area of the whole UK, as was shown in our previous work [64]. However, there the algorithm was evaluated on travel demand that was very sparsely and randomly sampled, not necessarily showing any correlation to actual travel demand profiles.

6.4.1 Experiment Settings

The experiments are situated in the Yorkshire area (East and West Yorkshire, East Riding of Yorkshire, York, and Selby administrative areas) which covers an area of approximately 130 by 70 km, i.e., around 9 100 km^2. According to the UK origin-destination census data from 2001 [86], there are 2 million passenger trips a day in the Yorkshire area. In order to focus on the timetabled public transport trips, the modal split in the UK across different modes of transport in 2001 [42] was used to estimate the number of trips for each mode, cf. Table 6.1.

Table 6.1: Numbers of trips per day in the Yorkshire area [42, 86].

Transport mode	**Modal split**	**100% trips**	**50% trips**	**5% trips**
Trains	5.3%	106 035	53 017	5 302
Coaches	0.3%	6 002	3 001	300
Local buses	6.0%	120 039	60 020	6 002
Passenger cars	88.3%	1 766 576	883 288	88 329
Total	100.0%	1 998 651	999 326	99 933

Since we assume that all agents are travelling on the same day and that all journeys must be completed within 24 hours, in what follows below we consider only public transport timetables data for Tuesdays (this is an arbitrary choice that could be changed without any problem).

The timetable data used in this work contains neither information about ticket prices nor distances between adjacent stops, only durations of journeys from one stop to another. This significantly restricts the design of a cost functions used for the planning problems. Therefore, the cost functions used in this work are based solely on the duration of journeys. The cost $c_{i,n}$ for agent i travelling from A to B in a group of n agents is then defined by equation (6.1):

$$c_{i,n} = \left(\tfrac{1}{n}\, 0.8 + 0.2\right) c_i \tag{6.1}$$

where c_i is the individual cost of the single action to i when travelling alone. In this work, we take this to be equal to the duration of the journey from A to B.

This is designed to approximately model the discount for the passengers if they buy a group ticket: The more agents travel together, the cheaper the shared (leg of a) journey becomes for each agent. Also, an agent cannot travel any cheaper than 20% of the single-agent cost. In reality, pricing for group tickets could vary, and while our experimental results assume this specific setup, the actual price calculation could be easily replaced by any alternative model.

The experiments are based on running the algorithm on one core of a 3.2 GHz Intel Core i7 processor of a Linux desktop computer. The data, source code, and scenarios in PDDL for the timetabled transport ridesharing problem are openly available in a repository[1] under LGPL license[2].

[1] https://github.com/agents4its/ridesharingontimetabledtransport
[2] http://www.gnu.org/licenses/lgpl.html

6.4.2 Experiment Scenarios

We used the following parameters as factors in experiment scenarios: (1) travel demand generation; (2) ridesharing demand proportion; (3) modes of transport considered; (4) maximum travel group size. The values of the parameters are summarised in Table 6.2. We set the maximum bearing difference parameter of the algorithm to $\Delta\varphi = 25$ degrees for all scenarios.

Table 6.2: Experiment scenarios parameters overview.

Scenario parameter	**Parameter values**
Travel demand generation	{realistic, random}
Ridesharing demand proportion	{100%, 50%, 5%}
Modes of transport	{trains only, trains and coaches}
Maximum travel group size	$\{2, 4, 6, 8\}$

Travel Demand Generation. We use two types of travel demand. The *realistic* travel demand generation is based on the UK census 2001 origin-destination data [86] that contains numbers of trips carried out from every origin district to every other destination district. District-to-district trip counts are mapped to stop-to-stop trip counts in the following way: For each origin-destination district pair, the desired number of trips is generated randomly from the Cartesian product of stops in the origin and destination district. Since the UK census origin-destination data is not provided at the level of granularity required to select concrete stops in the travel network, we have sampled these within each district with probability proportional to the density of services passing through a stop. This is based on the assumption that service density roughly follows numbers of travellers using a stop. In addition to the realistic demand, we also experimented using *random* travel demand generated randomly from the Cartesian product of stops in the Yorkshire area, assuming a uniform distribution over stops.

From the travel demand distribution generated, only trips with a straight-line distance between the origin and the destination in the interval 25–100 km are used for the evaluation (when using roads or rail tracks, this interval stretches approximately to a real distance of 40–160 km). This interval was chosen to filter out trips that are too short to be planned in advance and therefore not very suitable for sharing. This led to the removal of 86% of all trips in the realistic travel demand generation process and to the removal of 30% of all trips in the random travel demand generation process.

Ridesharing Demand Proportion. In order to observe the behaviour of the system with different densities of trips we set the portion of travel demand to 100%, 50%, and 5% of the total number of trips.

Modes of Transport. In order to evaluate the behaviour of the algorithm on both a unimodal and a multimodal public transport network, *trains only* and a combination of *trains and coaches* were used in the experiments. In the Yorkshire area, there are 150 (201) stops, 330 (495) connections in the relaxed domain, and 9 881 (10 289)

connections in the timetable for trains (and coaches).

Maximum Travel Group Size. The maximum travel group size $n_{\max}$ is one of the algorithm's inputs that restricts the size of groups created in the trip grouping phase. We set this parameter to 2, 4, 6, and 8 as after initial testing, it became clear that groups of a larger size are almost never practicable.

6.4.3 Evaluation Metrics

We evaluate the performance of the algorithm in terms of three different metrics: improvement in the cost of agents' journeys, their prolongation, and the computation time of the algorithm.

Cost Improvement. To evaluate the net benefit of using our method for ridesharing, we calculate the cost improvement for the agents' journeys. To calculate this, recalling that $C_i(\pi) = \sum_j c_i(a^j)$ for a plan is the cost of a plan $\pi = \langle a^1, \ldots, a^k \rangle$ to agent i, assume $n(a^j)$ returns the number of agents with whom the jth step of the plan is shared. We can define the cost of a shared travel plan $C_i'(\pi) = \sum_j c_{i,n(a^j)}(a^j)$ using equation (6.1). With this, we can calculate the cost improvement ΔC as follows:

$$\Delta C = \frac{\sum_{i \in N} C_i(\pi_i) - \sum_{i \in N} C_i'(\pi_N)}{\sum_{i \in N} C_i(\pi_i)} \tag{6.2}$$

where N is the set of all agents, π_i is the single-agent plan initially computed for agent i, and π_N is the final joint plan of all agents after completion of the algorithm (which, though it is in reality a set of several plans for different subgroups of N, is interpreted as a single plan for the "grand coalition" N and reflects how subgroups within N share parts of their individual journeys).

Prolongation. On the one hand, ridesharing is beneficial in terms of cost. On the other hand, a shared journey has a longer duration than a single-agent journey in most cases, because agents have to take later services than they could use on their own if they are waiting for co-travellers to arrive. In order to evaluate this trade-off, we measure journey prolongation. Assume that $T_i(\pi)$ is the total duration of a plan to agent i in plan π, and, as above, π_i/π_N denote the initial single-agent plans and the shared joint plan at the end of the timetabling phase, respectively. Then, the prolongation ΔT of a journey is defined as follows:

$$\Delta T = \frac{\sum_{i \in N} T_i(\pi_N) - \sum_{i \in N} T_i(\pi_i)}{\sum_{i \in N} T_i(\pi_i)} \tag{6.3}$$

Computation Time. To assess the scalability of the algorithm, we measure the amount of time needed to create groups of agents in the first phase of the algorithm and then to plan shared journeys for all agents in each group.

6.4.4 Results

In this section, we present the results of the evaluation in terms of journey cost improvement, journey prolongation, and computation time of the algorithm. Exhaustive

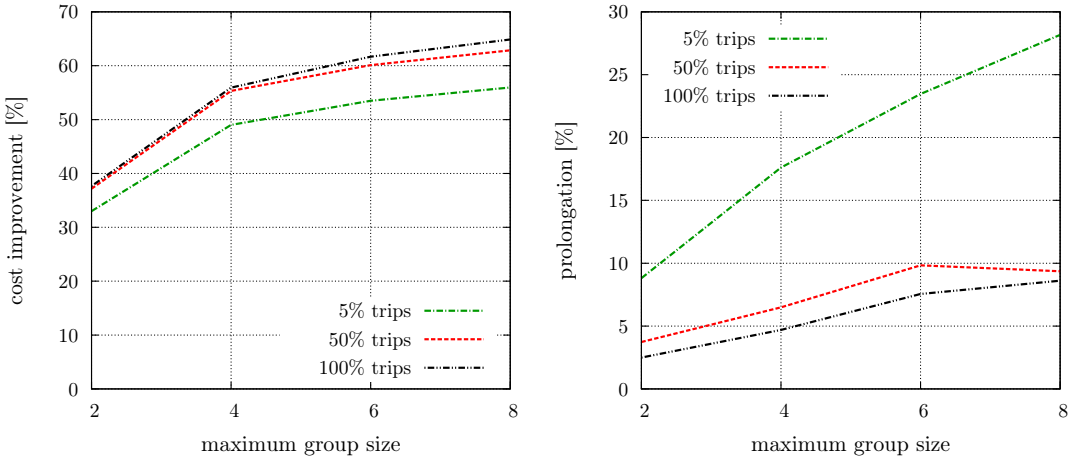
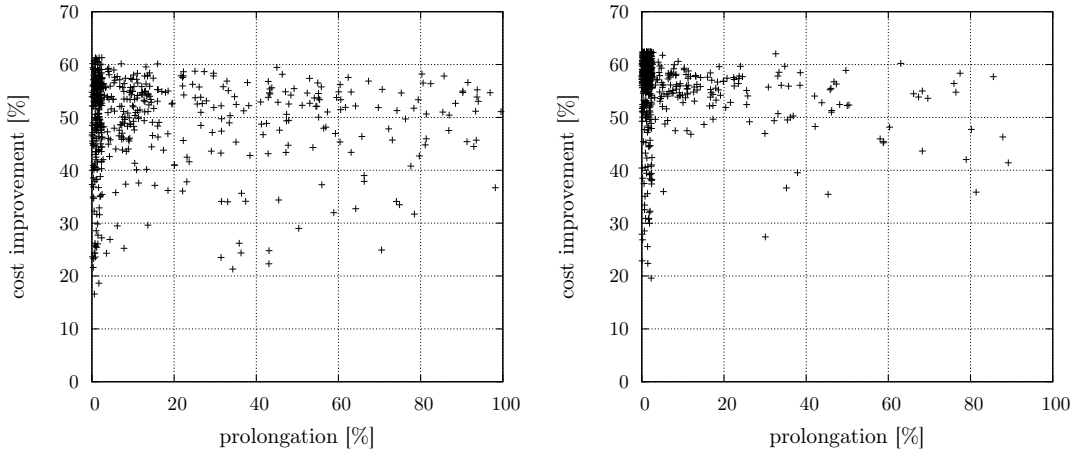


