eCO-friendly urban Multimodal route PlAnning Services for mobile uSers

FP7 - Information and Communication Technologies

Grant Agreement No: 288094
Collaborative Project
Project start: 1 November 2011, Duration: 38 months

D3.6 – Final Assessment of Multimodal Route Planning Algorithms

Workpackage:	WP 3 - Algorithms for Multimodal Human Mobility
Due date of deliverable:	30 November 2014
Actual submission date:	30 November 2014
Responsible Partner:	TomTom
Contributing Partners:	CTI, ETHZ, KIT, TomTom

Nature: ☒ Report ☐ Prototype ☐ Demonstrator ☐ Other

Dissemination Level:
☒ PU: Public
☐ PP: Restricted to other programme participants (including the Commission Services)
☐ RE: Restricted to a group specified by the consortium (including the Commission Services)
☐ CO: Confidential, only for members of the consortium (including the Commission Services)

Keyword List: Tourist Trip Design Problem, Route planning, Orienteering Problem, Time Window, Time Dependent Team Orienteering, Time dependent travel time, Multimodal, Public transport.

The eCOMPASS project (www.ecompass-project.eu) is funded by the European Commission, DG CONNECT (Communications Networks, Content and Technology Directorate General), Unit H5 - Smart Cities & Sustainability, under the FP7 Programme.

The eCOMPASS Consortium

Computer Technology Institute & Press 'Diophantus' (CTI) (coordinator), Greece

Centre for Research and Technology Hellas (CERTH), Greece

Eidgenössische Technische Hochschule Zürich (ETHZ), Switzerland

Karlsruhe Institute of Technology (KIT), Germany

TOMTOM INTERNATIONAL BV (TOMTOM), Netherlands

PTV PLANUNG TRANSPORT VERKEHR AG. (PTV), Germany

Document history			
Version	Date	Status	Modifications made by
0.1	08.10.2014	Table of Contents draft	Damianos Gavalas, CTI
0.2	31.10.2014	First Draft	Julian Dibbelt, KIT
0.3	05.11.2014	Second Draft	Tobias Pröger, ETHZ
0.4	05.11.2014	Third Draft	Damianos Gavalas, CTI
0.5	08.11.2014	Combined Draft	Julian Dibbelt, KIT
0.6	11.11.2014	Fifth Draft	Julian Dibbelt, KIT
1.0	13.11.2014	Sent to internal reviewers	Felix König, TomTom
1.1	14.11.2014	Updated Section 3	Tobias Pröger, ETHZ
1.2	17.11.2014	Reviewers' comments incorporated (sent to PQB)	Felix König, TomTom
1.3	21.11.2014	PQB's comments received	Felix König, TomTom
1.4	30.11.2014	Final (sent to the Project Officer)	Christos Zaroliagis, CTI

Deliverable manager

- Felix König, TomTom

List of Contributors

- Kateřina Böhmová, ETHZ
- Julian Dibbelt, KIT
- Damianos Gavalas, CTI
- Andreas Gemsa, KIT
- Felix König, TomTom
- Vlasios Kasapakis, CTI
- Matúš Mihalák, ETHZ
- Grammati Pantziou, CTI
- Tobias Pröger, ETHZ
- Ben Strasser, KIT
- Nikolaos Vathis, CTI

List of Evaluators

- Dionisis Kehagias, CERTH
- Florian Krietsch, PTV

Summary
In this deliverable we report on our final assessment of models and algorithmic solutions to multimodal route planning developed in WP 3. After reviewing the eCOMPASS results obtained by month 20 in Tasks 3.3, 3.4, and 3.5, respectively, we discuss modeling extensions, new algorithmic approaches, and rigorously experimentally evaluate them on real-world transportation networks. Based on this evaluation, we identify the most applicable and technically most robust solutions, which are integrated by project partners in WP 5 and extensively evaluated during the pilot in Berlin in WP 6.

Contents

1 Introduction

The aim of this deliverable is to document the results of Task 3.6. It describes progress made in Work Package 3 with respect to the algorithmic solutions of month 20 (c. f. Deliverable D3.3.x, D3.4.x, D3.5.x). The main goal of this deliverable is to assess the success of the algorithmic solutions developed within WP 3 upon real public transportation network and to identify the technically most robust solutions. This deliverable concludes WP 3.

Background. The aim of WP 3 is to provide novel methods for environmentally friendly routes in urban public transportation networks. In particular, the goal is to develop mathematically sound models for various (context-aware) route planning scenarios arising in the field of urban human mobility for city residents, commuters and tourists, as well as to provide algorithmic methods for multimodal routes in urban transportation networks with respect to multiple criteria and high robustness with a strong focus on the environmental footprint of these routes. Algorithms and methods developed within this work package will be implemented and an extensive experimental evaluation regarding performance and quality will be conducted following the Algorithm Engineering paradigm [58, 59]. This paradigm differs from traditional Algorithmic Design in Theoretical Computer Science in several key factors: instead of deriving asymptotic bounds for a given algorithm's performance with respect to worst-case inputs on abstract machine models, typical and realistic instances are examined on real machines to measure the practical performance of the implementation of a given algorithm. The results of this experimental evaluation are then analyzed and used to guide the design of the algorithm, the improvement of which is then again experimentally verified in a continuing feedback loop.

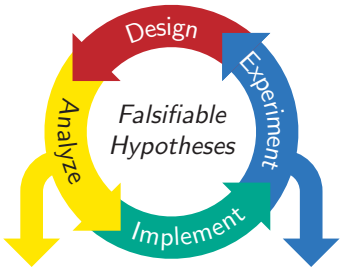

Figure 1: Algorithm Engineering cycle.

Objectives and Scope of Deliverable D3.6. The goal of WP 3 is the development of new models and solutions to route planning in multimodal urban transportation networks. Within Task 3.6 specifically, we have deepened our understanding of the algorithmic nature of route planning for multimodal urban transportation networks, rigorously tested our approaches on real world transportation network data, and identified the most promising solutions. Results were integrated by partners in WP 5 and tested in the pilot within the scope of WP 6. The research done so far in WP 3 has led to several peer-reviewed scientific publications as well as several pending publications (c. f. Technical Reports ECOMPASS-TR-003, TR-005, TR-006, TR-010, TR-011, TR-013, TR-015, TR-019, TR-020, TR-021, TR-022, TR-023, TR-026, TR-030, TR-031, TR-032, TR-033, TR-034, TR-047, TR-049, TR-055, TR-056, TR-057, TR-058, TR-060, TR-062, TR-063).

Outline. In Section 2, we discuss the progress made for robust multimodal route planning since Deliverables D3.3.1 and D3.3.2. We describe extensions, model refinements for eco-friendly routing, and experimental results. Section 3 reports on the final assessment for multimodal route planning algorithms using methods from stochasticity and machine learning and the progress made since Deliverables D3.4.1 and D3.4.2, and Section 4 shows the new achievements on context-aware multimodal daily routes for tourists made since Deliverables D3.5.1 and D3.5.2. In Section 5, we discuss and identify the most applicable and robust solutions to be integrated and prepared for piloting. We also highlight changes applied as a reaction to YR2 recommendations.

2 Final assessment of robust multimodal route planning algorithms

2.1 Brief overview of D3.3 algorithmic approaches

The main goal of Deliverable D3.3 ("New eco-aware models and solutions to robust multimodal route-planning and their empirical assessment") has been the design of new models and algorithmic solutions to multimodal route-planning. We identified the core algorithmic challenges that arise for fully multimodal urban networks as follows: Modeling issues and better solutions to subproblems, preprocessing flexible enough for user-defined path constraints, capturing the richness of "best" solutions in fully multimodal networks. In Deliverable D3.3, we proposed several interesting new approaches to these challenges, which we summarize below.

Connection Scan. The problem of computing "best" journeys in public transportation networks comes in several variants [53]: The simplest, called *earliest arrival*, takes a departure time as input, and determines a journey that arrives at the destination as early as possible. If further criteria, such as the number of transfers, are important, one may consider *multi-criteria* optimization [22, 29]. Finally, a *profile query* [20, 22] computes a set of optimal journeys that depart during a period of time (such as a day). Traditionally, these problems have been solved by (variants of) Dijkstra's algorithm on an appropriate graph model. Well-known examples are the time-expanded and time-dependent models [20, 34, 53, 57]. Recently, Delling et al. [22] introduced RAPTOR. It solves the multi-criteria problem (arrival time and number of transfers) by using dynamic programming directly on the timetable, hence, no longer requires a graph or a priority queue.

In Deliverable D3.3, we presented the *Connection Scan Algorithm* (CSA). In its basic variant, it solves the earliest arrival problem, and is, like RAPTOR [22], not graph-based (c f. [20, 34, 53, 57]). However, it is not centered around *routes* (as RAPTOR), but elementary *connections*, which are the most basic building block of a timetable. CSA organizes them as one single array, which it then scans once (linearly) to compute journeys to all stops of the network. The algorithm turns out to be intriguingly simple with excellent spatial data locality. CSA is easily extended to handle multi-criteria profile queries: For a full time period, it computes Pareto sets of journeys optimizing arrival time and number of transfers, very efficiently.

Moreover, CSA does not make use of heavy preprocessing, thus, enabling dynamic scenarios including train cancellations, route changes, real-time delays, etc. Our experiments on the dense metropolitan network of London validated the approach. With CSA, we computed earliest arrival queries in under 2 ms, and multi-criteria profile queries for a full period in 150 ms—faster than previous algorithms. For details please refer to ECOMPASS-TR-021.

User-Constrained Multimodal Route Planning. In Deliverable D3.3, we presented UCCH (User Constrained Contraction Hierarchies), the first multimodal speedup technique that handles arbitrary mode-sequence constraints as input to the query—a feature unavailable from previous techniques. Unlike Access-Node Routing [21], it also answers local queries correctly and requires significantly less preprocessing effort. We revisited one technique, namely *node contraction*, that has proven successful in road networks in the form of Contraction Hierarchies (CH), introduced by Geisberger et al. [36]. We showed how CH can be used to compute shortest paths with restrictions on sequences of transport modes. However, applying CH on the combined multimodal graph without careful consideration either yields incorrect results to the *Label Constrained Shortest Path Problem with Mode Sequence constraints* (LCSPP-MS) or predetermines the constraints automaton during preprocessing. We therefore introduced UCCH, a practical adaption of Contraction Hierarchies to LCSPP-MS that enables arbitrary modal sequence constraints as query input. By ensuring that shortcuts never span multiple modes of transport, we extended Contraction Hierarchies in a sound manner. Moreover, we showed how careful engineering further helps our scenario. As a

result, UCCH is the first, fast multimodal speedup technique that handles arbitrary modal sequence constraints at query time—a problem considered challenging before. Besides not determining the modal constraints during preprocessing, its advantages are small space overhead, fast preprocessing time and the ability to implicitly handle local queries without the need for a separate algorithm. Its preprocessing can handle huge networks of intercontinental size with many more stations and airports than those of previous multimodal techniques. Our experimental study showed that, unlike previous techniques, we can handle an intercontinental instance composed of cars, railways and flights with over 50 million nodes, 125 million edges, and 30 thousand stations. With only 557 MiB of auxiliary data, we achieved query times that are fast enough for interactive scenarios. For details please refer to ECOMPASS-TR-006.

Multicriteria Multimodal. Online services for journey planning have become a commodity used daily by millions of commuters. The problem of efficiently computing good journeys in transportation networks presents several algorithmic challenges, and has been an active area of research in recent years. Much focus has been given to the computation of routes both in road networks [1, 19, 23, 37, 42, 61] and in scheduled-based public transit [5, 6, 8, 18, 22, 29, 52, 54, 60], but these are often considered separately. In practice, however, users want an integrated solution that can find the "best" way to get to their destination considering all available modes of transportation, e. g., within a metropolitan area including buses, trains, driving, cycling, taxis, an walking. We refer to this as the *multimodal route planning* problem.

In fact, any public transportation network necessarily has a multimodal component, since journeys require some amount of walking. Existing solutions [6, 8, 20, 22, 29] handle this by predefining transfer arcs between nearby stations, and running a search algorithm on the public transit network to find the "best" journey. Unlike in road networks, however, defining "best" is not straightforward. For example, while some people want to arrive as early as possible, others are willing to spend a little more time to avoid extra transfers. Most recent approaches therefore compute the *Pareto set* [41] of non-dominating journeys optimizing multiple criteria, which is practical even for large metropolitan areas [22, 54].

Extending public transportation solutions to a full multimodal scenario (with unrestricted walking, biking, and taxis) may seem trivial at first: One could just incorporate routing techniques for road networks [19, 37, 42] to solve the new subproblems. Unfortunately, meaningful multimodal optimization needs to take more criteria into account, such as walking duration and costs. Some people are happy to walk 10 minutes to avoid an extra transfer, while others are not. In fact, some will walk half an hour to avoid using public transportation at all. Taking a taxi all the way to the airport is a good solution for some; users on a budget may prefer a cheaper solution. Not only do these additional criteria significantly increase the Pareto set [24, 35], but some of the resulting journeys tend to look unreasonable, as Figure 2 illustrates.

Given the limitations of current approaches, in Deliverable D3.3 we revisited the problem of finding multicriteria multimodal journeys on a metropolitan scale. Instead of optimizing each mode of transportation independently [30], we argued that most users optimize multiple criteria, e. g., travel time, convenience, and costs. While this produces a large Pareto set, we proposed using fuzzy logic [31, 68] to filter it in a principled way to a modest-sized set of representative journeys. This postprocessing step is not only quick, but can also be user-dependent, incorporating personal preferences. Building on the algorithmic developments of [22, 27, 37] allowed us to answer exact queries optimizing time and convenience in less than two seconds within a large metropolitan area, for the simpler scenario of walking, cycling, and public transit. To accelerate the queries further, we also proposed heuristics (still multicriteria) that are significantly faster, and closely match the top journeys in the Pareto set. By thorough experimental evaluation of all algorithms in terms of both solution quality and performance, we showed that our approach yields high quality results while being fast enough for interactive applications. Moreover, since it does not rely on heavy preprocessing, it can be used in fully dynamic scenarios. For details please refer to ECOMPASS-TR-022.