Figure 6.7: Average cost improvement and prolongation against maximum group size (realistic travel demand, trains and coaches).

experiment design was used; all metrics were evaluated for all combinations of all values of all scenario parameters. Specifically, for each type of travel demand generation, we tested all combinations of modes of transport, and for each ridesharing demand proportion, we generated the travel demand. Then for each maximum group size, the whole travel demand is an input for the algorithm which in its trip grouping phase creates the groups of agents for ridesharing. From the set of all groups created, a detailed journey plan with a timetable is found in the last three phases of the algorithm for a sample of 80 randomly chosen groups (sampling was performed to reduce experiment computational time while maintaining significance of the results). Each possible experiment configuration is averaged over 8 stochastic travel demand generation instances. This leads to an overall number of 30 720 groups of agents over which the algorithm was evaluated.

Cost Improvement. The average cost improvement obtained in our experiments is shown in Figure 6.7. It shows that the more agents are grouped together in the trip grouping phase of the algorithm, the higher the improvement. These results were obtained based on the specific cost function (6.1) we have introduced to favour ridesharing, and which would have to be adapted to the specific cost structure that is present in a given transport system. Also, the extent to which longer journey times are acceptable for the traveller depends on their preferences, but these could be easily adapted by using different cost functions.

Prolongation. The average prolongation of journeys is shown in Figure 6.7 where 8% of groups with prolongation greater than 100% is filtered out from the average calculation (these are the journeys which, though feasible, are unlikely to be accepted by travellers). The graph shows that the more agents are grouped together in the trip grouping phase of the algorithm, the higher the prolongation. Furthermore, the prolongation with the 5% ridesharing proportion is much higher then when consider-



(a) 5% trips (b) 50% trips

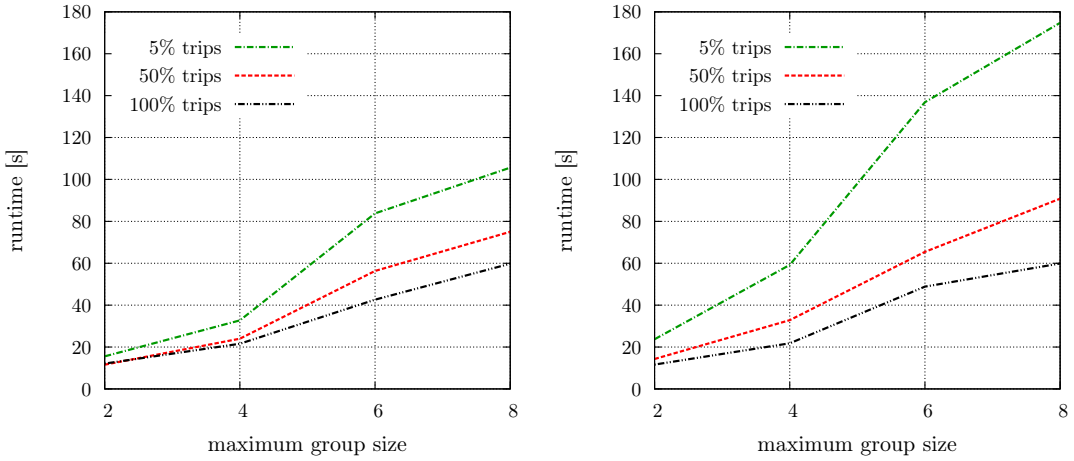
Figure 6.8: Cost improvement against prolongation (realistic travel demand, trains and coaches, maximum group size $n_{\max} = 4$).

ing 50% or 100% ridesharing proportion. As the density of trips drops, the agents in groups are more spatially dispersed, which causes higher relative prolongation ratios.

Figure 6.8 shows a scatter plot of cost improvement versus prolongation for individual trips for 5% and 50% ridesharing proportion. It can be observed that with a higher ridesharing proportion, the majority of the groups has either prolongation very close to 0% (identical trips are shared) or has a very high cost improvement (between 50% and 60%). With a lower ridesharing proportion, there are many more groups with lower cost improvement or higher prolongation. What is encouraging is that even for small populations of potential ridesharers, there are many shared journeys with a good cost improvement and a reasonable prolongation. In our algorithm, the balance between the two criteria could be calibrated by changing the weights in the cost function.

Computation Time. The first graph in Figure 6.9 shows the overall computation times of the algorithm for one created group of agents from the realistic demand and a combination of trains and coaches. The trip grouping phase of the algorithm is very fast (200 ms per group on average). The algorithm spends the majority of the computation time solving the problem of finding a joint plan for the group of agents. The graph indicates that the overall computation time grows roughly linearly with increasing numbers of agents in a group, which confirms that the algorithm avoids the exponential blowup in the action space characteristic for centralised multiagent planning. This is mainly a consequence of the best-response planning algorithm, and an expected result.

The second graph in Figure 6.9 shows the overall computation times of the algorithm for one created group of agents from the random demand and for a combination of trains and coaches. It can be observed that the algorithm is faster at the realistic



(a) Realistic travel demand

(b) Random travel demand

Figure 6.9: Computation time against maximum group size (trains and coaches).

5% ridesharing proportion than for random trips. At the 50% and 100% ridesharing proportion, there is not a very big difference between the computation times. This suggests that the trips from the realistic 5% ridesharing proportion reflects the public transport network making the journey planning easier whereas it is harder to plan for trips distributed randomly.

Regarding the modes of transport in the scenario, it is harder to find joint plans when a combination of trains and coaches is considered (on average, runtimes are 25% higher for scenarios with trains and coaches). Considering a combination of trains and coaches does not significantly affect neither the cost improvement, nor the prolongation.

While the overall computation times are considerable (up to 2 minutes for a group of 8 agents from the realistic 5% ridesharing proportion), we should emphasise that the algorithm is effectively computing equilibrium solutions in multi-player games with hundreds of thousands of states. Considering this, the linear growth hints at having achieved a level of scalability based on the structure of the domain that is far above naive approaches to plan jointly in such state spaces.

Finally, we have evaluated the overall computation time for all trips from the travel demand. We were able to compute shared journey plans for approximately 13 500 trips from realistic 100% ridesharing proportion when considering a combination of trains and coaches. It took less than 75 minutes for each setting of maximum group size while using 8 cores of 3.2 GHz Intel Core i7 processor on three computers in parallel.

The ridesharing algorithm can be further parallelised down to a level of individual groups, bringing the computation time to few minutes for the whole demand. This follows from the structure of the algorithm: In the trip planning phase, the identification of initial single-agent plans for the travellers consists of a set of completely

independent problems. In the best-response and timetabling phase, the planning problem of each group of agents is also completely independent from those of other groups or individual travellers.

6.4.5 Discussion

Our proposed algorithm clearly improves the cost of agents' journeys by sharing parts of the journeys, even though there is an inherent trade-off between cost improvement and the prolongation of journeys. On the one hand, the bigger the group, the better the improvement. On the other hand, the more agents share a journey, the higher the prolongation is likely to be. This will most likely lead to results that are not acceptable for users in larger groups. Whether prolongation or cost savings are more important in a given scenario will depend on the real preferences of travellers, and our system would allow them to customise these settings per individual on, for example, a Web-based ridesharing planner that would use our algorithm. It is also important to point out that our framework can be used without any significant modifications for any other cost function as appropriate for the transport system in question, and, subject to availability of the required real-world timetabled transport data, for any other geographical region.

Next, note that trip planning and best-response phases of the algorithm are completely domain-independent and can therefore easily be used for other types of transport problems, e.g., to plan routes that avoid traffic jams or to schedule parcel deliveries. What is more, additional constraints such as staying at a location for some time or travelling together with a specific person can be easily accommodated within standard planning languages, and the use of standard planning technology also implies that our method will directly benefit from future improvements in planning algorithms. On the other hand, the trip grouping and the timetabling phase of the algorithm are domain-specific, providing an example of the specific design choices that have to be made from an engineering point of view when applying standard AI methods to problems of decentralised decision-making in transport.

From an algorithm and systems engineering perspective, using off-the-shelf problem solvers such as AI planning systems for a complex real-world domain like ridesharing brings an additional benefit, which is that we do not need to engineer novel optimisation algorithms for the combinatorial problems arising in this family of problems *from scratch*. While it is certainly possible that faster algorithms that produce better solutions may exist for specific problems, our approach enables us to formalise different types of similar problems with comparatively little effort and to make use of the best available search heuristics in a lightweight fashion. We believe that this is an effective way of developing resource allocation and process optimisation systems in domains like transport, where it can be expensive to develop a custom solution for every different class of problems although many of them share many common characteristics.