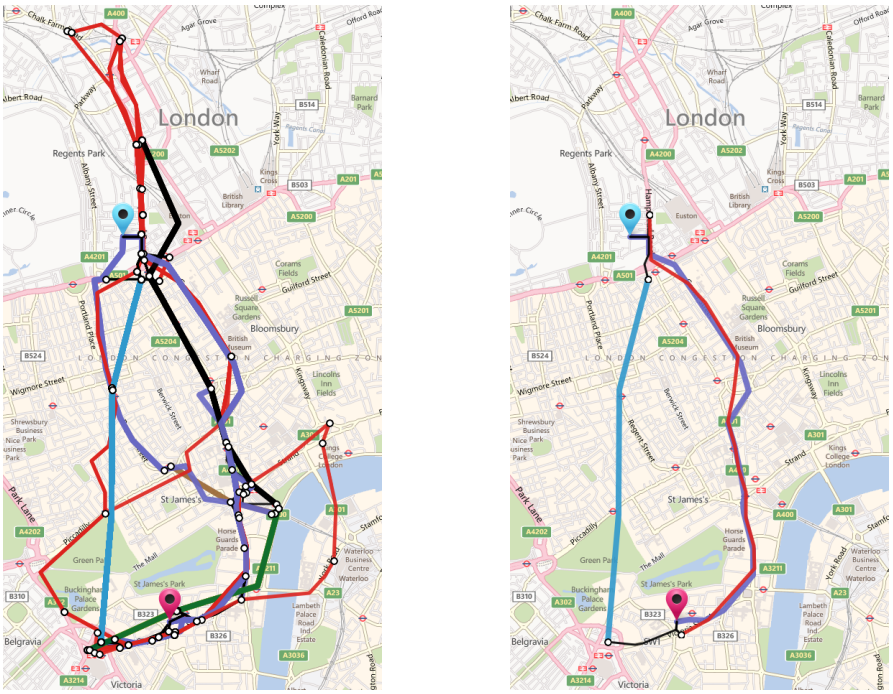

Figure 2: Exemplary multicriteria multimodal query on London with criteria arrival time, number of transfers, walking duration, and cost. The left figure shows the full Pareto set (65 journeys), while the right figure shows the three journeys with highest score. Each dot represents a transfer and included transportation modes are walking (thin black), taxi (thick purple), buses (thin red), and tube (other thick colors).

Preprocessing many-to-many multimodal time-dependent travel times for the Tourist Trip Design. In their most-inner loop, algorithms for tourist trip design optimization (c. f. Section 4) require the pairwise travel time distance between points of interest (POIs). Typically, in the literature such distances are assumed to be already available in a two-dimensional matrix quadratic in the number of POIs. In our scenario however, we consider multimodal travel times between POIs also to be dependent on the time of day. In WP 3, we developed algorithms to precompute such a time-dependent distance matrix between POIs. This preprocessing step ensures fast travel time lookup during tourist trip optimization. In Deliverable D3.5.2, we evaluated the performance of our approach for POIs chosen in the multimodal network of the greater Athens area. This public transit network has 7 778 stops, 570 routes, 26 192 trips, 1 003 188 daily departure events; in the time-dependent route model graph [57] this results in 29 055 vertices and 63 424 arcs. The walking network consists of 287 003 vertices and 685 850 arcs. Points of interests were obtained as described in Deliverable D3.2, however, the size of that data set has grown to 557 POIs total. As of month 20, evaluated on a single core of a 4x 12-core AMD Opteron-6172 machine clocked at 2.1 GHz, preprocessing this network took 162 seconds, after which 40 283 vertices remained in the core network. We ran 557 multimodal one-to-all profile (24h range) queries on this core, and acquired a multimodal distance matrix with 71 145 759 entries total. Computing these distances took approximately 105 minutes. Each of the 557 $\times$ 557 POI combinations had on average about 229 distinct walking/public transportation journeys throughout the day.

2.2 Extensions and New Solutions

In this deliverable, we build upon the algorithmic approaches summarized above. The main efforts in our research in the context of Deliverable D3.6 are: model extensions to further eCOMPASS objectives as well as parallelization and network decomposition to speedup algorithmic performance of our approaches. We show the practicability of our new approaches by means of detailed experimental evaluations. More specifically, in Section 2.2.1 we show how to accelerate our Connection Scan Algorithm, in Section 2.2.2 we discuss more detailed experiments on UCCH, in Section 2.2.3 report our final assessment on the transportation network of Berlin, in particular, we discuss model extensions developed for eCOMPASS for our multimodal multi-criteria route planning approach and report updated statistics on preprocessing travel time profiles for use in the TTDP problem.

2.2.1 Network decomposition and parallelization for faster public transit routing

Here, our approach uses the Connection Scan Algorithm (CSA) [26] as its core. As described above in Section 2.1, CSA is a simple algorithm that does not use a priority queue and does not rely on computing auxiliary data in a preprocessing phase. Its strength lies in its very low running time constants allowing it to solve even moderately sized timetable networks such as the London urban region in about 2 ms for earliest arrival queries, 150 ms for full day profile queries. However, sublinear running times are necessary on larger country-scale timetable networks such as the one used by `bahn.de`. We therefore introduce an additional preprocessing phase. We adapt the basic ideas of CRP [19, 25] and its multi-level overlay predecessors [60, 43] to timetables. Previous studies [7] have shown that directly applying speed-up techniques designed for road graphs to timetables does not work. It is crucial to adapt the algorithms to the special timetable structure. CSA's ability to handle moderately sized timetable networks efficiently makes our approach practical. Recent studies [22, 26] have shown that often algorithms not based on Dijkstra's [28] work better on timetable networks. We describe the first preprocessing based speedup technique that at its core is not based on Dijkstra's algorithm.

Accelerating Connection Scan. We utilize the core idea from CRP [19, 25]. It consists of subdividing a road graph into cells and computing for each cell a replacement graph that preserves shortest paths. Translated into the timetable setting, we recursively partition the stop set into k

Table 1: Instance size

#stops	252 374
#connections	46 218 148
#trips	2 395 656
#footpaths	103 535
#FIFO-routes	229 666
#TE-nodes	82 017 803
#TE-arcs	202 073 458

Table 2: Parallelized running time needed to compute overlays.

SIMD	transfer	running time [s]
○	○	45.4
●	○	49.2
○	●	2007.8
●	●	1794.7

cells over ℓ levels. The top cell is the full stop set. We define a connection to be interior (exterior) to a cell z if it (does not) departs in z. The base algorithm is correct if it processes all connections adjacent to a transfer in the sequence of an optimal journey. We therefore only preserve such connections. For every cell z we compute a set $T(z)$ of interior connections with the property: For every optimal journey j departing and arriving outside of z a journey j' with the same source and target stop, source time and quality exists such that all connections adjacent to a transfer in j's sequence are in $T(z)$ or exterior to z. These connections are the *transit connections* of z. In this section we use a $T(z)$-computing black box. We define the *long distance connection set* $D(z)$ of a cell z as the set of all transit connections of all direct children of z, i. e., $D(z) = \bigcup T(z_{\text{child}})$. (On the lowest level all interior connections are in $D(z)$.) At query time we merge all $D(z)$ of cells z containing the source or target stop. This union is passed to base Connection Scan algorithm. The main idea for acceleration is that this union does not contain transit connections of places far away from the source or target stops. The stops are partitioned using a graph partitioner [47]. Two stops are mapped onto the same node if there is a transfer between them. This prohibits transfers crossing borders. An edge exists if a corresponding connection exists and is weighted by the number of such connections.

We ran experiments on a dual 8-core Intel Xeon E5-2670 processor clocked at 2.6 GHz, with 64 GiB of DDR3-1600 RAM, 20 MiB of L3 and 256 KiB of L2 cache. We use g++ 4.7.1 with `-O3`. An enabled feature is marked by a "●" and a disabled one by a "○". SIMD is done using SSE with registers containing 4 integers of 32 bits. We ran 10^4 queries with random source and target stops and a random source time within the first 24 hours.

Instance. Our test instance is based on the data of `bahn.de` during winter 2011/2012. The data contains European long distance trains, German local trains, many buses inside of Germany and even more exotic vehicles such as rack railways in the Alps. To obtain an instance comparable in size with [6] we extracted all trips regardless of their day of operation and consider two successive identical days. We removed data noise such as exactly duplicated trips, vehicles driving at more than 300 km/h or footpaths at more than 50 km/h. More than half of the connections are buses. Minimum change times are modeled as transfer loops and footpaths as interstop transfer arcs. We transitively closed the transfers and obtain the sizes reported in Table 1. For comparison with [22] we indicate the number of FIFO-routes and for comparison with [6] the time-expanded (TE) graph size.

Table 3: Number of long distance and loop connections averaged per all cells on the same level. The top level is 5.

	only time $T_a(z)$		with transfers $T_t(z)$	
level	$\lvert D(z)\rvert$	$\lvert L(z)\rvert$	$\lvert D(z)\rvert$	$\lvert L(z)\rvert$
0	193 029	11 232	193 029	11 109
1	70 781	15 204	115 660	14 042
2	93 933	20 046	138 036	18 836
3	125 642	24 046	181 689	24 065
4	154 790	25 155	222 143	24 596
5	168 080	0	277 454	0

Table 4: Running times for the accelerated algorithm split up into the merge and scan phases and the number of processed connections.

range	accel.	merge [ms]	scan [ms]	processed conn /10^6
○	○	—	4 173	812
○	●	12	159	1.2
●	●	6	72	0.6

Computing Overlays. In Table 2 we report the times needed to compute the transit sets. To compute the $T_a(z)$ we set all transfer loops to zero in this experiment. "transfer" indicates whether $T_t(z)$ or $T_a(z)$ is computed. All 16 cores are used. We recursively subdivide the stops 5 times into 3 partitions. $T_a(z)$ can be computed in under a minute enabling real time updates. The alternative $T_t(z)$ can be computed in half an hour, which should be fast enough for most applications. SIMD-speedup is smaller than in the base query case because following journey pointers is sequential. Further interleaving pointers from different queries destroys cache locality which explains the slight slowdown for $T_a(z)$. We use $T_t(z)$ in all query experiments.

Table 3 shows the sizes of the overlays corresponding to Table 2. Figure 3 shows how the processing time is distributed among the various cells for the transfer transit connections $T_t(z)$. It shows that computing the transit connections for metropolitan areas on the lowest level is very expensive. The top cities appear decreasing by population count which is reasonable. This huge variance is why we decided to parallelize inside of the blackbox instead of running several of them in parallel to avoid unequal load balances. When computing $T_a(z)$ the same Berlin cell needs the maximum amount of time over all cells but it only needs 0.54s. Further the maximum amount of interior border stops is 313 (but for a different cell).

Earliest Arrival Time and Local Queries. We examine in Figure 4 the speedup of local queries. We use a geo-rank instead of the common Dijkstra-rank used for road networks. With the later it is unclear what source time to use. Further, especially in rural areas, travel times between neighboring villages can be huge. A Dijkstra-rank does not recognize these queries as being local. A geo-rank picks a random source stop and orders all other stops by geographical distance. It runs queries towards the 2^r-th stop. The *geo-rank* of the query is r. Local queries are accelerated on average by a factor of up to 6. The figure also shows that the running times have a lot of variance. The largest outliers are queries where no journey exists. With random source and target stops our query averages at 8.66ms which corresponds to a maximum geo-rank.

Accelerated Profile Queries. We evaluate in Table 4 the speedup of our technique. We assume that a EA time query ($\approx$ 8.66ms) was run to compute a minimum travel time τ while the user selects the time interval. We therefore only consider queries where a journey exists. Journey pointers were

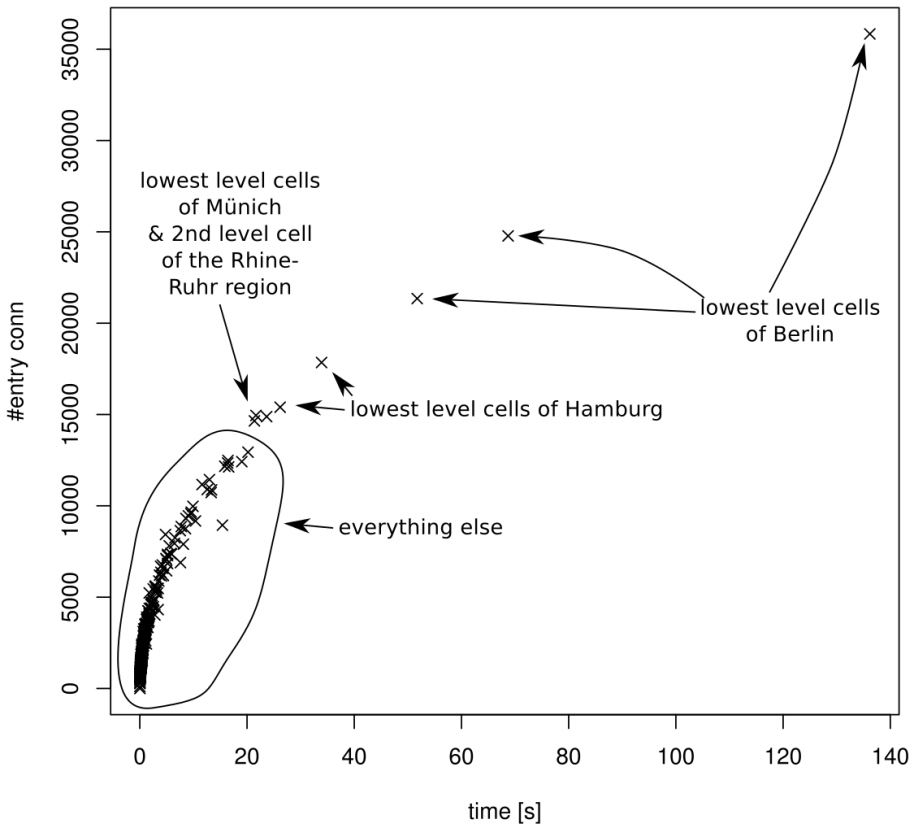

Figure 3: The number of entry connections against the time needed to compute the transfer transit connections. The data points are annotated with the geographical regions that they belong to.

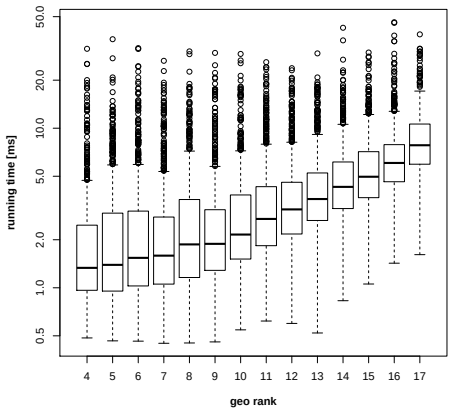

Figure 4: Geo-rank showing the running times of the tailored EA query

Table 5: Comparing size figures of our input instances. The column "Col." indicates whether we use the coloring approach to model the railway subnetwork. The bottom two instances are taken from [21].