The presented experiments work with a demand for a *whole day* and therefore the generated joint plans are not restricted to any particular part of a day. In reality,

however, travellers may not be so flexible in terms of timing their journeys. This problem can be easily solved by considering time constraints in the clustering performed in the trip grouping phase of the algorithm. Trips might for instance be put into one group only if their preferred departure and/or arrival times do not differ by more than a given time difference. The performance achieved for lower trip densities corresponding to 5% ridesharing proportion suggests that attractive shared journeys would be found even for the maximum time difference of one hour (one hour constitutes approximately 4% of a day), or even less during peek hours when demand is more concentrated.

There are many potential uses of the proposed approach to real-world ridesharing: From a traveller's perspective, it can be used to exploit current ticket discounts for group travel while enjoying the company of friends, fellow workers, and other co-travellers. A web- or smartphone-based application can be built which would collect user preferences and constraints and propose shared journey plans. Further, in future applications our approach could be combined with the use of private cars to mix public and private modes of transport. This can be achieved with fairly small modifications. It would work in a similar way as walking is combined with public transport in our evaluation, except that car travel enables non-timetabled transport for more than one individual. This is an important extension, as in most realistic settings, successful ridesharing would certainly include private cars. In fact, without this, we are only considering a very hard problem, were travellers have very limited flexibility. Naturally, if at least one person in each group has a car, this opens up (orders of magnitude) more options for joint trips. Also, for car sharing the cost benefit is arguably much higher, and can be much more objectively calculated than what we have assumed in our hypothetical cost function (e.g., cost/km divided by number of car passengers).

From a public policy and transport planning perspective, stakeholders in the public transport domain could use our method to predict customer behaviour when considering modifications to timetables, the introduction of new services, and modifications to pricing schemes to optimise usage, environmental footprint, and business revenue. Such scenario analysis could easily accommodate taking further factors into account, such as waiting times, travel interruptions for business and leisure activities, preferences of individuals to share trips with particular co-travellers, etc. In particular, it could give rise to new incentive schemes for ridesharing, such as discounts for group travel that depend on the cumulative amount of sharing or occupancy ratios along different legs of joint journeys involving various modes of transport and changing groups of jointly travelling individuals.

6.5 Contributions and Summary

In this chapter, we have presented a multiagent planning algorithm which is able to plan meaningful shared journeys using timetabled public transport services. The contribution of our work with respect to state-of-the-art techniques described in Sec-

tion 2.5 is threefold:

Firstly, we have applied state-of-the-art automated planning techniques to real-world public transport data to evaluate their feasibility in this domain. This allows to consider the inclusion of multimodal public transport services in ridesharing activities and to explore opportunities to combine their use with private cars in the future. In addition to serving as a basis for traveller-oriented ridesharing service, our system allows stakeholders to determine appropriate pricing policies to incentivise group travel and to predict the effects of potential service changes.

Secondly, we have presented an algorithm that combines different techniques in a practically-oriented way and works with real-world public transport timetables and realistic travel demand even though it is largely based on extensible, domain-independent, off-the-shelf heuristic problem solvers. The four-phase algorithm utilises performant single-agent planners to find individual plans in a simplified domain first and then merges them using a best-response planner which ensures resulting solutions are individually rational.

Thirdly, the algorithm has been implemented and evaluated on realistic scenarios based on real-world UK transport data. In the process, we have described the engineering steps that are necessary to deal with the challenges of real-world large-scale data and propose suitable solutions. Experiments with realistic travel demand show that, for a wide range of scenarios, the algorithm is capable of finding shared journeys with very attractive trade-offs between cost saving and journey time prolongation. The algorithm exhibits very good scalability. It scales linearly with the number of trips processed, regardless of the size of travel groups considered. The algorithm is also amenable to massive parallelisation which can bring the time required for planning shared journeys for real-world travel demand down to minutes.

The model for the timetabled transport ridesharing problem, the ridesharing planning algorithm, and the evaluation of the algorithm has been designed in collaboration with Michael Rovatsos. Michael has also contributed to related work regarding multiagent trip planning in Section 2.5. Finally, this topic has been published in the following articles [64, 65].

Chapter 7

Engineering Aspects

This chapter describes practical experience on the intersection of research and engineering related to development of journey planning systems. We first address a part of Research objective 2 and present our experience with memory-efficient data structures, optimisation of the shortest path algorithms, and deployment of the journey planning systems. Then we focus on Research objective 3 and present lessons learned related to the importance of testing the implemented algorithms in the real environment and how to achieve high-quality journey planning.

7.1 Engineering Routing Algorithms in Java

Traditionally, routing algorithms are implemented in C++ (e.g., Open Source Routing Machine[1]) or in Java (e.g., OpenTripPlanner[2]). Compared to C++, Java offers us very good portability (important for the evaluation of the algorithms in the computation cluster) and server-side ecosystem (important for running the backend parts of the routing systems). Java is also used at our department as the main programming language and it was used in the SUPERHUB project (see Section 5.4). Because of these reasons, we have implemented the algorithms presented in this thesis in Java programming language. Based on our experience, we discuss engineering aspects of the Java implementation related to data structures, algorithms, and deployment.

7.1.1 Memory-Efficient Data Structures

Since the real-world transport networks contain hundreds of thousands of nodes and edges, it is necessary to use memory-efficient data structures to represent the planning graphs. In this section, we show how we were able to lower memory requirements of the GTD graph (see Section 3.2.2) to one sixth compared to a straightforward implementation. We tested the memory optimisations on two GTD graphs with 3 days of timetables. At the beginning, Prague GTD graph had 693 MB while Milan

[1] http://project-osrm.org/
[2] http://www.opentripplanner.org/

GTD graph had 1218 MB using the straightforward representation. In the following list, we describe the key optimisations we have performed:

- Initially, we used a `long` identifier for each node that corresponded to the OSM identifier. The disadvantage was that the identifiers of the nodes did not constitute an increasing sequence. We have improved this by switching to an `integer` identifier and having node identifiers from 0 to $|V|-1$ where $|V|$ is the number of nodes in the graph. This enabled us to use arrays instead of maps in the representation of the graph.
- In the graph object, we have replaced `Map<Long, TNode>` with nodes representation in `ArrayList<TNode>`.
- We have improved the representation of outgoing edges by replacing `Map<Long, List<TEdge>>` with edge list `ArrayList<TEdge>` and `int[]` representing starting positions in the edge list for the adjacency list of each node.
- Each edge in the GTD graph is annotated with the allowed modes of transport. We have encoded the `Set<ModeOfTransport>` into a single `int` using bit operations.
- In the straightforward implementation of the PT timetables, we have used `TreeSet<Departure>` to capture all departures on a route edge (defined in Section 3.2.1). Each `Departure` object contained a `DateTime` object for departure time and `int` for the duration. With the defined comparator for the `Departure`, it was possible to call `higher` method of the `TreeSet` to get easily the next departure given a specified time. We have replaced this with `int[]` to capture the departures now represented as seconds from the start of the timetable validity. In addition, if all the durations on the route edge are the same, we use just one `int` to capture the duration; otherwise we use single `int` to specify duration for each departure on the route edge as before.

After performing the listed improvements, we measured the memory consumption again. We managed to lower the memory size of Prague GTD graph to 113 MB (16% of the original size) and the memory size of Milan GTD graph to 220 MB (18% of the original size). In conclusion, the data structures need to be carefully designed to capture urban-scale GTD graphs with the minimum amount of memory and to enlarge the graphs to country-scale in the future.

7.1.2 Heaps Choice for Shortest Path Algorithms

There are several points where the implementation of shortest path algorithms can be fine-tuned. In this section, we discuss the choice of a heap for the Dijkstra's and A* algorithms. There are two main candidates: *Fibonacci heap* and *binomial heap*. The main operations used during the search are insert, decrease key, and delete the minimum. In theory, the Fibonacci heap has a better time complexity then the

binomial heap for the decrease key operation. However, when implemented for the Dijkstra's and A* algorithm in Java, the results of our experiments using Helsinki transport network were the opposite. The binomial heap offered by 5% better search runtimes than the Fibonacci heap. In this case, the number of objects inserted in the heap is too small to exploit the better theoretical performance of the Fibonacci heap (although the Fibonacci heap decrease key operation works in constant time, the internal operations are more demanding than in the binomial heap). This shows that in some cases, we cannot decide just according to theoretical complexity but experimenting with the implemented data structures and algorithms is needed.

7.1.3 Deployment of Journey Planners

The last step to run a routing system backend is to deploy the system on a server. We first build the whole system as a WAR file (Web application ARchive) using Maven[3] tool that is used to manage the dependencies. Then we deploy the WAR file at an Apache Tomcat[4] server. Then the backend is ready to be queried by web or mobile clients using the RESTful API. In order to have a comprehensive overview about all endpoints of the API, we use Swagger Specification[5]. The specification shows all endpoints with supported methods (e.g., GET, POST) together with a description of the functionality. Thus, the API is comprehensible for all client developers. An example of bicycle routing system API specification is shown in Figure 7.1.

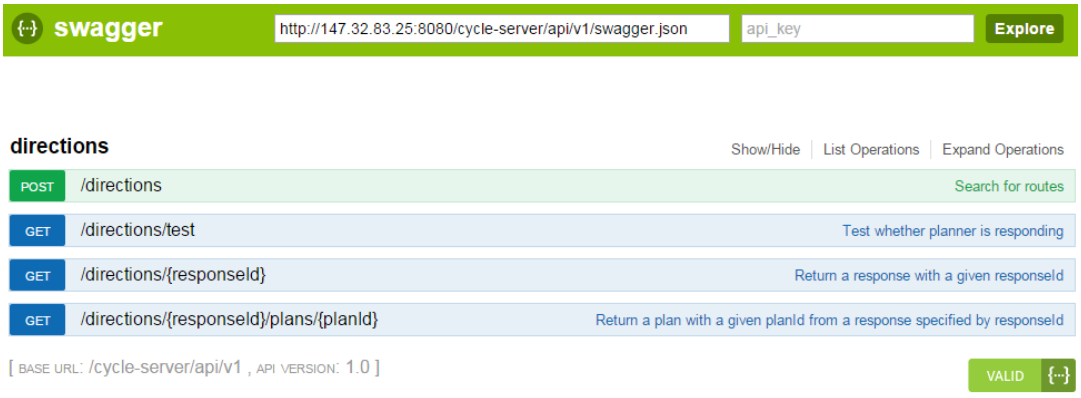


Figure 7.1: An example of bicycle routing system API specification using Swagger.

On the testing side, performance of the system is tested using Apache JMeter[6] that enables to send arbitrary number of requests to the API of the routing system. This helps us to check the behaviour of the system under certain load. As an example in Figure 7.2, we show a graph produced by JMeter during experiments with stress testing of bicycle routing API. We have created a load of 10 threads where each of

[3] http://maven.apache.org/
[4] http://tomcat.apache.org/
[5] http://swagger.io/ – Since 1st January 2016, it has been renamed to OpenAPI Specification.
[6] http://jmeter.apache.org/

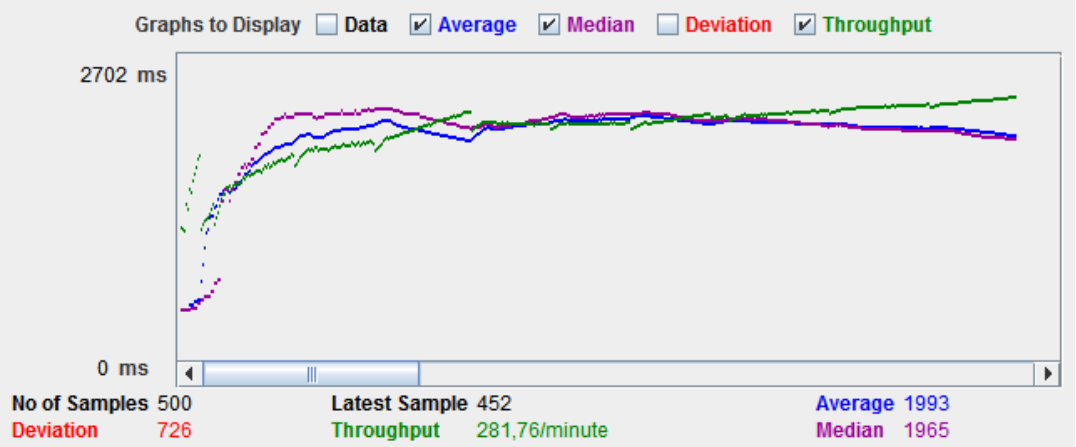


Figure 7.2: An example of the bicycle routing API stress testing results.

them performed 50 requests in a sequence. To conclude, we have also set up an automated service to detect failures in the journey planning backend. We have been promptly informed about the problems so we managed to keep the downtimes of the API reasonably low.

7.2 Importance of Testing in the Real Environment

We have learned that without deploying our journey planning techniques in the operational environment (TRL 7) we cannot get to know all issues related to the model, algorithm, and its implementation. Deploying in the operational environment opens several ways to test the solution.

First, the solution can be tested by thousands of people that use the planner using a Mobile or a Web App. From the users, we can get a valuable feedback on the returned plans. Users help us to identify plans that do not make sense or contain for example a dangerous road for cyclists. This feedback can help to identify the problems with the parameters and results of the routing algorithms and especially with the underlying data used to build the planning graphs. The plans suggested by the routing algorithm can be also reviewed by the experts, e.g., bicycle routing system has been reviewed by experts on cycling from AUTO*MAT NGO.

Second, various data can be gathered from the people. As an example, we have gathered the GPS bicycle tracks (see Figure 4.12) that can be used to compare the suggested plan and the route taken by the cyclist in the real environment.

Third, we can observe the performance of the algorithms under real demand and identify bottlenecks in both runtime and memory requirements.

Fourth, the identified issues are solved immediately during the development process or they can lead to open research questions and next research steps (described in Section 8.1). To summarise, the testing in the real-world environment enables to perform a closed-loop research process shown in Figure 7.3.

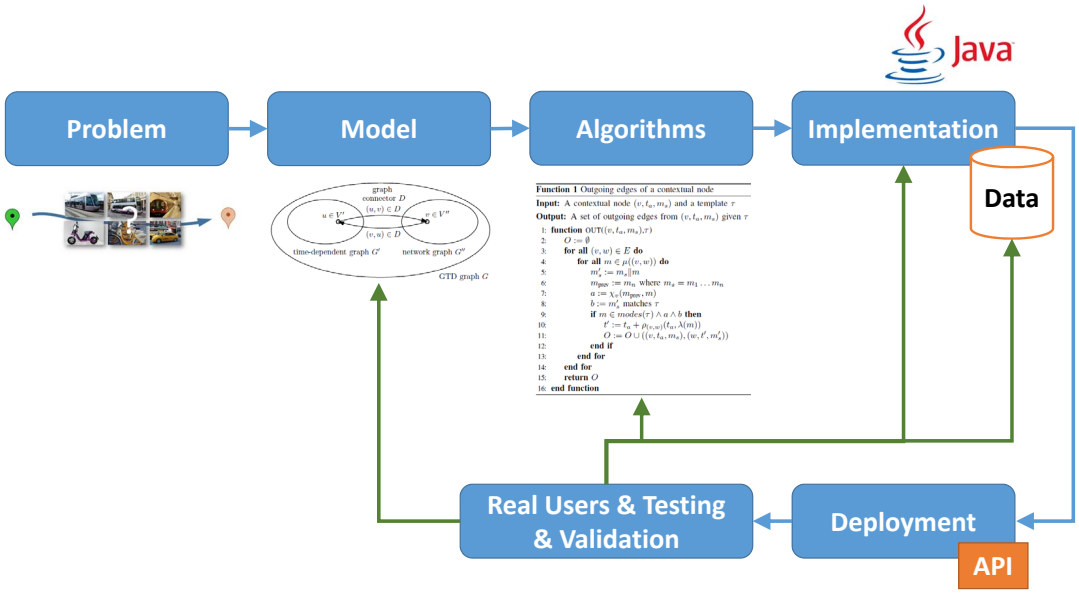


Figure 7.3: The research process is enclosed in a loop by using feedback (green arrows) from deployment in the real environment.

7.2.1 Maximizing Plan Quality in Journey Planning

Importing, validating, and integrating various data sources required for journey planning is a complex and error-prone task. Even minor inconsistencies in underlying data, their mutual mapping or translation into internal data models can severely degrade the quality of resulting journey plans. Data import and (semi-)automated validation tools are thus essential to build high-quality planning graphs for the routing systems.

We have developed a data import and validation pipeline that is depicted in Figure 7.4. At first, a preliminary planning graph is built based on OSM maps and GTFS timetables. Then, we perform a consistency check to detect a range of inconsistencies affecting the quality of the planning graph. We check the following:

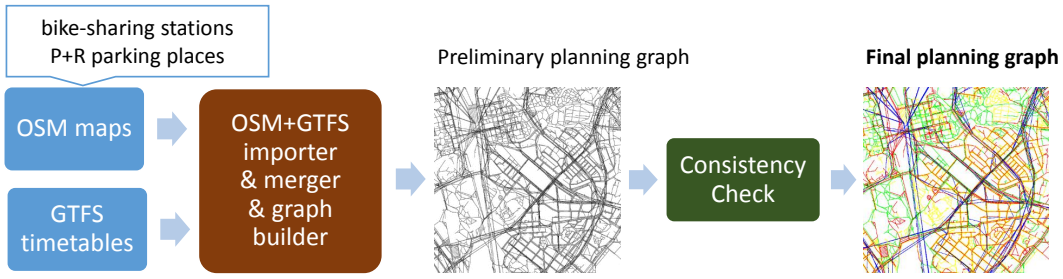


Figure 7.4: Data import pipeline to build a high-quality planning graph.

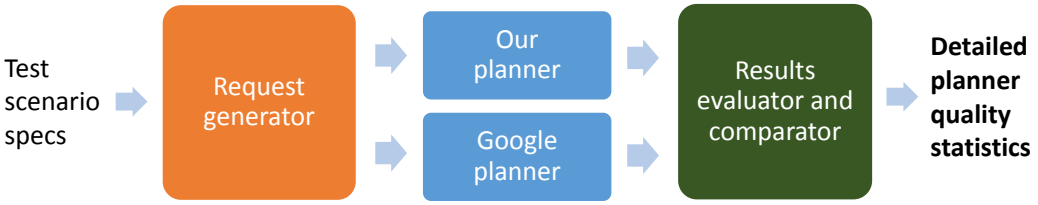


Figure 7.5: Automated planner benchmarking against external planning services.

zero and negative travel times, edge travel speed, presence of PT modes, strongly connected components, and absence of loops. If problems are detected, the data needs to be corrected and the whole pipeline repeated. Once the data are corrected, we progress to the construction of final planning graph.

Finally, the journey planning quality is tested using an automated planner benchmarking against external planning services. The quality testing uses Google Directions API and its schema is shown in Figure 7.5. Using this approach can help us discover problems both in journey planning algorithms (for example wrongly handled number of transfers) and in underlying data (for instance missing buses in the transport network). For more details see Section 5.4.1.

Chapter 8

Conclusions

In this work, we have investigated the models and algorithms for *journey planning for sustainable transport*, i.e., planning journeys from an origin to a destination that respect user preferences and utilise a combination of sustainable modes of transport such as (shared) bike, (shared) electric scooter, shared car, or public transport.

We have been solving three related problems of journey planning for sustainable transport. By researching the area of sustainable journey planning from different perspectives, we managed to gain an integrated view on intermodal journey planning. We are now able to draw upon this experience and build advanced journey planning systems as discussed in Section 8.1.