	Public Transportation			Road		
Network	Stations	Connections	Col.	Nodes	Edges	Density
`ny-road-rail`	16 897	2 054 896	•	579 849	1 527 594	1 : 56
`de-road-rail`	6 822	489 801	•	5 055 680	12 378 224	1 : 749
`europe-road-rail`	30 517	1 621 111	•	30 202 516	72 586 158	1 : 1 133
`wo-road-rail-flight`	31 689	1 649 371	•	50 139 663	124 625 598	1 : 1 846
`de-road-rail(long)`	498	16 450	○	5 055 680	12 378 224	1 : 10 711
`wo-road-flight`	1 172	28 260	○	50 139 663	124 625 598	1 : 139 277

computed. "accel" indicates whether transit sets were used. "range" indicates whether a source and target time restricted the scan. The source time τ_s is chosen at random and the target time τ_t is set to $\tau_s + 2\tau$. Using transit connections achieves a speedup of 24. This is less than the ratio of processed connections because the relevant connection data is no longer adjacent in memory. Restricting the time interval yields another factor 2 totaling in a speedup of 49 over a CSA profile baseline. Interestingly the number of processed connections is still large, which explains why the scan phase dominates.

Conclusion. Employing network decomposition techniques and applying multi-core and SIMD parallelization, we achieve running times of 8.7 ms for the earliest arrival time problem and of 78 ms for the profile problem on a *large-scale national* timetable networks with secondary transfer optimization. Depending on the formalization our preprocessing is either very fast (1 min) or fast (30 min). For details please refer to ECOMPASS-TR-034. However, detailed evaluation also shows that urban subnetworks are hard to decompose and remain at the core of problem. Hence, for urban route planning purposes, our query acceleration technique is currently less applicable. We will further investigate this in future research outside the scope of eCOMPASS.

2.2.2 Additional experimental details for user-constrained multimodal route planning

For our User-Constrained Contraction Hierarchies technique, we conducted further experiments, especially to demonstrate why *Label Constrained Shortest Paths* are a required ingredient for multimodal route planning. As before, we conducted our experiments on an Intel Xeon E5430 processor running SUSE Linux 11.1. It is clocked at 2.66 GHz, has 32 GiB of RAM and 12 MiB of L2 cache. The program was compiled with GCC 4.5, using optimization level 3. Our implementation is written in C++ using the STL and Boost. We use our own custom implementations for most data structures. In particular, we represent graphs as adjacency arrays, and as a priority queue we use a 4-ary heap. All runs are sequential for comparison.

Inputs We assemble a total of six multimodal networks where two are imported from [21]. Their size figures are reported in Table 5. For `ny-road-rail`, we combine New York's foot network with the public transit network operated by MTA [51]. We link bus and subway stops to road intersections that are no more than 500 m apart. The `de-road-rail` network combines the pedestrian and railway networks of Germany. The railway network is extracted from the timetable of the winter period 2000/01. It includes short and long distance trains, and we link stations using a radius of 500 m. The `europe-road-rail` network combines the road (as in car) and railway networks of Western Europe. The railway network is extracted from the timetable of the winter period 1996/97 and

Table 6: Query performance of UCCH compared to plain multimodal Dijkstra and Access-Node Routing. Figures for the latter are taken from [21], which were obtained on a different machine. We thus scale the running time with respect to Dijkstra.

		Dijkstra		ANR			UCCH		
Network	Automaton	Settled Nodes	Time [ms]	Settled Nodes	Time [ms]	Speed-Up	Settled Nodes	Time [ms]	Speed-Up
ny-road-rail	foot-and-rail	404 816	226	—	—	—	25 525	13.61	17
de-road-rail	foot-and-rail	2 611 054	2 005	—	—	—	18 275	12.78	157
europe-road-rail	car-and-rail	30 021 567	23 993	—	—	—	90 579	53.78	446
wo-road-rail-flight	car-and-flight	36 053 717	33 692	—	—	—	42 056	26.72	1 260
wo-road-rail-flight	hierarchical	36 124 105	35 261	—	—	—	126 072	70.52	500
wo-road-rail-flight	everything	25 267 202	23 972	—	—	—	71 389	50.77	472
de-road-rail(long)	foot-and-rail	2 735 426	2 075	13 524	3.45	602	12 509	3.13	663
wo-road-flight	car-and-flight	36 582 904	33 862	4 200	1.07	31 551	1 647	0.67	50 540

stations are linked within 5 km. The wo-road-rail-flight network is a combination of the road networks of North America and Western Europe with the railway network of Western Europe and the flight network of Star Alliance and One World. The flight networks are extracted from the winter timetable 2008. As radius we use 5 km.

Both de-road-rail(long) and wo-road-flight are from [21]. The data of the Western European and North American road networks (thus Germany and New York) was kindly provided to us by PTV AG [56] for scientific use. The timetable data of New York is publicly available through General Transit Feeds [38], while the data of the German and European railway networks was kindly provided by HaCon [40]. Unlike the data from HaCon, the New York timetable did not contain any foot path data for short transfers between nearby stops (as typically defined by the operator). Thus, we generated artificial foot paths with a known heuristic [20].

Our instances vary in the fractional size of their public transit subnetwork with respect to the total network size. We call the fraction of linked nodes in a subgraph *density* (see last column of Table 5). Our densest network is ny-road-rail, which also has the highest number of connections. On the other hand, de-road-rail(long) and wo-road-flight are rather sparse. However, we include them to compare our algorithm to Access Node Routing (ANR). Note that we take the figures for ANR from [21]. Since they used a different machine, we scale the running time figures by comparing the running time of Dijkstra's algorithm on our machine to theirs

Detailed query time analysis In this experiment we evaluate the query performance of UCCH and compare it to Dijkstra and ANR (where figures are available). Results are presented in Table 6. We observe that we achieve speedups of several orders of magnitude over Dijkstra, depending on the instance. Generally, UCCH's speedup over Dijkstra correlates with the ratio of core nodes after preprocessing (thus, indirectly with the density of transfer nodes): the sparser our networks are interconnected, the higher the speedups we achieve. On our densest network, ny-road-rail, we have a speedup of 17, while on wo-road-flight we achieve query times of less than a millisecond—a speedup of over 50,540. To further highlight how the density of the network affects the speedup, Figure 5 plots the speedup of UCCH on each instance subject to its density. Comparing UCCH to ANR, we observe that query times are in the same order of magnitude, UCCH being slightly faster. Note that we achieve these running times with significantly less preprocessing effort.

Detailed path properties Table 7 reports the impact of integrating modal sequence constraints on the paths output by the algorithm. It does so by evaluating three main figures: The percentage of the total number of paths that utilize a certain transportation mode (foot, car, rail with transfers,

Table 7: Evaluating the impact of integrating modal sequence constraints on the paths.

		Mode Used in % Paths				# Modal Changes		Stretch		
Network	Automaton	Foot	Car	Rail	Flight	Avg.	Max.	Avg.	Max.	Ident. [%]
`ny-road-rail`	`everything`	100	—	57	—	4.1	22	—	—	—
`ny-road-rail`	`foot-and-rail`	100	—	57	—	1.1	2	1.07	2.83	64
`de-road-rail`	`everything`	100	—	100	—	6.8	24	—	—	—
`de-road-rail`	`foot-and-rail`	100	—	100	—	2.0	2	1.08	2.94	87
`europe-road-rail`	`everything`	—	100	41	—	1.2	10	—	—	—
`europe-road-rail`	`car-and-rail`	—	100	41	—	0.8	2	1.03	1.46	92
`wo-road-rail-flight`	`everything`	—	100	13	85	2.2	12	—	—	—
`wo-road-rail-flight`	`hierarchical`	—	100	9	85	1.8	4	1.08	2.25	89
`wo-road-rail-flight`	`car-and-flight`	—	100	—	85	1.7	2	1.06	2.34	84

and flight with transfers), the average and maximum number of interchanges between transportation modes along the journeys, and the average and maximum factor by which the travel time increases when mode sequence constraints are enabled. Note that for the latter, we only count paths that actually differ from the unconstrained one, additionally reporting the amount of paths where mode sequence constraints have no impact (Ident.). Each instance in Table 7 is evaluated on both an appropriate constrained automaton as well as the `everything` automaton, which essentially corresponds to running unrestricted queries.

We observe that on `ny-road-rail` 57 % of the paths utilize the rail network, regardless whether we constrain paths by the `foot-and-rail` automaton. However, 36 % of the paths are indeed different, and enabling constraints reduces the average number of modal interchanges by a factor of almost four with only a 7 % increase in travel time. Figures for `de-road-rail` are similar: All paths use the rail network, and enabling constraints reduces the number of modal interchanges by a factor of almost 3.5 with only little increase in travel time. On our sparser long-distance networks the effects are less pronounced. For example, on `wo-road-rail-flight`, we see that 89 % of the paths already follow a hierarchical use of transportation modes, and the difference in the number of modal interchanges decreases only by 0.4. However, while this difference may seem small, we argue that model constraints are nevertheless important, since our experiment shows that in 11 % of the cases the (unconstrained) path violates the modal constraints, which may render it completely infeasible to the user.

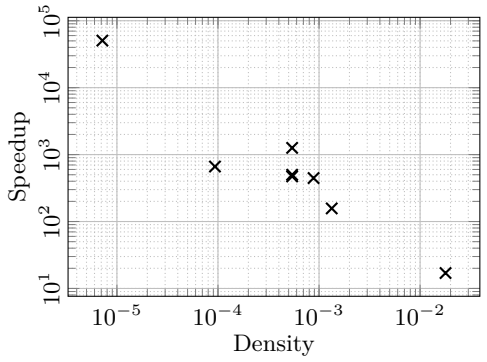

Figure 5: Evaluating the speedup of UCCH subject to the density of the input from Table 5.

For a detailed discussion of UCCH please refer to ECOMPASS-TR-058.

2.2.3 Assessment on the multimodal transportation network of Berlin

Improved Multimodal Multi-Criteria Route Planning. From M20 onwards, we have adapted our final assessment of the multimodal multi-criteria prototype to the data sources available for the metropolitan area of Berlin. The considered multimodal network of Berlin consists of public transit, walking, taxi, and points of interests. The public transit network provided to us by the Verkehrsverbund Berlin-Brandenburg (VBB) has 12 876 stops, 7 450 routes, 55 250 trips, and 1 230 735 daily

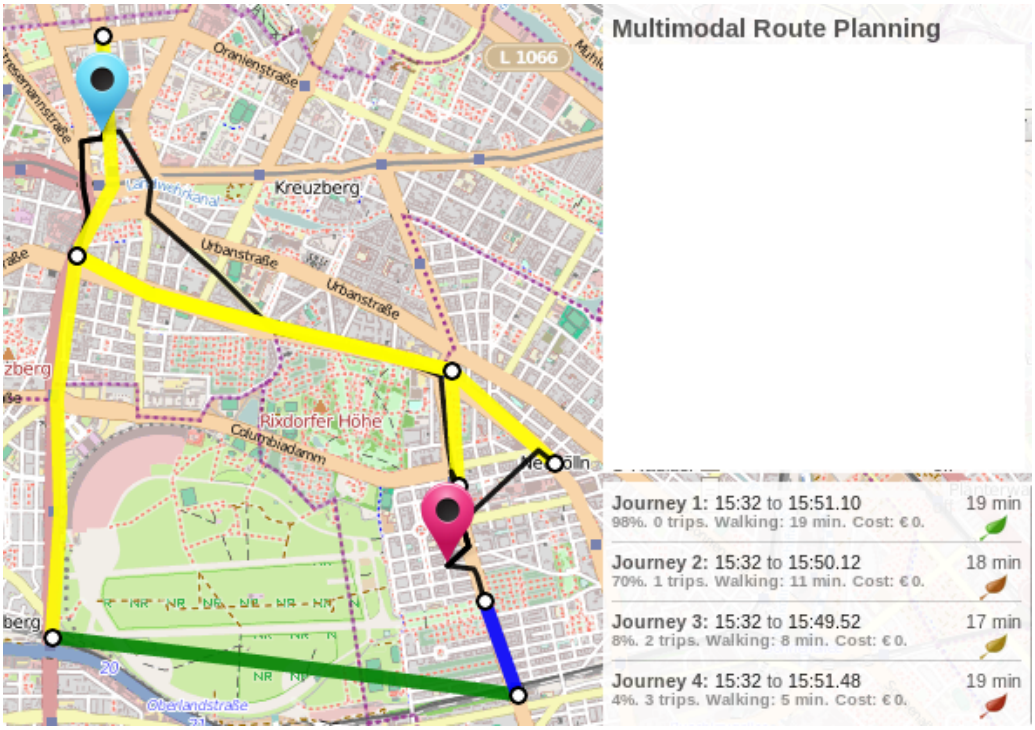

Figure 6: Multi-criteria optimization of multimodal journeys in Berlin, Germany. For the given source and destination locations several alternative journeys are suggested based on arrival time, walking duration, number of transfers. Eco-friendliness estimates are visualized as colored leafs, where green indicates the most ecological, dark red the most un-ecological travel alternative. This example shows that eco-friendly means of traveling are available at just small costs of additional travel time.

departure events. The walking network consists of 357 018 vertices and 822 196 arcs. The taxi network consists of 175 918 vertices and 332 869 arcs. The multimodal overlay consists of 12 972 vertices and 38 916 links.

To enable the eco-awareness feature of our prototype we have added CO_2 emission calculations to our implementation, based on road characteristics, travel speed and distance traveled, as well as the mode of transportation used. Since VBB, the association of transit operators in Berlin, could not provide us with detailed consumption data, we instead relied on data provided by the UK Department of Energy & Climate Change and Department of Environment, Food & Rural Affairs[1][2].

Figure 6 shows an example of a location-to-location query as produced by our multimodal multi-criteria route planning prototype with emission estimates. For a quantitative evaluation, we ran random queries on a single core of an 8-core Intel Xeon E5-2670 processor clocked at 2.6 GHz, with 64 GiB of DDR3-1600 RAM, 20 MiB of L3 and 256 KiB of L2 cache, compiling on g++ 4.7.1 with optimization level `-O3`. The results are summarized in Table 8. It shows that eco-friendlier ways means of traveling can be chosen among a set of increasingly slower journeys. To obtain a clearer picture, we distinguish between journeys solely based on taxi or pure walking as well as journeys based on a mix of public transit use and walking. Please note that travel times for taxi do not

[1] 2008 Guidelines to Defras GHG Conversion Factors: Methodology Paper for Transport Emission Factors, July 2008, `http://www.gov.uk/defra`

[2] 2013 Government GHG Conversion Factors for Company Reporting: Methodology Paper for Emission Factors, July 2013, `http://www.gov.uk/defra`

Table 8: Relative increase in travel time versus decrease in consumption. We report average figures for journeys made of a single taxi ride, the fastest public transit and walking journey, the second fastest public transit and walking journey, the second-most eco-friendly public transit and walking journey, the first-most eco-friendly public transit and walking journey, and finally average figures for walking-only journeys. We take the fastest public transit and walking journey as a baseline.