Below, we summarise the most important achievements of the thesis corresponding to the research objectives defined in Section 1.1. The achievements are also summarised in a matrix in Table 8.1. Each of the researched sustainable journey planning topics is associated with its model, algorithm, deployment, and published articles. More detailed summary of our contributions related to the three studied problems is available in Section 4.6, Section 5.5, and Section 6.5. These sections also specify contributions of people that have collaborated on the studied problems.

1. **Formal models.** In Chapter 3, we have formalised the models to represent both static and time-dependent transport networks as graphs. Our first contribution is the well-grounded formal model of multi-criteria bicycle routing problem that is based on the *cycleway graph*. Our second contribution is the *generalised time-dependent graph* which allows representing the combined static and time-dependent transport networks in a single model. The models gave us solid understanding of the transport domain and enabled development of efficient algorithms for the three following sustainable journey planning problems: *multi-criteria bicycle routing problem*, *intermodal earliest arrival problem*, and *timetabled transport ridesharing problem*.

2. **Efficient algorithms.** In the thesis, we have proposed three algorithms to solve the instances of sustainable journey planning problems defined by Research objective 1. First, the *heuristic-enabled multi-criteria label-setting algo-*

Table 8.1: Thesis achievements matrix.

	Multi-Criteria Bicycle Routing (Chapter 4)	**Intermodal Journey Planning** (Chapter 5)	**Ridesharing on Timetabled Transport Services** (Chapter 6)
Formal models	Cycleway Graph (Section 3.1.2) Multi-Criteria Bicycle Routing Problem	Generalised Time-dependent Graph (Section 3.2.2) Intermodal Earliest Arrival Problem	Relaxed and Full Transport Services Domain Timetabled Transport Ridesharing Problem
Efficient algorithms	HMLS Algorithm with Speedups	Intermodal Planning Algorithm	Ridesharing Planning Algorithm
Validation in real-world deployments	Bicycle Routing System	Intermodal Journey Planning System	-
Published in	[66, 67, 68, 103]	[63, 71]	[64, 65]

rithm together with a combination of heuristics solves the multi-criteria bicycle routing problem. Second, the *intermodal planning algorithm* using a generalised time-dependent graph structure solves the intermodal earliest arrival problem. Finally, the *ridesharing planning algorithm* using state-of-the-art automated planning techniques solves the timetabled transport ridesharing problem. Findings related to memory-efficient data structures and implementation of the algorithms are discussed in Section 7.1. All implemented algorithms for sustainable journey planning has been thoroughly evaluated in an exhaustive set of experiments using real-world map and PT timetables data. Runtime of the algorithms, quality of returned plans and other metrics have been measured, presented, and discussed.

3. **Validation in real-world deployments.** Finally, in Section 4.5 and Section 5.4, we have presented how the *Bicycle Routing System* and *Intermodal Journey Planning System* have been validated in real-world deployments with real users. It has been validated that the systems are capable of working in an operational environment (TRL 7) and they have the potential to help people with the navigation on their urban journeys. Based on deployments, we have also summarised engineering aspects of developing journey planning systems in Chapter 7. The key lessons learned are related to testing in real environment and the need for high-quality planning graphs. The experience is valuable with respect to the future work in the sustainable journey planning field.

8.1 Future Work

In this section, we list several directions worth exploring in the future that has emerged during the work on the thesis. We conclude this section by looking more ahead to the trend of seamless door-to-door mobility that can be supported by our work on journey planning for sustainable transport.

The multi-criteria search often produces large Pareto sets with many similar routes. We therefore plan to provide a filtering method (e.g., based on our preliminary work that uses clustering [103]) that would select several representative routes from a potentially very large set of Pareto routes. This will be useful for the multi-criteria bicycle routing problem as well as for the use of multi-criteria search in intermodal journey planning.

We aim to incorporate real-world GPS tracks we have been collecting into the routing process (e.g., according to [29]). This should further improve the quality of route suggestions by making it possible to consider route-choice factors that cannot be reflected based solely on the underlying map data. Furthermore, comparing real-world GPS tracks with the transport network graph is a powerful way of (semi-)automated detection of problems in transport network data.

In order to deliver high-quality journey plans to users, it is necessary to build journey planning systems on top of high-quality planning graphs that represent transport networks. Based on our experience, importing, validating, and integrating various data sources required for journey planning is a complex and error-prone task. In addition, the transport network changes over time, e.g., new cycleways are built or PT timetables are updated. We would like to extend our data import and validation pipeline (described in Section 7.2.1) to an automated tool that would build, validate, and also maintain a high-quality planning graph.

In the long run, we would like to contribute to the concept of *seamless door-to-door mobility* where the users are provided with intelligent journey planning tools that arrange the whole door-to-door journey. Importantly, this concept employs resource allocation for automated seat reservation and ticket purchase using booking and ticketing APIs provided by mobility services operators. We have started to explore this concept by researching the combination of several existing journey planner APIs to deliver intermodal journey planning functionality [83]. We could also use our experience with ridesharing to enable seamless door-to-door ridesharing.

In the thesis, we have made scientific contributions to solving several journey planning problems. Equally importantly, we have demonstrated that our scientific results can be translated into real-world solutions, which can ultimately facilitate the critically needed shift towards sustainable mobility.

Bibliography

[1] Mobility 2001: World Mobility at the End of the Twentieth Century and Its Sustainability. Technical report, Massachusetts Institute of Technology and Charles River Associated Incorporated, 2001.

[2] H. Bast. Car or Public Transport – Two Worlds. In S. Albers, H. Alt, and S. Naher, editors, *Efficient Algorithms*, volume 5760 of *Lecture Notes in Computer Science*, pages 355–367. Springer Berlin Heidelberg, 2009.

[3] H. Bast, M. Brodesser, and S. Storandt. Result Diversity for Multi-Modal Route Planning. In *13th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS)*, pages 123–136, Sophia Antipolis, France, 2013.

[4] H. Bast, E. Carlsson, A. Eigenwillig, R. Geisberger, C. Harrelson, V. Raychev, and F. Viger. Fast Routing in Very Large Public Transportation Networks Using Transfer Patterns. In M. Berg and U. Meyer, editors, *European Symposium on Algorithms (ESA)*, volume 6346 of *Lecture Notes in Computer Science*, pages 290–301. Springer Berlin Heidelberg, 2010.

[5] H. Bast, D. Delling, A. V. Goldberg, M. Muller-Hannemann, T. Pajor, P. Sanders, D. Wagner, and R. F. Werneck. Route planning in transportation networks. *CoRR*, 2015.

[6] H. Bast, S. Funke, D. Matijevic, P. Sanders, and D. Schultes. In Transit to Constant Time Shortest-Path Queries in Road Networks. In *Meeting on Algorithm Engineering and Experiments (ALENEX)*. SIAM, 2007.

[7] H. Bast, S. Funke, P. Sanders, and D. Schultes. Fast Routing in Road Networks with Transit Nodes. *Science*, 316(5824), 2007.

[8] R. Bauer and D. Delling. SHARC: Fast and robust unidirectional routing. *ACM Journal of Experimental Algorithmics*, 14, 2009.

[9] R. Bauer, D. Delling, and D. Wagner. Experimental study of speed up techniques for timetable information systems. *Networks*, 57(1):38–52, 2011.

[10] G. Berbeglia, J. F. Cordeau, and G. Laporte. Dynamic pickup and delivery problems. *European Journal of Operational Research*, 202(1):8–15, 2010.

[11] C. Boutilier and R. Brafman. Partial-order planning with concurrent interacting actions. *Journal of Artificial Intelligence Research*, 14:105–136, 2001.

[12] R. Brafman and C. Domshlak. From One to Many: Planning for Loosely Coupled Multi-Agent Systems. In *International Conference on Automated Planning and Scheduling (ICAPS)*, pages 28–35. AAAI Press, 2008.

[13] R. Brafman, C. Domshlak, Y. Engel, and M. Tennenholtz. Planning Games. In *International Joint Conference on Artificial Intelligence (IJCAI)*, volume 21, pages 73–78, 2009.

[14] J. Broach, J. Dill, and J. Gliebe. Where do cyclists ride? A route choice model developed with revealed preference GPS data. *Transportation Research Part A: Policy and Practice*, 46(10):1730–1740, 2012.

[15] G. S. Brodal and R. Jacob. Time-dependent Networks as Models to Achieve Fast Exact Time-table Queries. *Electronic Notes in Theoretical Computer Science*, 92:3–15, 2004.

[16] I. Carreras, S. Gabrielli, D. Miorandi, A. Tamilin, F. Cartolano, M. Jakob, and S. Marzorati. SUPERHUB: A user-centric perspective on sustainable urban mobility. In *6th ACM workshop on Next generation mobile computing for dynamic personalised travel planning*, pages 9–10. ACM, 2012.

[17] V. Codina, J. Mena, and L. Oliva. Context-aware user modeling strategies for journey plan recommendation. In F. Ricci, K. Bontcheva, O. Conlan, and S. Lawless, editors, *User Modeling, Adaptation and Personalization*, volume 9146 of *Lecture Notes in Computer Science*, pages 68–79. Springer International Publishing, 2015.

[18] A. J. Coles, A. I. Coles, M. Fox, and D. Long. POPF2: a Forward-Chaining Partial Order Planner. In *International Planning Competition (IPC)*, 2011.

[19] K. L. Cooke and E. Halsey. The shortest route through a network with time-dependent internodal transit times. *Journal of Mathematical Analysis and Applications*, 14:493–498, 1966.

[20] J. F. Cordeau and G. Laporte. The dial-a-ride problem: models and algorithms. *Annals of Operations Research*, 153(1):29–46, 2007.

[21] D. W. Corne. The good of the many outweighs the good of the one: evolutionary multi-objective optimization. In *IEEE Connections Newsletter*, pages 9–13. IEEE, 2003.

[22] J. Cox and E. Durfee. An efficient algorithm for multiagent plan coordination. In *International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 828–835, Utrecht, The Netherlands, 2005.

[23] J. Cox and E. Durfee. Efficient and distributable methods for solving the multiagent plan coordination problem. *Multiagent and Grid Systems*, 5(4):373–408, 2009.

[24] M. de Weerdt and B. Clement. Introduction to planning in multiagent systems. *Multiagent and Grid Systems*, 5(4):345–355, 2009.

[25] B. C. Dean. Continuous-time dynamic shortest path algorithms. Master's thesis, Massachusetts Institute of Technology, 1999.

[26] D. Delling, J. Dibbelt, T. Pajor, D. Wagner, and R. F. Werneck. Computing and Evaluating Multimodal Journeys. Technical Report 2012-20, Faculty of Informatics, Karlsruhe Institut of Technology, 2012.

[27] D. Delling, J. Dibbelt, T. Pajor, D. Wagner, and R. F. Werneck. Computing Multimodal Journeys in Practice. In *International Symposium on Experimental Algorithms*, pages 260–271, 2013.

[28] D. Delling, J. Dibbelt, T. Pajor, and R. Werneck. Public transit labeling. In E. Bampis, editor, *Experimental Algorithms*, volume 9125 of *Lecture Notes in Computer Science*, pages 273–285. Springer International Publishing, 2015.

[29] D. Delling, A. Goldberg, M. Goldszmidt, J. Krumm, K. Talwar, and R. Werneck. Navigation Made Personal: Inferring Driving Preferences from GPS Traces. In *23rd International Conference on Advances in Geographic Information Systems (ACM SIGSPATIAL)*. ACM Press, 2015.

[30] D. Delling, A. Goldberg, and R. Werneck. Hub label compression. In V. Bonifaci, C. Demetrescu, and A. Marchetti-Spaccamela, editors, *Experimental Algorithms*, volume 7933 of *Lecture Notes in Computer Science*, pages 18–29. Springer Berlin Heidelberg, 2013.

[31] D. Delling, A. V. Goldberg, T. Pajor, and R. F. Werneck. Customizable Route Planning in Road Networks. *Transportation Science*, 2015.

[32] D. Delling, T. Pajor, and D. Wagner. Accelerating Multi-modal Route Planning by Access-Nodes. In A. Fiat and P. Sanders, editors, *European Symposium on Algorithms (ESA)*, volume 5757 of *Lecture Notes in Computer Science*, pages 587–598. Springer, 2009.

[33] D. Delling, T. Pajor, and R. F. Werneck. Round-Based Public Transit Routing. In D. A. Bader and P. Mutzel, editors, *Meeting on Algorithm Engineering and Experiments (ALENEX)*, pages 130–140. SIAM, 2012.

[34] D. Delling and D. Wagner. Time-dependent route planning. In R. Ahuja, R. Mohring, and C. Zaroliagis, editors, *Robust and Online Large-Scale Optimization*, volume 5868 of *Lecture Notes in Computer Science*, pages 207–230. Springer Berlin Heidelberg, 2009.

[35] J. Dibbelt, T. Pajor, B. Strasser, and D. Wagner. Intriguingly simple and fast transit routing. In V. Bonifaci, C. Demetrescu, and A. Marchetti-Spaccamela, editors, *Experimental Algorithms*, volume 7933 of *Lecture Notes in Computer Science*, pages 43–54. Springer Berlin Heidelberg, 2013.

[36] J. Dibbelt, T. Pajor, and D. Wagner. User-constrained multimodal route planning. *Journal of Experimental Algorithmics (JEA)*, 19:3.2:1–19, 2015.

[37] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1(1):269–271, 1959.

[38] Y. Dimopoulos, M. A. Hashmi, and P. Moraitis. μ-satplan: Multi-agent planning as satisfiability. *Knowledge-Based Systems*, 29(0):54–62, 2012.

[39] C. Dora and M. Phillips. *Transport, environment and health.* WHO Regional Office for Europe, Copenhagen, 2000.

[40] S. E. Dreyfus. An appraisal of some shortest-path algorithms. *Operations Research*, 17(3):395–412, 1969.

[41] E. Ephrati, M. E. Pollack, and J. S. Rosenschein. A Tractable Heuristic that Maximizes Global Utility through Local Plan Combination. In *1st International Conference on MultiAgent Systems (ICMAS)*, pages 94–101, 1995.

[42] Eurostat. Modal split of passenger transport, [Online], available at tinyurl.com/eurostat-modal-split, [Accessed: Oct 26, 2012].

[43] V. Filler. (AUTO*MAT) Private communication, 2013. Why cycle route planners are important for cyclists.

[44] J. Finnegan, M. Islam, C. Polisini, M. Jakob, and J. Hrncir. D5.2: Design and protocols for the real-time mobility resources negotiator, SUPERHUB deliverable, 2013.

[45] D. E. Foulser, M. Li, and Q. Yang. Theory and algorithms for plan merging. *Artificial Intelligence*, 57(2-3):143–181, 1992.

[46] M. L. Fredman and R. E. Tarjan. Fibonacci heaps and their uses in improved network optimization algorithms. *Journal of the ACM (JACM)*, 34(3):596–615, 1987.

[47] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman & Co., New York, USA, 1979.

[48] R. Geisberger. Contraction of timetable networks with realistic transfers. In P. Festa, editor, *Experimental Algorithms*, volume 6049 of *Lecture Notes in Computer Science*, pages 71–82. Springer Berlin Heidelberg, 2010.

[49] R. Geisberger. *Advanced Route Planning in Transportation Networks*. PhD thesis, Karlsruher Instituts für Technologie, 2011.

[50] R. Geisberger, P. Sanders, D. Schultes, and D. Delling. Contraction Hierarchies: Faster and Simpler Hierarchical Routing in Road Networks. In *7th International Conference on Experimental Algorithms*, pages 319–333. Springer-Verlag, 2008.

[51] M. Ghallab, D. Nau, and P. Traverso. *Automated Planning: Theory and Practice*. Morgan Kaufmann, 2004.

[52] A. V. Goldberg and C. Harrelson. Computing the shortest path: A search meets graph theory. In *16th ACM-SIAM Symposium on Discrete algorithms (SODA)*, Philadelphia, PA, USA, 2005. Society for Industrial and Applied Mathematics.

[53] A. V. Goldberg and R. F. Werneck. Computing Point-to-Point Shortest Paths from External Memory. In C. Demetrescu, R. Sedgewick, and R. Tamassia, editors, *Meeting on Algorithm Engineering and Experiments (ALENEX)*, pages 26–40. SIAM, 2005.

[54] F. Gundling, M. H. Keyhani, M. Schnee, and K. Weihe. Fully realistic multi-criteria multi-modal routing. Technical report, TU Darmstadt, 2014.

[55] D.-H. Han, Y.-D. Kim, and J.-Y. Lee. Multiple-criterion shortest path algorithms for global path planning of unmanned combat vehicles. *Computers & Industrial Engineering*, 71(0):57–69, 2014.

[56] L. Han, H. Wang, and W. Mackey Jr. Finding shortest paths under time-bandwidth constraints by using elliptical minimal search area. *Transportation Research Record: Journal of the Transportation Research Board*, No. 1977:225–233, 2006.

[57] P. Hansen. Bicriterion path problems. In G. Fandel and T. Gal, editors, *Multiple Criteria Decision Making Theory and Application*, volume 177 of *Lecture Notes in Economics and Mathematical Systems*, pages 109–127. Springer Berlin Heidelberg, 1980.

[58] P. Hart, N. Nilsson, and B. Raphael. A Formal Basis for the Heuristic Determination of Minimum Cost Paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968.

[59] H. H. Hochmair and J. Fu. Web Based Bicycle Trip Planning for Broward County, Florida. In *ESRI User Conference*, 2009.

[60] M. E. T. Horn. Multi-modal and demand-responsive passenger transport systems: a modelling framework with embedded control systems. *Transportation Research Part A: Policy and Practice*, 36(2):167–188, 2002.

[61] M. E. T. Horn. Procedures for planning multi-leg journeys with fixed-route and demand-responsive passenger transport services. *Transportation Research Part C: Emerging Technologies*, 12(1):33–55, 2004.

[62] J. Hrncir. Improving a Collaborative Travel Planning Application. Master's thesis, The University of Edinburgh, 2011.

[63] J. Hrncir and M. Jakob. Generalised Time-Dependent Graphs for Fully Multimodal Journey Planning. In *16th International IEEE Conference on Intelligent Transportation Systems (ITSC)*, pages 2138–2145. IEEE, 2013.

[64] J. Hrncir and M. Rovatsos. Applying Strategic Multiagent Planning to Real-World Travel Sharing Problems. In *7th Workshop on Agents in Traffic and Transportation (AAMAS)*, 2012.

[65] J. Hrncir, M. Rovatsos, and M. Jakob. Ridesharing on Timetabled Transport Services: A Multiagent Planning Approach. *Journal of Intelligent Transportation Systems*, 19(1):89–105, 2015.

[66] J. Hrncir, Q. Song, P. Zilecky, M. Nemet, and M. Jakob. Bicycle Route Planning with Route Choice Preferences. In *Prestigious Applications of Artificial Intelligence (PAIS)*, pages 1149–1154. IOS Press, 2014.

[67] J. Hrncir, P. Zilecky, Q. Song, and M. Jakob. Speedups for Multi-Criteria Urban Bicycle Routing. In G. F. Italiano and M. Schmidt, editors, *15th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS)*, pages 16–28, 2015.

[68] J. Hrncir, P. Zilecky, Q. Song, and M. Jakob. Practical Multi-Criteria Urban Bicycle Routing. *IEEE Transactions on Intelligent Transportation Systems (accepted)*, 2016.

[69] C. Hsu and B. W. Wah. The SGPlan Planning System in IPC-6. In *International Planning Competition (IPC)*, 2008.

[70] P. L. Jacobsen. Safety in numbers: more walkers and bicyclists, safer walking and bicycling. *Injury Prevention*, 9(3):205–209, 2003.