	Taxi only	1st fastest	2nd fastest	2nd eco	1st eco	Walking only
Rel. travel time	46.4 %	100.0 %	109.4 %	119.2 %	164.3 %	325.5 %
Rel. consumption	219.7 %	100.0 %	97.1 %	86.8 %	79.2 %	0.0 %

account for the time spent waiting for the taxi. Likewise, if the same journey would be taken by a private car, one would need to consider additional time need for parking and unparking. Accounting for these effects with eight additional minutes on an average journey would increase taxi-only relative travel time to 60–70 %, easily making it the least favorable mode of transportation in terms of travel time–consumption tradeoff. However, even for the public transit mode of transportation CO_2 emission savings can be realized by choosing between bus, tram, underground, as well as different amounts of walking. For the second-most eco-friendly route we see an estimated decrease of 13.2 % emissions at only 19.2 % increase in travel time (which corresponds to about 9 minutes additional travel time in total).

Improved TTDP travel time matrix preprocessing. It was suggested to apply parallelization to the TTDP travel time preprocessing. Indeed, since the required output is a time-dependent matrix between all points of interests (POIs), its computation is easily parallelized per POI.

We reran our experiments on a dual 8-core Intel Xeon E5-2670 processor clocked at 2.6 GHz, with 64 GiB of DDR3-1600 RAM, 20 MiB of L3 and 256 KiB of L2 cache. (Please note that this is a fairly recent, two year old machine, much newer than the one used in Deliverable D3.5.) Our implementation is done in C++ (with OpenMP for parallelization), compiling on g++ 4.7.1 with optimization level `-O3`.

Points of interests for Berlin were obtained as described in Deliverable D3.2. For Berlin our data set contains 733 POIs total. After preprocessing we acquire a multimodal distance matrix with 143 327 755 entries total. Computing these distances on a single core takes approximately 76 minutes. Applying multi-core parallelization, computation time is reduced to 11 minutes. Each of the 733 × 733 POI combinations has on average about 267 distinct walking/public transportation journeys throughout the day. The average walking/public transportation travel time is about 46 minutes, the maximal travel time is about 4.2 hours of walking during the nightbreak of public transit operation.

2.3 Conclusions

In Task 3.6, regarding techniques for robust multimodal route planning, we investigated network decomposition techniques, multi-core and SIMD parallelization to accelerate route planning queries (extending our previous approaches from Deliverable D3.3.1 and D3.3.2). While parallelization proved very helpful in the context of eCOMPASS, we have learned that while (at least our current approach to) network decomposition has large positive effects on transportation networks of national size, it does not contribute to relevant speedups on urban transportation networks, which are in the focus of the eCOMPASS project.

Furthermore, we have extended our multimodal multi-criteria route planning prototype to the transportation network of Berlin, adding the assessment of eco-friendliness based on CO_2 consumption estimates obtained utilizing a principled approach. The multi-criteria optimization approach enables us to deliver the end-user several good alternative travel plans, offering travel choices that range from fast to more environmentally sustainable. By increasing *eco-awareness*, we

aim to enable end-users to establish more eco-friendly traveling styles.

From our final assessment we conclude that both our multimodal multi-criteria route planning prototype and the module for preprocessing TTDP travel time profile distances are the most apt for inclusion in the eCOMPASS pilot. These prototypes will hence be integrated by WP 5 partners in preparation of the WP 6 Berlin pilot.

3 Final assessment of multimodal route planning algorithms using methods from stochasticity and machine learning

3.1 Brief overview of D3.4 algorithmic approaches

Consider a dense public transportation network in which buses, trams, metros, etc. operate with high frequency. Imagine that you would like to use public transport to travel from home to some important appointment that takes place at time t_A. On one hand, one would not like to arrive much earlier than t_A; on the other hand, it is important to really arrive not later than t_A. In an ideal situation, every bus and every tram is on time, and it sufficient to compute a route that is planned to leave the starting point as late as possible but still arrives at the destination before time t_A. However, in reality, traffic can be congested due to a huge amount of vehicles on the street, road work, accidents, etc. Thus, in reality, delays can always occur, so we are interested in the following natural problem. Given two stops s, t, and a time t_A, we are searching for a *robust* route from s to t that likely arrives before time t_A, but still leaves s at a "reasonable" time. In Deliverable D3.4 we described an approach for finding such robust routes. Additional details on the algorithms can be found in eCOMPASS-TR-023. This deliverable presents an improvement of the algorithms. In particular, we present an improvement of the subroutine that generates all feasible routes from s to t. To compare the new methods with the ones presented in D3.4, we performed an extensive experimental study on historic real-world delay data from Zürich.

Mathematical Preliminaries Given a directed graph $G = (V, E)$ and a vertex $v \in V$, the in- and out-neighbourhoods of v are denoted by $N^+(v)$ and $N^-(v)$, respectively. A *walk* in G is a sequence of vertices $\langle v_0, \dots, v_k \rangle$ such that $(v_{i-1}, v_i) \in E$ for all $i \in [1, k]$. A *path* is a walk $\pi = \langle v_0, \dots, v_k \rangle$ such that $v_i \neq v_j$ for all $i \neq j$ in $[0, k]$, i.e. a path is a walk without crossings. A path $\pi = \langle s = v_0, v_1, \dots, v_{k-1}, v_k = t \rangle$ is called an *st*-path, and every contiguous subsequence $\pi' = \langle v_i, \dots, v_j \rangle$ of π is called a *subpath*. The length of a path (or walk) $\pi = \langle v_0, \dots, v_k \rangle$ is k, the number of edges in the walk, and is denoted by $|\pi|$. A path π of length $|\pi| \geq 1$ is called *non-degenerate*. For a walk $w = \langle v_0, \dots, v_k \rangle$ and some vertex $v \in V$, we write $v \in w$ if and only if there exists an index $i \in [0, k]$ with $v = v_i$. For two walks $w_1 = \langle u_0, \dots, u_k \rangle$ and $w_2 = \langle v_0, \dots, v_l \rangle$ with $u_k = v_0$, $w_1 \cdot w_2$ denotes the *concatenation* $\langle u_0, \dots, u_k = v_0, \dots, v_l \rangle$ of w_1 and w_2. Given two integers i, j, we define the function δ_{ij} (Kronecker delta) as 1 if $i = j$ and 0 if $i \neq j$.

Model A *public transportation network* is a directed graph $G = (V, E)$ plus a set of non-degenerate paths (called *lines*) $\mathcal{L}$ in G, such that every vertex and each edge of G belongs to at least one path in $\mathcal{L}$. In other words, the set of lines covers all the vertices and edges of G. We explicitly distinguish two lines that contain the same vertices but have opposite directions (these may be operated under the same identifier in reality). The vertices of G are also called *stops*. In the following, let $M = \sum_{l \in \mathcal{L}} |l|$ denote the sum of the lengths of all lines. For $\beta \in \mathbb{N}$, a sequence of lines $r = \langle l_1, \dots, l_\beta \rangle$ is called an *st-route* if there exist $\beta + 1$ stops $v_0 := s, v_1, \dots, v_{\beta-1}, v_\beta := t$ such that both v_{i-1} and v_i are stops on the line l_i, and the line l_i visits v_{i-1} (not necessarily directly) before v_i. Notice that a line might occur multiple times in r; however, we assume that any two consecutive lines in r are different. For $i \in \{1, \dots, \beta - 1\}$, we say that a *transfer* between the lines l_i and l_{i+1} occurs at the *transfer stop* v_i. Notice that there might exist multiple transfer stops between two lines. The length of the route r is $|r|$, i.e. the number of transfers plus one. We say that a path π *follows* the route $\langle l_1, \dots, l_\beta \rangle$ if π is the concatenation of non-degenerate subpaths of $l_1, \dots, l_\beta$, in this order. For two vertices $s, t \in V$ and an integer $\beta \in \mathbb{N}$, let $\mathcal{R}^\beta_{st}$ denote the set of all *st*-routes with length at most β, i.e. the set of all *st*-routes with at most $\beta - 1$ transfers. Furthermore, we define the *$\mathcal{L}$-distance* from s to t as the length of a minimum *st*-route (i.e., an *st*-route with minimum number of transfers), and denote it by $d^{\mathcal{L}}_G(s, t)$.

Trips and Timetables While the only information associated with a line itself are its consecutive stops, it usually is operated multiple times per day. Each of these concrete realisations that departs at a given time of the day is called a *trip*. Let τ be a trip. We denote the line associated with τ by $L(\tau)$. If $L(\tau)$ contains some stop $v \in V$, then $A(\tau, v)$ denotes the arrival time of τ at v, and $D(\tau, v)$ denotes the departure time of τ at v. In the following, we assume time to be modelled by integers. For a given trip τ, we require $A(\tau, v) \leq D(\tau, v)$ for every stop $v \in L(\tau)$. Furthermore we require $D(\tau, v_1) \leq A(\tau, v_2)$ for every two stops $v_1, v_2 \in V$ where $L(\tau)$ visits v_1 before v_2. A set of trips is called a *timetable*. We distinguish between

1. the *planned* timetable $T_{planned}$. We assume it to be periodic, i.e., every line realised by some trip τ will be realised by a later trip τ' again (not necessarily on the same day).
2. *recorded* timetables T_i that describe how various lines were operated during a given time period (i.e., on a concrete day or during a concrete week). These recorded timetables are concrete executions of the planned timetable.

In the following, *timetable* refers both to the planned as well as to a recorded timetable. We assume that timetables respect the FIFO property, i.e. for two trips τ_1, τ_2 with $L = L(\tau_1) = L(\tau_2)$ we either have $A(\tau_1, v) \leq D(\tau_1, v) \leq A(\tau_2, v) \leq D(\tau_2, v)$ for every stop $v \in L$, or $A(\tau_2, v) \leq D(\tau_2, v) \leq A(\tau_1, v) \leq D(\tau_1, v)$ for every stop $v \in L$. The FIFO property intuitively states that two buses or trams *of the same line* do not overtake each other.

Goal For the rest of this deliverable, we assume that $s, t \in V$ are the departure and the target stop, $\beta \in \mathbb{N}$ is the maximum number of allowed (not necessarily different) lines, $\mathcal{R} = \mathcal{R}_{st}^{\beta}$ is the set of all *st*-routes with length at most β, t_A is the latest allowed arrival time at t, and $\mathcal{T}$ is a set of recorded typical timetables for comparable time periods (e.g., daily recordings for the past Mondays). A *journey* consists of a departure time t_D, a route $\langle l_1, \ldots, l_\alpha \rangle \in \mathcal{R}_{st}^{\alpha}$ with $\alpha \leq \beta$, and a sequence of transfer stops $\langle s_{CH}^{(1)}, \ldots, s_{CH}^{(\alpha-1)} \rangle$. The intuitive interpretation of such a journey is to depart at stop s at time t_D, take the first line l_1 (more precisely, the first arriving trip of the line l_1), and for every $i \in \{1, \ldots, \alpha - 1\}$, leave l_i at stop $s_{CH}^{(i)}$ and take the next arriving line l_{i+1} immediately. Our goal is to compute a recommendation to the user in form of one or more (robust) journeys from s to t that will likely arrive on time (i.e., before time t_A) on a day for which the concrete travel times are not known yet. We formalise the notion of robustness later.

A Similarity-Based Approach In Deliverable D3.4 we investigated how a general approach to robust optimisation designed by Buhmann et al. [13] can be applied for computing robust journeys. Let $T \in \mathcal{T}$ be a timetable and $\gamma \in \mathbb{N}_0$. We define an *approximation set* $A_\gamma(T)$ as the set of all routes $r \in \mathcal{R}$ such that there exists a journey along r that departs at s at time $t_A - \gamma$ or later, and that arrives at t at time t_A or earlier (both times refer to timetable T). The major advantage of this definition over classical approximation definitions (such as the multiplicative approximation) is that we can consider multiple recorded timetables at the same time, and that the parameter γ still has the direct interpretation as the time that we depart before t_A. Especially, if we consider approximation sets $A_\gamma(T_1), \ldots, A_\gamma(T_k)$ for $T_1, \ldots, T_k \in \mathcal{T}$, every approximation set contains only routes that are realised in the same time period $[t_A - \gamma, t_A]$, and that are therefore comparable among different approximation sets.

The approach of Buhmann et al. [13] expects that exactly two timetables $T_1, T_2 \in \mathcal{T}$ are given. To compute a *robust* route when only two timetables are available, we consider $A_\gamma(T_1) \cap A_\gamma(T_2)$: the only chance to find a route that is likely to be good in the future is a route that was good in the past for *both* recorded timetables. The parameter γ determines the size of the intersection: if γ is too small, the intersection will be empty. If γ is too large, the intersection contains many (and maybe all) *st*-routes, and not all of them will be a good choice. Assuming that we knew the "optimal" parameter γ_{OPT}, we could pick a route from $A_{\gamma_{OPT}}(T_1) \cap A_{\gamma_{OPT}}(T_2)$ at random. Buhmann et al.

[13] suggest to set γ_{OPT} to the value γ that maximises the so-called *similarity* of the timetables T_1 and T_2 at value γ,

$$S_\gamma = \frac{|\mathcal{R}||A_\gamma(T_1) \cap A_\gamma(T_2)|}{|A_\gamma(T_1)||A_\gamma(T_2)|}. \tag{1}$$

The maximum similarity $S_{\gamma_{OPT}}$ is a number indicating how "similar" the two timetables T_1 and T_2 are. Assume, for example, that T_1 and T_2 are completely identical, i.e. they contain the same trips with identical arrival and departure times at each stop of the corresponding lines. Then, for every γ we have $A_\gamma(T_1) = A_\gamma(T_2) = A_\gamma(T_1) \cap A_\gamma(T_2)$, and $S_\gamma = |\mathcal{R}|/|A_\gamma(\cdot)|$. Now it is easy to see that the similarity is maximised when the approximation sets are as small as possible. Thus, we obtain $\gamma_{OPT} = \min_\gamma \{\gamma \mid A_\gamma(\cdot) \neq \emptyset\}$, and the maximum similarity is $S_{\gamma_{OPT}} = |\mathcal{R}|/n_{OPT}$ where n_{OPT} is the number of different optimum routes. On the other hand, timetables can differ substantially. Imagine, for example, that there were only two (tram) lines l_1 and l_2 departing from s, and that there was a time $t_B < t_A$ such that in T_i no trams of line l_i were departing from s after time t_B. Such a situation could possibly occur if the rails of l_i were blocked due to some accident on the day of T_i after time t_B. Obviously we have $\gamma_{OPT} \geq t_A - t_B > 0$, and $|A_{\gamma_{OPT}}(T_1)|$ and $|A_{\gamma_{OPT}}(T_2)|$ might be large while the intersection contains only very few routes. Thus, $S_{\gamma_{OPT}}$ will be much smaller than $|\mathcal{R}|$. For more information about the similarity of instances, refer to [13].

Notice that up to now we did not consider how often a route is realised by a journey in a recorded timetable. For public transportation, this is undesirable: when we pick a route from $A_{\gamma_{OPT}}(T_1) \cap A_{\gamma_{OPT}}(T_2)$ at random, the probability to obtain a route should depend on how frequently it is realised. Therefore we change the definition of $A_\gamma(T)$ to a *multiset* of routes, and $A_\gamma(T)$ contains a route r as often as it is realised by a journey starting at time $t_A - \gamma$ or later, and arriving at time t_A or earlier.

Now the approximation set $A_\gamma(T)$ can be represented by a function $\mu_\gamma^T : \mathcal{R} \to \mathbb{N}_0$, where for a route $r \in \mathcal{R}$, $\mu_\gamma^T(r)$ is the number of journeys starting at time $t_A - \gamma$ or later, arriving at time t_A or earlier and following the route r. Thus, we have $|A_\gamma(T)| = \sum_{r \in \mathcal{R}} \mu_\gamma^T(r)$, and for two recorded timetables T_1, T_2, we need to compute

$$\gamma_{OPT} = \arg\max_\gamma \frac{\sum_{r \in \mathcal{R}} \min(\mu_\gamma^{T_1}(r), \mu_\gamma^{T_2}(r))}{\left(\sum_{r \in \mathcal{R}} \mu_\gamma^{T_1}(r)\right) \cdot \left(\sum_{r \in \mathcal{R}} \mu_\gamma^{T_2}(r)\right)}. \tag{2}$$

After computing the value γ_{OPT}, we pick a route r from $A_{\gamma_{OPT}}(T_1) \cap A_{\gamma_{OPT}}(T_2)$ at random according to the probability distribution defined by

$$p_r := \frac{\min\left(\mu_{\gamma_{OPT}}^{T_1}(r), \mu_{\gamma_{OPT}}^{T_2}(r)\right)}{\sum_{r \in \mathcal{R}} \min\left(\mu_{\gamma_{OPT}}^{T_1}(r), \mu_{\gamma_{OPT}}^{T_2}(r)\right)}. \tag{3}$$

An alternative strategy consists of picking a route r from the intersection which has a maximum number of realisations.

After computing γ_{OPT} and picking a route r from the intersection (using one of the two strategies mentioned above), we have to suggest a time t_D for departing at s. A natural choice might be $t_D = t_A - \gamma_{OPT}$, but this has disadvantages in the following (realistic) situation. Let l be the first line of the suggested route r, and assume that the given timetables T_1 and T_2 both contain a trip of line l departing at time γ_{OPT}. Now if this trip is late by one minute in both instances, $t_D = t_A - \gamma_{OPT}$ is a bad choice if no delay in a future instance occurs: since we will miss the trip departing on time, we have to wait for the succeeding trip of the same line which might lead to an arrival at t only after time t_A. Assuming that a planned timetable $T_{planned}$ is available, let $\tau_1, \ldots, t_k$ be the trips of line l in this planned timetable. We set

$$t_D = \max_{i \in [1,k]} \{D(\tau_i, s) \mid D(\tau_i, s) \leq t_A - \gamma_{OPT}\}, \tag{4}$$

i.e. we choose the latest departure time at stop s of a trip of line l which departs at least γ_{OPT} time units before the latest allowed arrival time t_A.

3.2 New Improved Algorithm for Enumeration of all Solutions

For this section we assume that a public transportation network $G = (V, E)$ with lines $\mathcal{L}$, and two stops $s, t \in V$ are given. We first describe an algorithm that computes an st-path which follows a *minimum* route r. After that we describe an algorithm that, given $\beta \in \mathbb{N}$, enumerates all paths π which follow an st-route from $\mathcal{R}^{\beta}_{st}$, i.e. it enumerates all paths that follow a route of length at most β. Notice that this algorithm generates *paths* (not routes) while the algorithms in the following section assess the robustness of *routes* (not paths). However, for a fixed path π, the set $R(\pi) = \{r \in \mathcal{R}^{\beta}_{st} \mid r \text{ follows } \pi\}$ can easily be computed, and we can just unite these sets to obtain $\mathcal{R}^{\beta}_{st}$, the set of all st-routes with length at most β. This section is a joint work with Gustavo Sacomoto and Marie-France Sagot. More information about the presented algorithms, especially the analysis of their running time, can be found in eCOMPASS-TR-056.

Finding a Solution with a Minimum Number of Transfers For computing a route with a minimum number of transfers, we first construct a weighted graph $\Gamma[G] = (V[\Gamma] \supset V, E[\Gamma])$ such that for any two vertices $s, t \in V$, the cost of a shortest st-path in $\Gamma[G]$ is exactly $d^{\mathcal{L}}_G(s, t)$, the $\mathcal{L}$-distance from s to t. For a given vertex $v \in V$, let $\mathcal{L}_v \subseteq \mathcal{L}$ be the set of all lines that contain v. We add every vertex $v \in V$ to $V[\Gamma]$. Additionally, for every vertex $v \in V$ and every line $l \in \mathcal{L}_v$, we create a new vertex v_l and add it to $V[\Gamma]$. The set $E[\Gamma]$ contains three different types of edges:

1) For every vertex w that is the direct successor of a vertex v on a line l, we create a *travelling* edge (v_l, w_l) with cost 0. These edges are used for travelling along a line l.

2) For every vertex v and every line $l \in \mathcal{L}(v)$, we create a *boarding* edge (v, v_l) with cost 1. These edges are used to board the line l at vertex v.

3) For every vertex v and every line $l \in \mathcal{L}(v)$, we create a *leaving* edge (v_l, v) with cost 0. These edges are used to leave the line l at vertex v.

Figure 3.2 shows an example of the graph construction.

Theorem 1. *A minimum st-route r and a corresponding st-path π that follows r can be computed in time $\mathcal{O}(M \log M)$.*

Proof. We compute the graph $\Gamma[G]$ and run Dijkstra's algorithm in the vertex s. Let π_{st} be a shortest st-path in $\Gamma[G]$. It is easy to see that the cost of π_{st} is exactly $d^{\mathcal{L}}_G(s, t)$. Furthermore, π_{st} induces an st-path π in G by replacing every travelling edge (v_l, w_l) by (v, w), and ignoring the edges of the other two types. Analogously, a corresponding route r can be extracted from π_{st} by considering only the boarding edges (we start with the empty line sequence $r = \langle\rangle$, and every time that π_{st} traverses a boarding edge (v, v_l) for a vertex $v \in V$ and a line $l \in \mathcal{L}$, we append l to r). Since only the boarding edges have cost 1 while all other edges have cost 0, we have $|r| = d^{\mathcal{L}}_G(s, t)$, and therefore r really is a minimum route.

For every vertex v served by a line l, the graph $\Gamma[G]$ contains at most two vertices (namely, v_l and v), thus we have $|V[\Gamma]| \in \mathcal{O}(M)$. Furthermore, $E[\Gamma]$ contains every edge e of every line, and exactly two additional edges for every vertex v_l. Thus we obtain $|E[\Gamma]| \in \mathcal{O}(M)$. Since Dijkstra's algorithm runs in time $\mathcal{O}(|V| \log |V| + |E|)$, both the route r as well as a corresponding path π can be computed in time $\mathcal{O}(M \log M)$. □

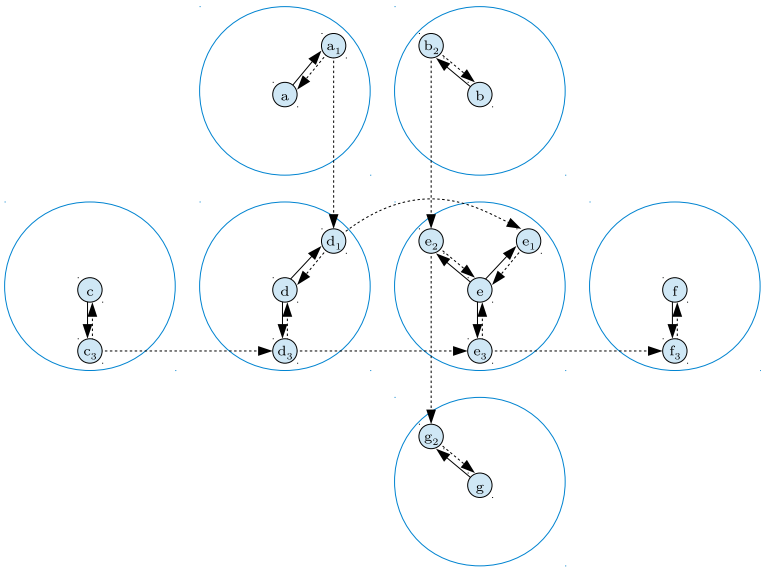

Figure 7: Consider a public transportation network G with the vertices $V = \{a, \ldots, g\}$ and three lines $l_1 = \langle a, d, e\rangle$, $l_2 = \langle b, e, g\rangle$ and $l_3 = \langle c, d, e, f\rangle$. The figure shows the graph $\Gamma[G]$ for this public transportation network. Dotted lines represent edges of cost 0, solid lines represent edges of cost 1. The blue circles represent meta-vertices of the corresponding stations.

Generating all Solutions Let $\mathcal{P}^{\beta}_{st}(G, \mathcal{L})$ denote the set of all st-paths π that follow a route r with length at most β. We will now present an algorithm that enumerates all solutions in $\mathcal{P}^{\beta}_{st}(G, \mathcal{L})$. It is important to stress that the order in which the solutions are output in the algorithm is fixed, but arbitrary. The algorithm, inspired by the binary partition method, recursively partitions the solution space at every call until the considered subspace is a singleton (i.e., contains only one solution) and in that case outputs the corresponding solution. At a generic recursive step on some vertex u (initially, $u = s$), let π_{su} be the su-path discovered so far (initially, $\pi_{su} = \langle s\rangle$). Let G' be the graph that we obtain after removing all vertices in π_{su} except u from G (initially, $G' = G$). To bound the overall running time of the algorithm, we maintain the invariant that the current partition contains at least one solution. More precisely, the algorithm works as follows.

Invariant: (I) There exists at least one ut-path π_{ut} in G' that extends π_{su} so that it belongs to $\mathcal{P}^{\beta}_{st}(G, \mathcal{L})$, i.e. $\pi_{su} \cdot \pi_{ut} \in \mathcal{P}^{\beta}_{st}(G, \mathcal{L})$.

Base case: When $u = t$, output the st-path π_{su}.

Recursive rule: Let $\mathcal{P}_{\beta}(\pi_{su}, u, G')$ denote the set of st-paths to be listed by the current recursive call, i.e. the subset of paths in $\mathcal{P}^{\beta}_{st}(G, \mathcal{L})$ that have prefix π_{su}. This set is the union of the following disjoint sets, for each edge $e = (u, v)$ outgoing from u:

- The st-paths in $\mathcal{P}_{\beta}(\pi_{su} \cdot e, v, G' - u)$ that use e, where $G' - u$ is the subgraph of G' after the removal of u and all its incident edges. In order to maintain the invariant (I), we only perform this recursive call when $\mathcal{P}_{\beta}(\pi_{su} \cdot e, v, G' - u)$ is not empty. We will immediately see how this condition can efficiently be checked.

Algorithm 1 implements this recursive partition strategy. The solutions are only output in the leaves of the recursion tree (which correspond to the base case described above, step 2 in the algorithm), where the partition is always a singleton. Moreover, in order to guarantee that every leaf in the recursion tree outputs one solution, we have to test if $\mathcal{P}_{\beta}(\pi_{su} \cdot e, v, G' - u)$ is not empty before the recursive call (step 9). Given a prefix π_{su}, let $d(\pi_{su}, e, l_i)$ be the length of a minimum

route $(\pi_{su} \cdot e, \gamma)$ such that l_i is the last line of γ. For a line $l \in \mathcal{L}$, let $l|_{G'-u}$ be the line that we obtain from l after removing all edges from l that are *not* contained in $G' - u$. Notice that $l|_{G'-u}$ might potentially be disconnected. Let $\mathcal{L}' - u = \{l|_{G'-u} \mid l \in \mathcal{L}\}$, and let $d^{\mathcal{L}'-u}_{G'-u}(v, t, l_j)$ be the $(\mathcal{L}' - u)$-distance from v to t in $G' - u$ such that l_j is the first line used. For a vertex $v \in V$, let $\mathcal{L}_v \subseteq \mathcal{L}$ be the set of all lines that contain an outgoing edge from v. Analogously, for an edge $e \in E$, let $\mathcal{L}_e$ be the set of all lines that contain e. Now, the set $\mathcal{P}_\beta(\pi_{su} \cdot e, v, G' - u)$ is not empty if and only if

$$\min\left\{d(\pi_{su}, e, l_i) - \delta_{ij} + d^{\mathcal{L}'-u}_{G'-u}(v, t, l_j) \mid l_i \in \mathcal{L}_e \text{ and } l_j \in \mathcal{L}_v\right\} \leq \beta.$$

Basically, $\min\{d(\pi_{su}, e, l_i) - \delta_{ij} + d^{\mathcal{L}'-u}_{G'-u}(v, t, l_j) \mid l_i \in \mathcal{L}_e \text{ and } l_j \in \mathcal{L}_v\}$ is the length of the minimum route that contains the prefix $\pi_{su} \cdot e$. Now we see that $\mathcal{P}_\beta(\pi_{su} \cdot e, v, G' - u, \mathcal{L}' - u)$ is empty if and only if the minimum route has length larger than β.

Observe that we can compute the values $d(\pi_{su}, e, l_i)$ and $d^{\mathcal{L}'-u}_{G'-u}(v, t, l_j)$ using the algorithm presented at the beginning of this section. The values $d(\pi_{su}, e, l_i)$ need to be computed only for edges $e = (u, v) \in E$, and only for lines $l_i \in \mathcal{L}_e$. Consider the graph G'' that contains every edge from π_{su} and every edge $(u, v) \in E$, and that contains exactly the vertices incident to these edges. Now we compute $H = \Gamma[G'']$ and run Dijkstra's algorithm from the vertex s. For every $v \in N^-(u)$ and every line $l_i \in \mathcal{L}_e$, the length of a shortest path in H from s to v_{l_i} is exactly $d(\pi_{su}, e, l_i)$. For computing $d^{\mathcal{L}'-u}_{G'-u}(v, t, l_j)$, we can consider the $(\mathcal{L}' - u)$-distances from t in the reverse graph $G'^R - u$ (with all the edges and lines in $\mathcal{L}' - u$ reversed). Considering G' instead of G ensures that lines don't use vertices that have been deleted in previous recursive calls of the algorithm. Thus we compute $\Gamma[G'^R - u]$ and start Dijkstra's algorithm from the vertex t. Then, the length of a shortest path in $\Gamma[G'^R - u]$ from t to v_{l_j} is exactly $d^{\mathcal{L}'-u}_{G'-u}(v, t, l_j)$.