[71] M. Jakob, J. Hrncir, L. Oliva, F. Ronzano, P. Zilecky, and J. Finnegan. Personalized Fully Multimodal Journey Planner. In *Prestigious Applications of Intelligent Systems (PAIS)*, pages 1225–1226, 2014.

[72] A. Jonsson and M. Rovatsos. Scaling Up Multiagent Planning: A Best-Response Approach. In *International Conference on Automated Planning and Scheduling (ICAPS)*, pages 114–121. AAAI Press, 2011.

[73] D. Kirchler. *Efficient routing on multi-modal transportation networks*. PhD thesis, Ecole Polytechnique, Paris, 2013.

[74] M. Levandowsky and D. Winter. Distance between sets. *Nature*, (5323):34–35, 1971.

[75] J. Linka. Android App for Bicycle Route Planning and Navigation. Bachelor thesis, Czech Technical University in Prague, FEE, 2015.

[76] E. Machuca and L. Mandow. Multiobjective heuristic search in road maps. *Expert Systems with Applications*, 39(7):6435–6445, 2012.

[77] L. Mandow and J. De La Cruz. Multiobjective A* search with consistent heuristics. *Journal of the ACM*, 57(5), 2008.

[78] E. Q. V. Martins. On a multicriteria shortest path problem. *European Journal of Operational Research*, 16(2):236–245, 1984.

[79] D. Monderer and L. S. Shapley. Potential Games. *Games and Economic Behavior*, 14(1):124–143, 1996.

[80] M. Müller-Hannemann and M. Schnee. Finding all attractive train connections by multi-criteria pareto search. In *4th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS)*, pages 246–263, 2004.

[81] M. Müller-Hannemann and K. Weihe. On the cardinality of the pareto set in bicriteria shortest path problems. *Annals of Operations Research*, 147(1):269–286, 2006.

[82] R. Nissim, R. Brafman, and C. Domshlak. A general, fully distributed multi-agent planning algorithm. In *International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, volume 9, pages 1323–1330, 2010.

[83] J. Nykl, J. Hrncir, and M. Jakob. Achieving Full Plan Multimodality by Integrating Multiple Incomplete Journey Planners. In *18th International IEEE Conference on Intelligent Transportation Systems (ITSC)*, pages 1430–1435. IEEE, 2015.

[84] J. Nykl, M. Jakob, and J. Hrncir. Advanced Public Transport Network Analyser. *Prestigious Applications of Intelligent Systems (PAIS)*, pages 1229–1230, 2014.

[85] J. Nykl, M. Jakob, and J. Hrncir. Efficient fine-grained analysis of urban transport accessibility. In *Smart Cities Symposium Prague (SCSP)*, pages 1–5, 2015.

[86] Office for National Statistics. Special Travel Statistics (Level 1). [computer file], ESRC/JISC Census Programme, Census Interaction Data Service, University of Leeds and University of St. Andrews, 2001.

[87] T. Pajor. Multi-Modal Route Planning. Master's thesis, Universitat Karlsruhe, 2009.

[88] C. H. Papadimitriou and M. Yannakakis. On the approximability of trade-offs and optimal access of web sources. In *41st Annual Symposium on Foundations of Computer Science*, pages 86–92, 2000.

[89] P. Perny and O. Spanjaard. Near admissible algorithms for multiobjective search. In *European Conference on Artificial Intelligence (ECAI)*, pages 490–494, Amsterdam, 2008. IOS Press.

[90] E. Pyrga, F. Schulz, D. Wagner, and C. Zaroliagis. Experimental Comparison of Shortest Path Approaches for Timetable Information. In L. Arge, G. F. Italiano, and R. Sedgewick, editors, *Meeting on Algorithm Engineering and Experiments (ALENEX)*, pages 88–99. SIAM, 2004.

[91] E. Pyrga, F. Schulz, D. Wagner, and C. Zaroliagis. Efficient models for timetable information in public transportation systems. *Journal of Experimental Algorithmics (JEA)*, 12, 2008.

[92] K. Rehrl, S. Bruntsch, and H.-J. Mentz. Assisting multimodal travelers: Design and prototypical implementation of a personal travel companion. *IEEE Transactions on Intelligent Transportation Systems*, 8(1):31–42, 2007.

[93] S. Richter and M. Westphal. The LAMA planner. Using landmark counting in heuristic search. In *International Planning Competition (IPC)*, 2008.

[94] P. Sanders and D. Schultes. Highway Hierarchies Hasten Exact Shortest Path Queries. In G. S. Brodal and S. Leonardi, editors, *European Symposium on Algorithms (ESA)*, volume 3669 of *Lecture Notes in Computer Science*, pages 568–579. Springer, 2005.

[95] P. Sanders and D. Schultes. Engineering Highway Hierarchies. In Y. Azar and T. Erlebach, editors, *European Symposium on Algorithms (ESA)*, volume 4168 of *Lecture Notes in Computer Science*, pages 804–816. Springer, 2006.

[96] G. Sauvanet and E. Neron. Search for the best compromise solution on multiobjective shortest path problem. *Electronic Notes in Discrete Mathematics*, 36:615–622, 2010.

[97] P. Scarf. Route choice in mountain navigation, Naismith's rule, and the equivalence of distance and climb. *Journal of Sports Sciences*, 25(6), Apr. 2007.

[98] D. Schultes. *Route Planning in Road Networks*. PhD thesis, Universitat Karlsruhe, 2008.

[99] F. Schulz. *Timetable Information and Shortest Paths*. PhD thesis, Universitat Karlsruhe, 2005.

[100] F. Schulz, D. Wagner, and K. Weihe. Dijkstra's Algorithm On-Line: An Empirical Case Study From Public Railroad Transport. *Journal of Experimental Algorithmics (JEA)*, 5, 2000.

[101] F. Schulz, D. Wagner, and C. Zaroliagis. Using Multi-level Graphs for Timetable Information in Railway Systems. In *Revised Papers from the 4th International Workshop on Algorithm Engineering and Experiments (ALENEX)*, London, UK, 2002.

[102] D. Slunecko. Using GPS Traces for Journey Duration Estimates in Transport Networks. Bachelor thesis, Czech Technical University in Prague, FEE, 2015.

[103] Q. Song, P. Zilecky, M. Jakob, and J. Hrncir. Exploring Pareto Routes in Multi-Criteria Urban Bicycle Routing. In *17th International IEEE Conference on Intelligent Transportation Systems (ITSC)*, pages 1781–1787. IEEE, 2014.

[104] B. S. Stewart and C. C. White. Multiobjective A*. *Journal of the ACM (JACM)*, 38(4):775–814, 1991.

[105] S. Storandt. Route Planning for Bicycles - Exact Constrained Shortest Paths Made Practical via Contraction Hierarchy. In L. McCluskey, B. Williams, J. R. Silva, and B. Bonet, editors, *International Conference on Automated Planning and Scheduling (ICAPS)*. AAAI, 2012.

[106] B. Strasser and D. Wagner. Connection Scan Accelerated. In C. C. McGeoch and U. Meyer, editors, *Meeting on Algorithm Engineering and Experiments (ALENEX)*. SIAM, 2014.

[107] J. G. Su, M. Winters, M. Nunes, and M. Brauer. Designing a route planner to facilitate and promote cycling in Metro Vancouver, Canada. *Transportation Research Part A: Policy and Practice*, 44(7):495–505, 2010.

[108] L. Telesca, J. Finnegan, P. Ferronato, P. Malone, F. Ricci, and K. Stanoevska-Slabeva. Open Negotiation Environment: An Open Source Self-Learning Decentralised Negotiation Framework for Digital Ecosystems. In *Inaugural IEEE International Conference on Digital Ecosystems and Technologies (IEEE-DEST)*, 2007.

[109] A. Torreno, E. Onaindia, and O. Sapena. An approach to multi-agent planning with incomplete information. In *European Conference on Artificial Intelligence (ECAI)*, 2012.

[110] I. Tsamardinos, M. E. Pollack, and J. F. Horty. Merging plans with quantitative temporal constraints, temporally extended actions, and conditional branches. In S. Chien, S. Kambhampati, and C. A. Knoblock, editors, *5th International Conference on Artificial Intelligence Planning Systems (AIPS)*, pages 264–272, 2000.

[111] C. T. Tung and K. L. Chew. A multicriteria pareto-optimal path algorithm. *European Journal of Operational Research*, 62(2):203–209, 1992.

[112] R. J. Turverey, D. D. Cheng, O. N. Blair, J. T. Roth, G. M. Lamp, and R. Cogill. Charlottesville bike route planner. In *Systems and Information Engineering Design Symposium (SIEDS)*, pages 68–72, 2010.

[113] R. van der Krogt, M. D. Weerdt, and Y. Zhang. Of Mechanism Design and Multiagent Planning. In *European Conference on Artificial Intelligence (ECAI)*, pages 423–427. IOS Press, 2008.

[114] M. Winters, G. Davidson, D. Kao, and K. Teschke. Motivators and deterrents of bicycling: comparing influences on decisions to ride. *Transportation*, 38(1):153–168, 2011.

[115] J. Woodcock, P. Edwards, C. Tonne, B. G. Armstrong, O. Ashiru, D. Banister, S. Beevers, Z. Chalabi, Z. Chowdhury, A. Cohen, et al. Public health benefits of strategies to reduce greenhouse-gas emissions: urban land transport. *The Lancet*, 374(9705):1930–1943, 2009.

[116] Y. Wu, L. Guan, and S. Winter. Peer-to-peer shared ride systems. *GeoSensor Networks*, pages 252–270, 2008.

[117] H. Yu and F. Lu. A Multi-Modal Route Planning Approach With an Improved Genetic Algorithm. In E. Guilbert, B. Lees, and Y. Leung, editors, *Joint International Conference on Theory, Data Handling and Modelling in GeoSpatial Information Science*, pages 343–348, 2010.