Algorithm 1: ListPaths(π_s, u, G')

```
if u = t then
    Output(π_s)
    return
end
Compute d(π_su, (u, v), l_i) for each v ∈ N⁻(u) and l_i ∈ L_(u,v)
Compute d^{L'-u}_{G'-u}(v, t, l_j) for each v ∈ G' − u and l_j ∈ L_v
for e ← (u, v) ∈ E do
    d ← min{d(π_su, e, l_i) + d^{L'-u}_{G'-u}(v, t, l_j) − δ_ij | i ∈ L_e and l_j ∈ L_v}
    if d ≤ β then
        ListPaths(π_su · e, v, G' − u, L' − u)
    end
end
```

Theorem 2. *Algorithm 1 runs in time $\mathcal{O}(nM \log M \cdot K)$, where is K the number of returned paths.*

3.3 Additional Methods for Assessing Robustness of Solutions

Let $T \in \mathcal{T}$ be some timetable and $r = \langle l_1, \ldots, l_\alpha \rangle \in \mathcal{R}$ be a route. Let $\tau_1, \ldots, \tau_k$ be the trips of line l_1 in T. We define

$$\gamma(r, T) = \min_{i \in [1,k]} \left\{ t_A - D(\tau_i, a) \;\middle|\; \begin{array}{l} \tau_i \text{ can be extended to a journey along } r \text{ that} \\ \text{arrives in } T \text{ at stop } t \text{ at time } t_A \text{ or earlier} \end{array} \right\} \tag{5}$$

which intuitively can be interpreted as follows: to arrive on time using route r on the day at which T is realised, one has to leave s at least $\gamma(r,T)$ units of time before the latest allowed arrival time t_A. Let $f : (\mathbb{R}^+)^{|\mathcal{T}|} \to \mathbb{R}$ be some objective function (some examples for reasonable choices of f follow immediately). Now we search a route $r \in \mathcal{R}$ that minimises

$$f\big(\gamma(r,T_1),\ldots,\gamma(r,T_{|\mathcal{T}|})\big). \tag{6}$$

For the sake of simplicity we write γ_i instead of $\gamma(r,T_i)$. However, notice that the values γ_i are not constant but still depend on the route r.

3.3.1 A Mean-Risk Model

This section describes the *mean-risk model*, which was just recently used for finding robust routes in private transportation [48], but also has numerous other applications such as portfolio optimisation. Let $c \in \mathbb{R}_0^+$ be the *risk-aversion coefficient*, where $c = 0$ corresponds to the situation where the risk is being completely ignored. The objective function associated with this model is

$$f^c_{MR}\big(\gamma_1,\ldots,\gamma_{|\mathcal{T}|}\big) = \mathrm{Mean}\big(\gamma_1,\ldots,\gamma_{|\mathcal{T}|}\big) + c\cdot\sqrt{\mathrm{Var}\big(\gamma_1,\ldots,\gamma_{|\mathcal{T}|}\big)}, \tag{7}$$

where the mean and the variance of $\gamma_1,\ldots,\gamma_{|\mathcal{T}|}$ are defined as

$$\mathrm{Mean}\big(\gamma_1,\ldots,\gamma_{|\mathcal{T}|}\big) = \frac{1}{|\mathcal{T}|}\sum_{i=1}^{|\mathcal{T}|}\gamma_i \tag{8}$$

$$\mathrm{Var}\big(\gamma_1,\ldots,\gamma_{|\mathcal{T}|}\big) = \frac{1}{|\mathcal{T}|-1}\sum_{i=1}^{|\mathcal{T}|}\big(\gamma_i - \mathrm{Mean}(\gamma_1,\ldots,\gamma_{|\mathcal{T}|})\big)^2. \tag{9}$$

Let r_{OPT} be a route that minimises $f^c_{MR}(\gamma(r,T_1),\ldots,\gamma(r,T_{|\mathcal{T}|}))$. We set

$$\gamma_{OPT} := f^c_{MR}(\gamma(r_{OPT},T_1),\ldots,\gamma(r_{OPT},T_{|\mathcal{T}|})), \tag{10}$$

and, as in Section 3.1, search in the planned timetable $T_{planned}$ for a journey along r_{OPT} that leaves s as late as possible, but at least γ_{OPT} time units before t_A.

3.3.2 Norm-Based Approaches

Let $p \in [1,\infty]$ be a real-valued number. As an objective function we use

$$f^p_{NORM}\big(\gamma_1,\ldots,\gamma_{|\mathcal{T}|}\big) = \big\|\big(\gamma_1,\ldots,\gamma_{|\mathcal{T}|}\big)\big\|_p. \tag{11}$$

It is easy to see that we have

$$f^1_{NORM}\big(\gamma_1,\ldots,\gamma_{|\mathcal{T}|}\big) = \sum_{i=1}^{|\mathcal{T}|}\gamma_i = |\mathcal{T}|\cdot\mathrm{Mean}\big(\gamma_1,\ldots,\gamma_{|\mathcal{T}|}\big) \tag{12}$$

$$f^\infty_{NORM}\big(\gamma_1,\ldots,\gamma_{|\mathcal{T}|}\big) = \max_{i\in[1,|\mathcal{T}|]}\{\gamma_i\}. \tag{13}$$

Thus, a route r^1_{OPT} minimising the function f^1_{NORM} is a route which in average departs as late as possible. Since $|\mathcal{T}|$ is constant (i.e., it does not depend on $\gamma_1,\ldots,\gamma_{|\mathcal{T}|}$), every route that is optimal to f^1_{NORM} is simultaneously optimal to f^0_{MR}, the mean-risk approach with a risk-aversion coefficient set to 0.

On the other hand, a route r^∞_{OPT} minimising the function f^∞_{NORM} is a route that minimises the maximum time between the departure and the latest allowed arrival time t_A. The route r^∞_{OPT} can

alternatively be seen as the route that maximises the earliest departure time that is necessary to arrive on time in all instances in $\mathcal{T}$. The norm-based approach with $p = \infty$ can be related to the approximation set-based approach developed in Section 3.1. Let $\gamma_{FI} = \min\left\{\gamma > 0 \mid \bigcap_{i=1}^{|\mathcal{T}|} A_\gamma(T_i) \neq \emptyset\right\}$ be the smallest value for γ such that the intersection of all γ-approximation sets is non-empty. One can observe that every route r contained in $\bigcap_{i=1}^{|\mathcal{T}|} A_{\gamma_{FI}}(T_i)$ minimises f^∞_{NORM} and vice versa.

Now, let $p \in [1, \infty]$ be arbitrary and let r^p_{OPT} be a route that minimises f^p_{NORM}. As for the other approaches, we have to suggest a departure time t_D. However, if $p < \infty$, setting the departure time to $t_A - f^p_{NORM}(\gamma(r^p_{OPT}, T_1), \ldots, \gamma(r^p_{OPT}, T_{|\mathcal{T}|}))$ (as for the mean-risk approach) is far too pessimistic. Imagine that we had $p = 1$, $|\mathcal{T}| = 24$, and the average travel time of an optimum route was an hour. Then, suggesting $t_A - f^1_{NORM}(\cdot)$ as a departure time would lead to a departure of at least one day in advance! Thus we have to develop an alternative strategy for suggesting a reasonable departure time. For this purpose we use the previous insights how f^1_{NORM} and f^∞_{NORM} behave. For $p = \infty$, suggesting $t_A - f^\infty_{NORM}(\cdot)$ as a departure time seems to be reasonable as f^∞_{NORM} finds a route that minimises the maximum time difference between the earliest departure and the latest allowed arrival time. On the other hand, instead of suggesting $t_A - f^1_{NORM}(\cdot)$ as a departure time, it seems reasonable to suggest $t_A - f^1_{NORM}(\cdot)/|\mathcal{T}|$ as f^1_{NORM} finds a route that minimises the average time difference between the earliest departure and the latest allowed arrival time. Now for other values p, we scale the time linearly with respect to $p = 1$ and $p = \infty$. More concretely, we set

$$\gamma^p_{OPT} = f^\infty - \left(\frac{f^p - f^\infty}{f^1 - f^\infty}\right) \cdot \left(f^\infty - f^1/|\mathcal{T}|\right) \tag{14}$$

where $f^q = f^q_{NORM}(\gamma(r^p_{OPT}, T_1), \ldots, \gamma(r^p_{OPT}, T_{|\mathcal{T}|}))$ for every $q \in [1, \infty]$. Notice that in the definition of f^q, r^p_{OPT} is always a route which minimises f^p and not necessarily f^q. It is easy to see that $\gamma^1_{OPT} = f^1/|\mathcal{T}|$ and $\gamma^\infty_{OPT} = f^\infty$. As for the previous approaches we use the optimal route r^p_{OPT} and γ^p_{OPT} to find a journey along r^p_{OPT} that is scheduled to leave s as late as possible, but at least γ^p_{OPT} time units before t_A.

3.4 Experimental Results

3.4.1 Experiments on Synthetic Data

In Deliverable D3.4 we promised to implement the presented algorithms and to evaluate the quality of the suggested journeys. At the time when the deliverable was written, only a planned timetable $T_{planned}$ of the public transportation network of Zürich was available for us. This timetable contained information about the network structure and the planned arrival and departure times, but no real-world delay data was included. Thus, our preliminary experiments used artificially generated delays.

For this purpose, we carefully chose a small set of problematic stops S' where delays usually occur. Then we generated 100 pairs of stops (s, t) uniformly at random. For each pair, we generated three timetables T_1, T_2 and T_3 from $T_{planned}$ by delaying every trip τ in $T_{planned}$ between 0 and 3 minutes at every stop $v \in S'$ (if v occurs on τ). These delays are 0 or 3 minutes with probability 1/8, and 1 or 2 minutes with probability 3/8. Now we computed journeys from s to t using the instances T_1 and T_2 as input for the algorithm(s), and measured the arrival time of the suggested journey(s) with respect to T_3. More details about the experimental setup and the obtained results can be found in eCompass-TR-023.

However, in the meantime, we were provided historic real-world delay data from the Verkehrsbetriebe Zürich (VBZ) which we used for the extensive evaluation and testing of the proposed algorithms.

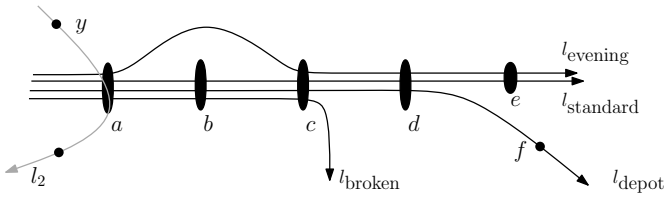

Figure 8: Different trajectories of one line.

3.4.2 Description of the Data

To assess the proposed methods, we performed experiments on real world data from the public transportation network of the city of Zürich, Switzerland. This network consists of 401 stops. The data contains information on tram lines, operating under 15 different labels, bus lines, operating under 22 different labels, and trolleybus lines, operating under 6 different labels. We are also given a planned timetable which contains the scheduled departure times of all the lines at all their stops during the whole day. There are 9999 trips scheduled in the planned timetable.

Moreover, we are given real-world delay information for several days, in the form of past observations. They describe how the scheduled trips were realised in reality. From these observations, we reconstructed timetables, one for each day. For our experiments, we reconstructed the timetables of seven almost consecutive Thursdays for the period from 4 April to 23 May 2013 (except for 9 May, which was a public holiday in Zürich).

In the experiments we model the given scheduled lines as 292 lines. This especially implies that one line label is in average modelled by more than 6.7 lines. Such a situation occurs since there are several lines operating under the same label but with different trajectory. In particular, there is a forward and backward realisation of the line, trajectories that correspond to the vehicles coming from and going to the depot station, etc. We discuss in more details the rationale behind modelling one label with several lines in the following section.

3.4.3 Modelling Challenges

When modelling an urban public transportation network and its behaviour, we try to capture the properties specific to this domain. However, to make a model that is also reasonably simple and clear, we assume certain behaviour. For instance, since all the trams of one line in one direction use the same rail-track, one can assume that such trams do not overtake each other. For the simplicity of the model, we will assume that this FIFO property also holds for buses and other means of transport that are not restricted to rails. While such assumptions mostly hold, in reality certain events can cause them to be violated and one has to decide whether and how to model these situations.

In the following we describe some challenges that arise when dealing with real-world data. We discuss some of our choices either to adapt our model to enhance better the reality or to neglect some situations that occur very rarely.

In particular, defining the lines is a crucial and a nontrivial task. The available data contain the set of stops, the planned timetable capturing the planned trips, and a detailed information on the set of trajectories for each physical vehicle during the day. Also, each trajectory of a physical vehicle has a label that indicates a line. However, there is a problem: Not all the trajectories with the same label correspond to the same sequence of stops. To illustrate this (cf. Figure 8), let us imagine a line l that serves the stops $\langle a, b, c, d, e\rangle$. Even though most of the day this line serves exactly this sequence of stops, twice a day it goes into the depot station and serves $\langle a, b, c, d, f\rangle$, instead. Also, in the evening, for logistic reasons, it skips the stop b and serves $\langle a, c, d, e\rangle$. Moreover, because of a larger delay, the vehicle was once rescheduled (unlike the previous two situations, this is not planned in the timetable) to turn back in advance, resulting in a sequence $\langle a, b, c\rangle$. Even though the user may perceive these sequences as variants of the same line l, it is not clear whether to treat these

lines together as one line or separately. For instance, if one wants to travel from a to f, the line l is recommendable for this purpose only at the two specific times of the day when l goes to the depot. On the other hand, if one wants to reach the stop e, but happens to travel at the time when l goes to the depot, the line l might not be the best choice. In our model, we have chosen to capture the provided information in the following way. The set of stops in our model directly corresponds to the given set of stops from the provided data. In the model, we define one line for every sequence of stops that occurs as a trajectory in the reality. In particular, the line l from the example would be modelled by 3 different lines in our model (we deal with the broken line separately, when it occurs). Clearly, even though this is a viable choice, it also has drawbacks. We believe that unifying the lines, especially if they differ only on parts that do not effect the user, would lead to a better overall frequencies of the lines and result in a better prediction capabilities of all the algorithms and thus more robust solutions. However, it is not easy to do this algorithmically in a concise and systematic way. We plan to consider this issue in a further research. Our goal is not only to find an approach that would allow to join lines of the same label, but also to join lines of different labels on those parts where they serve the same sequence of stops. Then, the recommendation to the user could be in the form "Take the first l_1 or l_5 in the direction D and change to l_4 at stop a."

Let us now describe in more details some real-world situations that influence the behaviour of the lines. We distinguish two categories depending on whether the trajectory is planned in the planned timetable or not.

Topology of the Planned Lines

Standard realisation of a line. In most cases, a line $l = \langle v_1, \ldots, v_k \rangle$ is realised throughout the whole day with high frequency, and there exists a similar line in the opposite direction $l' = \langle v_k, \ldots, v_1 \rangle$ which contains the same stops as line l but in reverse order. Notice that it rarely happens that planned lines change in the evening and leave certain stops out. In Figure 8, the line $l_{evening}$ exhibits such a behaviour.

A vehicle goes to the depot. Such a situation happens roughly twice a day, the vehicle usually operates under the label l, it follows at least partially the path of l, but the end of the line is modified. The problem with this situation is the low frequency of such a line going to the depot: Imagine a situation as in Figure 8. If a user wants to get from the stop y to f, taking the line l_2 and switching to l_{depot} might be the only way to get to f with at most one transfer. If the user wants to travel precisely at one of the few times during the day when l_{depot} runs and this journey worked well on the previous days (i.e., one would arrive on time with this journey), it will be recommended to the user for the next day. However, if the line l_2 is delayed on the next day, and, as a result, the user misses the line l_{depot}, he cannot continue the recommended journey until the next line l_{depot} comes, few hours later. It is not clear whether these journeys containing low frequency lines should be recommended or not. On one hand, if in the past observations they performed well, they should not be suppressed. On the other hand, the failure mode is very inconvenient. In the experiments we choose not to suppress these solutions.

The line l differs in the two directions. That is, the sequences of stops of the forward line l and the backward line l differ. This situation can be caused, for instance, by one way streets. This may lead to problems, because it violates a natural assumption that it is never a good idea to change from some line to the corresponding line in the opposite direction. In the case where the forward and backward lines differ, this might be necessary. For simplicity, we do not address this problem in the experiments. That is, we do not allow an immediate transfer to the line that has the same label.

The line l in fact forms a cycle. That is, the line l does not have a corresponding backward line going in the opposite direction. Instead, after the last stop of the line l in one realisation,

the vehicle which realised it continues to the first stop of the next realisation of l. This is a situation that is rather difficult to capture, since in the data there is some stop that is used as an origin of the line and the cycle is broken accordingly into subsequent realisations of the line. That is, in reality, it might be possible to continue from the last stop of one realisation of the line l to the first stop of the next realisation of l, without a transfer (and without the minimum transfer time needed). For simplicity, in the experiments we deal with these lines as they are given (as separate lines, without the information that they form a cycle) and we do not explicitly capture the cyclic topology.

Behaviour not Planned in the Timetable

A vehicle turns back in advance. Sometimes a vehicle is delayed or there is some incident on the tracks and the strategy to avoid further delay or a dead end consists in turning the vehicle back before reaching the final destination. This results in a line that is realised just partially. This may clearly cause problems because the vehicle unexpectedly does not serve certain stops on the line so some stops that were supposed to be reachable with this line are not. In the experiments, if this happens on the recommended journey, we assume that the user leaves the vehicle before it turns back and waits for the next vehicle of the same line. In reality, there might be an additional vehicle waiting at the turning point and serving the rest of the trip. In the experiments, however, we do not capture this – such a vehicle is not recognised and thus not considered.

Two vehicles of the same line leave the stop at the exact same time. It may happen, due to delay, that two consecutive vehicles of the same line depart at certain stop at the same time (e.g., if the times are specified in minutes only). The problem is that from the perspective of the user standing at the stop, it might be difficult to distinguish which of the two vehicles is delayed and which not, and especially, which of the vehicles will arrive earlier to the destination. In reality, unless the vehicles of the same line overtake (see the next bullet), the vehicle which arrives first to the stop will arrive first also to the destination. In the experiments, since both the vehicles have the same arrival time in minutes, we cannot distinguish which of them arrives first in reality. In these situations, one of the vehicles is considered and the other one is ignored (independently on the arrival times in the destination).

The FIFO property is broken. It seems natural to assume that two vehicles of the same line do not overtake. Even though this is mostly true, in some cases this might be violated. In particular, in these rare cases, the next bus may arrive earlier then the previous one. In our experiments, we decide to ignore these situations (and consider only the first bus that arrives), since in reality, when a bus is coming, this information is not available, and one boards the first bus that comes with the very same assumption.

Other Issues

Nearby stations. Some of the larger stations, where many lines meet, are sometimes split into several stops that are very close to each other, but have slightly different names. The question is whether these stops should be considered together as one station, or not. The argument towards joining would be the fact that one may want to use such compound station as a transfer station. An argument against would be that even though the stops are relatively close to each other, it may not be easy to find them for a person who does not know the area. Thus, it is not clear how much time would one need on such a transfer. Multimodal solution (adding walking arcs) may help to solve the problem, however, it remains difficult to set the minimum transfer times so that the non-local people have enough time to find the right stop, but without creating too much slack time for the people familiar with the area. Furthermore, people with reduced mobility may prefer to avoid transfers between the stops in a compound

station completely. In the experiments, for simplicity, we take the stops as given and we do not create compound stations from nearby stops.

3.4.4 Compared Methods and Used Instances

We experimentally compare several approaches for robust routing in urban public transportation networks. The considered approaches can be divided into three categories, depending on the amount of delay information used for prediction.

Methods that Use Only Information from the Planned Timetable The following simple methods use only the planned timetable and ignore the recorded timetables completely.

Opt-TT: Latest departing journey which arrives on time in the planned timetable. This corresponds to the optimal journey if no delays occur.

Buffer-δ: Latest departing journey which arrives on time in the planned timetable but requires, apart from the minimum transfer time, an additional waiting time of δ minutes at every transfer stop. In the experiments we considered additional waiting times of 1–10, 12, and 15 minutes.

End-Buffer-δ: Latest departing journey which arrives in the planned timetable at least δ minutes before the specified latest arrival time. In the experiments we considered the end buffer times of 1, 3, 5, 7, and 9 minutes.

Methods that Use Delay Data from Exactly Two Recorded Timetables The following routing methods use exactly two recorded timetables for suggesting a journey. Some of these methods could use more timetables as input, we just restricted the number of input instances so that the methods are comparable.

Similarity-MRR: A journey obtained by the similarity-based approach, by picking the most realised route from the intersection.

Similarity-Random: A journey obtained by the similarity-based approach, by picking a route uniformly at random from the intersection. In our experiments we returned the whole intersection and computed the expected arrival time and the expected arrival rate, i.e. we averaged the results over all routes in the intersection. See the next section for more details on the experimental setup.

Mean-Risk-Only2-c: A journey obtained by the mean-risk method when only two instances are used as input, and the the risk-aversion coefficient is set to c. In our experiments we experimented with c set to 0, 0.125, 0.25, 0.5, 1, 2, 4, 8, and 16.

Norm-p-Only2: A journey obtained by the norm-based method when only two instances are used as input, and the norm parameter is set to p. In the experiments we considered the parameter p set to 1, 2, and to ∞. The last variant is denoted by Norm-Inf-Only2.

Methods that Use Delay Data from All/Many Recorded Timetables The following routing methods compute their prediction using all recorded timetables.

Mean-Risk-c: A journey obtained by the mean-risk method when the the risk-aversion coefficient is set to c. In our experiments we experimented with c set to 0, 0.125, 0.25, 0.5, 1, 2, 4, 8, and 16.

NORM-p: A journey obtained by the norm-based method when the norm parameter is set to p. In the experiments we considered the parameter p set to 1, 2, and to ∞. The last variant is denoted by NORM-INF.

To compare the proposed methods, we consider real-world delay data of seven given Thursdays in April and May 2013, namely 4 April, 11 April, 18 April, 25 April, 2 May, 16 May, and 23 May. Notice that in general, the traffic on different weekdays might differ substantially (e.g., the traffic on Mondays behaves completely different than the one on Thursdays). However, many people have weekly regular schedules. Thus, we can expect the traffic on our selected Thursdays to behave similarly, and the occurring delays can be considered "typical". For similar reason we left out 9 May, which was a public holiday in Zürich.

3.4.5 General Results

In the first series of experiments we assess and compare the results obtained by different approaches. For these experiments we used the recorded timetable of 23 May as the test instance. The training instances consist either of the planned timetable, or the recorded timetables of 2 May and 16 May, or all six recorded timetables in the period of 4 April to 16 May for the three types of methods, respectively, as described in Section 3.4.4. Our goal was to perform experiments for a time when the traffic is dense, so we set the latest allowed arrival time t_A to 18:00. We performed 10000 experiments, in each we randomly chose a pair of stops (s, t), and for each method the goal was to compute a journey from s to t. If some algorithm suggested multiple journeys $J_1, \ldots, J_k$ that arrive in the test instance at times $t_A(J_1), \ldots, t_A(J_k)$ (e.g., the SIMILARITY-RANDOM strategy might do this), then we considered the average arrival time $\frac{1}{k}\sum_{i=1}^{k} t_A(J_i)$ and the average chance to arrive on time, $\frac{1}{k}\sum_{i=1}^{k} \mathbb{I}\{t_A(J_i) \leq t_A\}$, where $\mathbb{I}\{t_A(J_i) \leq t_A\}$ has the value 1 if $t_A(J_i) \leq t_A$, and 0 otherwise. If any of the algorithms suggested a journey which is not feasible in the test instance (see section 3.4.3 when such a situation can occur), then for this (s, t) pair, the results of all algorithms were discarded. In our experiments, we considered the following aspects of the proposed journeys:

- Probability that the proposed journey arrives on time in the test instance,
- Departure time of the proposed journey,
- Standard deviation of the arrival time of the proposed journey in the test instance.

These three measures are computed with respect to the individual experiments for each (s, t) pair. Notice that the probabilities to arrive on time of the considered methods have no direct implications on the concrete percentage of delayed vehicles in the network. In particular, an arrival rate of, e.g., 40% of the OPT-TT method does not imply that 60% of the vehicles are delayed.

Arrival Rate vs. Departure Time We compared the probability to arrive on time (arrival rate) with the proposed departure time of the different methods. We display the relation of the methods considering these aspects in Figures 9–14. Intuitively, an earlier departure time leads to a higher probability to arrive on time. Figures 9–14 show that, independent of the considered method, there is a clear trade-off between the departure time and the arrival rate, and thus reconfirm the intuition.

Figures 10 and 11 display the approaches based uniquely on the planned timetable. Since OPT-TT finds the best journey in the planned timetable, but does not consider any delays, it chooses to depart very late, which would be optimal in a no-delay situation, but fails to arrive on time in the presence of delays. Since these methods have no delay information available, better arrival rate is achieved by enforcing earlier departure time directly (by adding minimum spare time at the end of the journey which the END-BUFFER-δ strategy does) or indirectly (by adding minimum spare time on the transfers which the BUFFER-δ strategy does). Intuitively, by increasing

the additional waiting time δ, we achieve a better arrival rate. Interestingly, since no buffer method is dominated by another, all considered values for δ give a meaningful trade-off between the arrival rate and the departure time. However, it is difficult to choose a particular value for δ, because a value that gives a desired arrival rate in one instance may provide an unsatisfactory rate in another instance. On the other hand, by setting δ to a large value, the departure time may be too pessimistic. We note (not shown on the figure) that also the number of transfers of the selected journey influences the performance of the BUFFER-δ strategies, and makes it even more challenging to choose the parameter δ.

Figures 12 and 13 display the approaches based on delay information from 2 instances. The similarity based method needs no parameter tweaking, and still proposed a solution which departs not too early and gives a reasonable arrival rate. The two strategies for the SIMILARITY method, i.e., SIMILARITY-MRR and SIMILARITY-RANDOM, behave similarly. Notice that the SIMILARITY approach gives a slightly better arrival rate than NORM-INF-ONLY2, while departing at the same minute. By increasing the parameter c, the arrival rate of the solution produced by MEAN-RISK-ONLY2-c initially quickly grows. After some threshold, however, the rate begins, surprisingly, to drop again. This is probably because the standard deviation considered is too high and the number of instances is too low to compensate for it. As a result, setting the risk parameter to a value above a certain threshold, the method produces solutions that are dominated by other solutions (with smaller c), and thus strictly worse. We note that we observed the same tendency when we experimented also with other instances (i.e., other pairs of days).

Figure 14 displays the MEAN-RISK-c approach based on delay information from 6 instances. As with 2 instances, by increasing the parameter c, the arrival rate of the produced solution initially quickly grows. However, unlike with 2 instances only, above a certain threshold, the growth of the arrival rate slows down, but does not drop. Thus all the considered values of c yield a meaningful solution. However, one could argue that after a certain point the gain in the arrival rate is very small for the decrease in the departure time.

Figure 9 displays representants of all the methods. The figure shows that the more informed methods clearly have an advantage. For instance, MEAN-RISK-c, for large enough values of c (in the experiments for $c \geq 1$), dominates most of the other methods.

Arrival Rate vs. Standard Deviation Also, we compare the probability to arrive on time vs. the standard deviation on the arrival time of the different methods. We show the relation of the methods in these aspects in Figures 15–20. We observe in the figures that there is a trade-off between arrival rate and the standard deviation.

Figures 16 and 17 display the approaches based solely on the planned timetable. The general tendency shown in the figures is that by increasing the value of δ for the methods BUFFER-δ and END-BUFFER-δ, the arrival rate increases, but so does also the standard deviation. Interestingly, this is not fully true, as some of the methods with small values of δ (e.g., BUFFER-δ for $\delta \leq 4$) are strictly worse and dominated in these aspects by other methods. At the moment we cannot explain this behaviour.

Figures 18 and 19 display the approaches based on delay information from 2 instances. Notice that the SIMILARITY-MRR approach again gives a slightly better arrival rate than NORM-INF-ONLY2 (which corresponds to the first intersection), while having almost the same standard deviation. Both these methods are dominated by NORM-2-ONLY2 (even though the difference between these methods is very small). By increasing the parameter c, the arrival rate of the solution produced by MEAN-RISK-ONLY2-c initially quickly grows with only a small increase of the standard deviation. As we saw already in Figure 13, after some threshold, the rate begins, to drop again. Unfortunately, after this threshold, the standard deviation grows rapidly. Thus, also in these aspects, by setting the risk parameter to a value above a certain threshold, the method produces solutions that are dominated by other solutions (with smaller c), and thus strictly worse (again, we remark that we observed the same tendency in experiments with different choice of the instances).

Figure 20 displays the MEAN-RISK-c approach based on delay information from 6 instances. Again, as with 2 instances, by increasing the parameter c, the arrival rate of the produced solution initially quickly grows, with a rather small increase in the standard deviation. Above a certain threshold for c, the rate continues to grow, but very slowly, and at the same time the standard deviation increases rapidly.

Figure 15 displays representants of all the methods. The figure again shows that the more informed methods clearly have an advantage with respect to the achieved arrival rate. On the other hand, it is clear that with the increase of the probability to arrive on time, the standard deviation increases notably.

In both sets of experiments we have seen the behaviour of the methods considering certain aspects. Most of the methods that require setting a parameter behave well for some values, but for some other values the trade-off becomes rather inefficient (for small increase in arrival rate, a big increase in standard deviation and decrease in departure time is needed). We remark that finding the right parameter is a non-trivial task, as the behaviour of the methods is likely to differ in different set of instances, and even more in a different public transportation network.