[118] P. Zilecky. Junction-Aware Multicriteria Bicycle Route Planning. Master's thesis, Czech Technical University in Prague, FEE, 2015.

Appendix A

Publications and Responses

The list of publications is supplemented with the contribution percentage and citations. Citations are based on Google Scholar and Research Gate in May 2016 and exclude self-citations.

Publications Related to the Topic of the Thesis

Journals with IF (2)

- **J. Hrnčíř**, P. Žilecký, Q. Song, and M. Jakob. Practical Multi-Criteria Urban Bicycle Routing. *IEEE Transactions on Intelligent Transportation Systems, (accepted)*, 2016. **(40%)**

- **J. Hrnčíř**, M. Rovatsos, and M. Jakob. Ridesharing on Timetabled Transport Services: A Multiagent Planning Approach. *Journal of Intelligent Transportation Systems*, 19(1):89–105, 2015. **(70%) (2 citations)**

  - A. Baykasoglu and V. Kaplanoglu. An application oriented multi-agent based approach to dynamic load/truck planning. *Expert Systems with Applications*, 42(15–16):6008–6025, 2015.
  - A. Garrett, C. Wrigley, N. Russell, and J. Matthews. A methodological survey of future mobility literature: Opportunities for design research. In *6th International Association of Societies of Design Research Congress*, pages 2578–2595, 2015.

In Proceedings (9)

- **J. Hrnčíř**, Q. Song, P. Žilecký, M. Német, and M. Jakob. Bicycle Route Planning with Route Choice Preferences. In *Prestigious Applications of Artificial Intelligence (PAIS)*, pages 1149–1154, 2014. **(40%)**

- Q. Song, P. Žilecký, M. Jakob, and **J. Hrnčíř**. Exploring Pareto Routes in Multi-Criteria Urban Bicycle Routing. In *IEEE 17th International Conference on Intelligent Transportation Systems (ITSC)*, pages 1781–1787, 2014. **(25%)**

- **J. Hrnčíř**, P. Žilecký, Q. Song, and M. Jakob. Speedups for Multi-Criteria Urban Bicycle Routing. In G. F. Italiano and M. Schmidt, editors, *15th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS)*, pages 16–28, 2015. **(45%)**

- **J. Hrnčíř** and M. Jakob. Generalised Time-Dependent Graphs for Fully Multimodal Journey Planning. In *IEEE Intelligent Transportation Systems Conference (ITSC)*, pages 2138–2145, 2013. **(80%) (6 citations)**

  - S. Wells, P. Forbes, J. Masthoff, S. Gabrielli, and A. Jyllha. Superhub: Integrating digital behaviour management into a novel sustainable urban mobility system. In *27th International BCS Human Computer Interaction Conference (BCS-HCI)*, pages 62:1–62:2, 2013.
  - M. Beutel, S. Gokay, W. Kluth, K.-H. Krempels, C. Samsel, and C. Terwelp. Product oriented integration of heterogeneous mobility services. In *17th International IEEE Conference on Intelligent Transportation Systems (ITSC)*, pages 1529–1534, 2014.
  - M. Beutel, S. Gokay, W. Kluth, K.-H. Krempels, C. Samsel, C. Terwelp, and M. Wiederhold. Heterogeneous Travel Information Exchange. In *2nd EAI International Conference on Mobility in IoT*, 2015.
  - V. Codina, J. Mena, and L. Oliva. Context-aware user modeling strategies for journey plan recommendation. In F. Ricci, K. Bontcheva, O. Conlan, and S. Lawless, editors, *User Modeling, Adaptation and Personalization*, volume 9146 of *Lecture Notes in Computer Science*, pages 68–79, 2015.
  - D. Esztergar-Kiss and C. Csiszar. Evaluation of multimodal journey planners and definition of service levels. *International Journal of Intelligent Transportation Systems Research*, 13(3):154–165, 2015.
  - M. C. Beutel, B. S. Zaunbrecher, S. Himmel, K.-H. Krempels, and M. Ziefle. Evaluation of an integrated intermodal travel service. In *5th International Conference on Smart Cities and Green ICT Systems*, pages 363–371, 2016.

- M. Jakob, **J. Hrnčíř**, L. Oliva, F. Ronzano, P. Žilecký, and J. Finnegan. Personalized Fully Multimodal Journey Planner. In *Prestigious Applications of Intelligent Systems (PAIS)*, pages 1225–1226, 2014. **(30%)**

- J. Nykl, M. Jakob, and **J. Hrnčíř**. Advanced Public Transport Network Analyser. *Prestigious Applications of Intelligent Systems (PAIS)*, pages 1229–1230, 2014. **(20%)**

- J. Nykl, M. Jakob, and **J. Hrnčíř**. Efficient fine-grained analysis of urban transport accessibility. In *Smart Cities Symposium Prague (SCSP)*, pages 1–5, 2015. **(15%)**

- J. Nykl, **J. Hrnčíř**, and M. Jakob Achieving Full Plan Multimodality by Integrating Multiple Incomplete Journey Planners. In *IEEE 18th International Conference on Intelligent Transportation Systems (ITSC)*, pages 1430–1435, 2015. **(25%)**

- **J. Hrnčíř** and M. Rovatsos. Applying Strategic Multiagent Planning to Real-World Travel Sharing Problems. In *7th Workshop on Agents in Traffic and Transportation (AAMAS)*, June 2012. **(85%) (1 citation)**

  - J. Takahashi, R. Kanamori, and T. Ito. A preliminary study on anticipatory stigmergy for traffic management. In *IEEE/WIC/ACM International Conferences on Web Intelligence and Intelligent Agent Technology (WI-IAT)*, volume 3, pages 399–405, 2012.

Appendix B

Pseudocodes of Multi-Criteria Speedups

This appendix contains pseudocode of six methods for the multi-criteria speedups described in Section 4.3 and Section 4.3.1.

Algorithm 5: Extraction of Pareto set of routes

Input: Full Pareto set of labels
Output: Full Pareto set of routes

```
function extractRoutes(set Labels)
begin
    m := Labels.size()
    for p := 1, 2, ..., m do
        π_p := empty set
        L := Labels.get(i)
        while L != null do
            u := L.getNode()
            π_p.insertAtBeginning(u)
            L := L.getPredecessorLabel()
        end
    end
    return Π = {π_1, π_2, ..., π_m}
end
```

Algorithm 6: Default dominance check

```
function MLS.checkDominance(label next)
global Bag
begin
    v := next.getNode()
    c→ := next.getCost()
    foreach label L ∈ Bag(v) do
        l→ := L.getCost()
        if l→ ≺ c→ then return false
        if c→ ≺ l→ then remove(L)
    end
    return true
end
```

Algorithm 7: Termination condition for ratio-based pruning speedup

```
function RatioPruning.terminationCondition(label current)
global d  // destination
global reachedDestination = false
begin
    l→ := current.getCost()
    u := current.getNode()
    upperCost := ∞
    if u == d && !reachedDestination then
        upperCost = α · l_1
        reachedDestination := true
    end
    if l_1 > upperCost then return true
    return false
end
```

Algorithm 8: Check dominance function for cost-based pruning speedup

```
function CostPruning.checkDominance(label next)
global Bag
begin
    v := next.getNode()
    c→ := next.getCost()
    foreach label L ∈ Bag(v) do
        l→ := L.getCost()
        if Σ_{i=1}^{k} (c_i − l_i)^2 ≤ γ^2 ∥ l→ ≺ c→ then return false
        if c→ ≺ l→ then remove(L)
    end
    return true
end
```

Algorithm 9: Check dominance function for ϵ-dominance speedup

```
function EpsilonDominance.checkDominance(label next)
global Bag
begin
    v := next.getNode()
    c→ := next.getCost()
    foreach label L ∈ Bag(v) do
        l→ := L.getCost()
        if l→ ≺ (1 + ε)c→ then return false
    end
    foreach label L ∈ Bag(v) do
        l→ := L.getCost()
        if c→ ≺ (1 + ε)l→ then remove(L)
    end
    return true
end
```

Algorithm 10: Check dominance function for buckets speedup

```
function BUCKETS.CHECKDOMINANCE(label next)
global Bag
begin
    c⃗ := next.getCost()
    foreach label L ∈ Bag do
        l⃗ := L.getCost()
        if bucketValue(l⃗) ≺ bucketValue(c⃗) then return false
        if bucketValue(c⃗) ≺ bucketValue(l⃗) then remove(L)
    end
    return true
end
```

Appendix C

Feature Values Overview

Table C.1: Travel time, slowdown, surface, and traffic feature values overview. The unit of r'_{slowdown} is seconds, other feature values are without a unit.

category:entity:key:value	r'_{time}	r'_{slowdown}	r'_{surface}	r'_{traffic}
surface:way:surface:cobblestone	0.7		5	
surface:way:surface:compacted	0.9		1.5	
surface:way:surface:gravel	0.5		5	
surface:way:surface:ground	0.6		4	
surface:way:surface:paving_stones	0.75		1.5	
surface:way:surface:setts	0.8		2	
surface:way:surface:unpaved	0.75		4	
obstacles:node:highway:elevator		38		
obstacles:node:highway:steps		8		
obstacles:node:traffic_calming:bump		2		
crossing:node:highway:traffic_signals		15		
crossing:node:highway:stop		8		
crossing:node:crossing:uncontrolled		8		
crossing:node:highway:crossing		8		
for_bicycles:way:highway:cycleway				0.2
for_bicycles:way:cycleway:lane				0.6
for_bicycles:way:cycleway:shared_lane				0.8
motor_roads:way:highway:living_street				0.5
motor_roads:way:highway:tertiary				2
motor_roads:way:highway:secondary				6
motor_roads:way:highway:primary				10

A complete list of feature values is available in the repository[1].

[1] https://github.com/agents4its/cycleplanner/blob/mcspeedups/cycle-planner/src/main/resources/feature_values.csv