Running Time of the Algorithms We implemented the mentioned algorithms for assessing the robustness of routes in Java 7. For enumerating all feasible st-routes, we used the algorithm proposed in eCOMPASS-TR-023. The experiments were performed on one core of an Intel Core i5-3470 CPU clocked at 3.2 GHz with 4 GB of RAM running Debian Linux 7.0. Enumerating all routes takes in average 35ms. Assuming that the set of all routes has already been computed, the trivial buffer strategies have a running time of 1ms or less, the similarity-based methods need in average 8ms while the norm-based and the stochastic-based methods need in average 24ms. Thus, the running times of all algorithms is really fast, and the algorithms could be used in real-world applications.

Notice that our current running times are faster than the ones described in eCOMPASS-TR-023. One reason is that the preliminary experiments were performed on a network with more stops (611 instead of 401) and more lines (471 instead of 292). Although both networks describe the network of Zürich, the one previously used did not only include buses and trams but also boats, cable cars and regional trains, and it also contained stops of rural villages surrounding Zürich. Another reason why the running time of the algorithms decreased is that we considered the running time only for those experiments where some feasible route was found. In our preliminary experiments, we considered all results. Notice that the situation where no feasible route is found occurs only due to some peculiarities in the data (see Section 3.4.3 for details). Furthermore, in the meantime we also improved the code of the implementations to be more efficient.

3.4.6 Results over the Day

In this series of experiments, we compared how the behaviour of the methods changes over the day, influenced by the rush hours and more quiet times of the day. The experimental setup is similar to the one described in Section 3.4.5. However, we did not fix only one latest allowed arrival time, but performed 10000 experiments for multiple values of t_A. More concretely, the value of t_A ranges from 7:30 to 20:00, in steps of 10 (during the rush hours) to 30 (during the quiet times of the day) minutes. We were especially interested in the following aspects:

- Probability that the proposed journey arrives on time in the test instance,
- Standard deviation of the arrival time of the proposed journey in the test instance.

Arrival Rate vs. Time of the Day We studied how the probability to arrive on time changes through the day for different methods. The relation of the methods considering this aspect is shown in Figures 21–26.

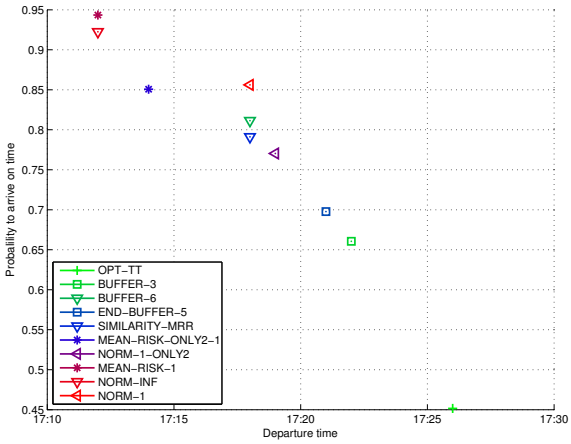

Figure 9: Arrival rate vs. departure time: comparing various methods

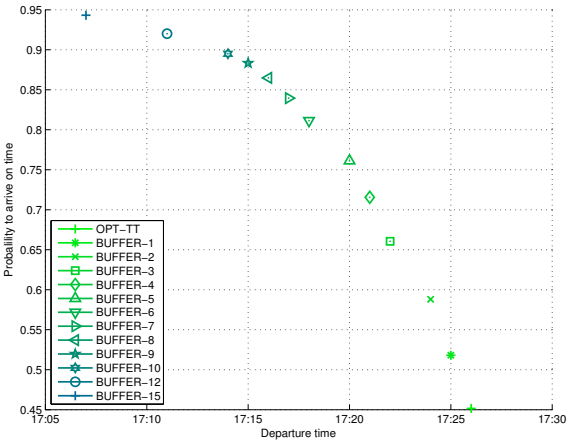

Figure 10: Arrival rate vs. departure time: comparing different buffer times for transfers

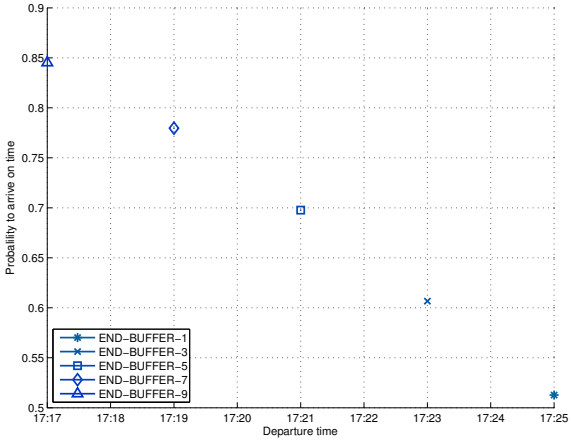

Figure 11: Arrival rate vs. departure time: comparing different buffer times at the end

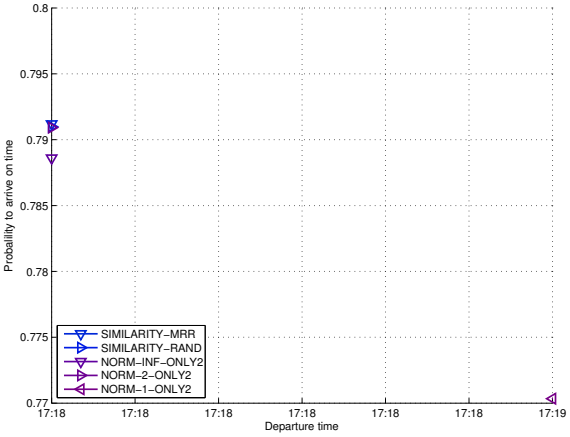

Figure 12: Arrival rate vs. departure time: comparing similarity and norm-based approaches based on 2 given instances

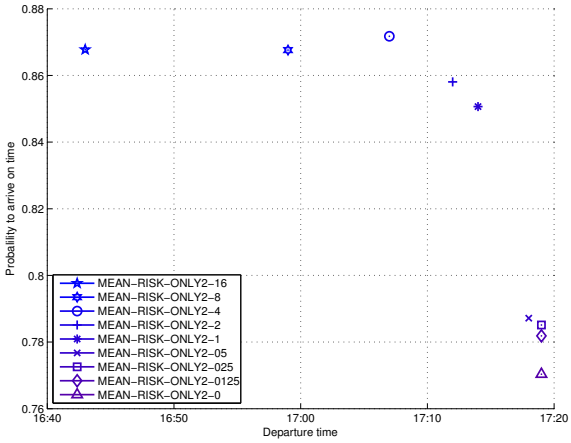

Figure 13: Arrival rate vs. departure time: comparing mean-risk approach based on 2 given instances

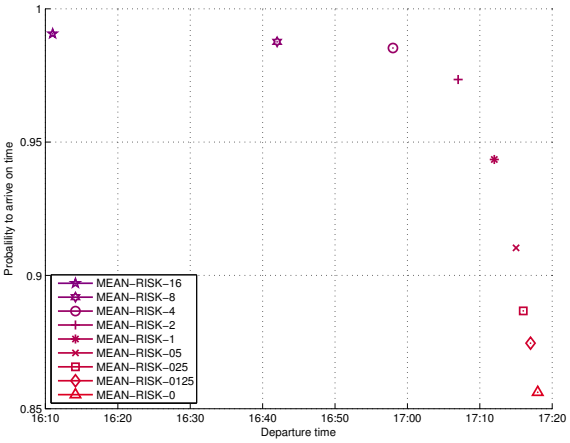

Figure 14: Arrival rate vs. departure time: comparing mean-risk approach based on 6 given instances

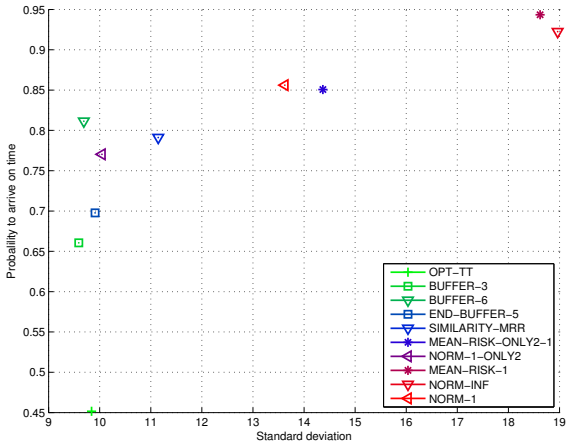

Figure 15: Arrival rate vs. standard deviation: comparing various methods

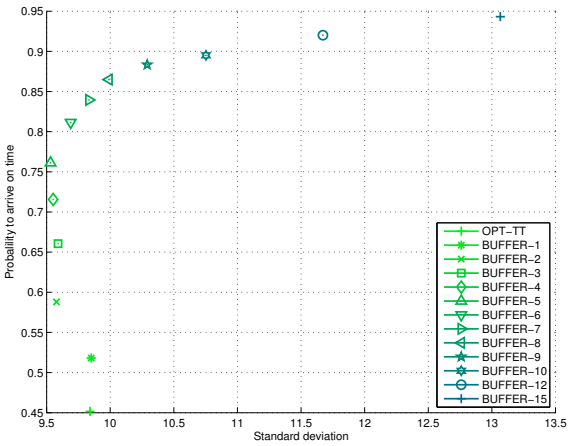

Figure 16: Arrival rate vs. standard deviation: comparing different buffer times for transfers

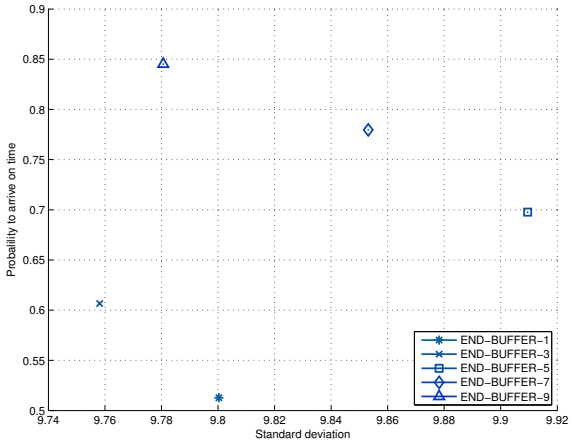

Figure 17: Arrival rate vs. standard deviation: comparing different buffer times at the end

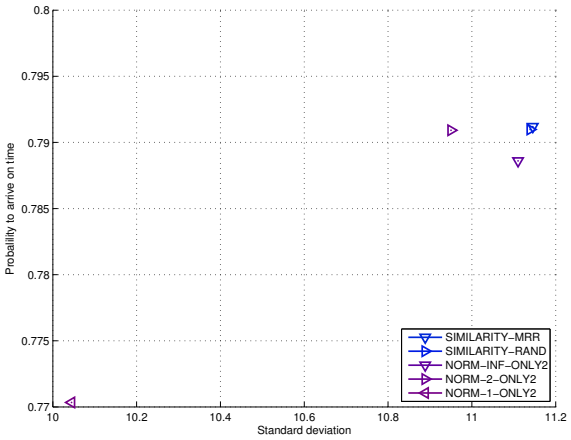

Figure 18: Arrival rate vs. standard deviation: comparing similarity and norm-based approaches based on 2 given instances

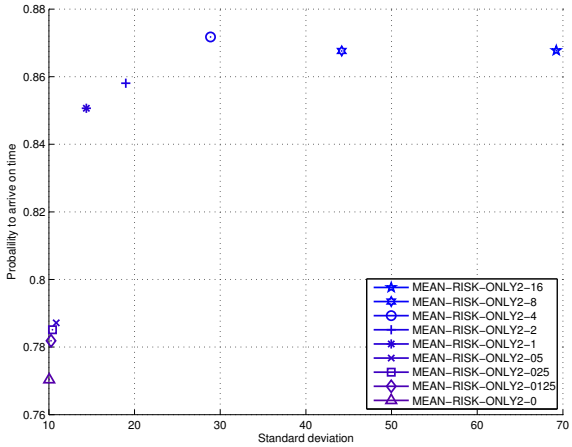

Figure 19: Arrival rate vs. standard deviation: comparing mean-risk approach based on 2 given instances

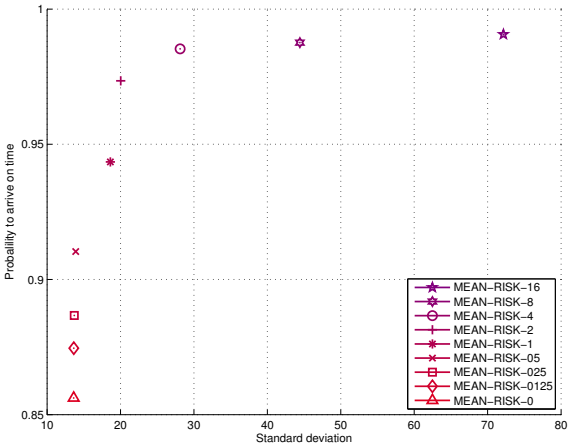

Figure 20: Arrival rate vs. standard deviation: comparing mean-risk approach based on 6 given instances

In all these figures we clearly see that all the considered methods behave similarly through the day, influenced by the peak hours. In particular, independently of the used method, the probability to arrive on time decreases significantly when the desired arrival time is between 8:00 and 9:00, which can be explained by the morning rush hour. After 9:00, this probability quickly increases and reaches its maximum around noon. The probability to arrive on time then drops again noticeably around 17:00–18:00, which can be explained by the afternoon rush hour. Notice that we did not consider arrival times prior to 7:30 or later than 20:00, since we expected the traffic to behave very regular, and thus the situation is less interesting for finding robust routes.

Figures 22–23 show the results for the approaches that compute the prediction based solely on the planned timetable. We reconfirm the observation (from Figures 10 and 11) that by increasing the value of the parameter δ for the methods Buffer-δ and End-Buffer-δ, the arrival rate increases. Notice that the relative order of these methods based on the arrival rate does not change in the course of the day.

Figure 24 compares the similarity-based approaches with the norm-based approach when $p = \infty$ and only two training instances are used. We see that these methods behave almost identically. This is likely due to the fact that the value of similarity is maximised at the smallest γ such that the intersection of the γ-approximation sets is non-empty (which is the behaviour of Norm-Inf-Only2).

Figure 25 shows that the arrival rate of Mean-Risk-c (based on 6 instances) increases with a growing value of the parameter c. Notice that when c is greater than a certain threshold, the arrival rate becomes less affected by the rush hours.

Figure 26 shows the results of the norm based methods, namely Norm-1, Norm-2, and Norm-Inf, all based on 6 input instances. The figure clearly indicates that the arrival rate of the journeys suggested by Norm-1 is significantly lower than the arrival rate of the journeys proposed by the other two methods, independently on the day time.

Figure 21 compares the representants of all the considered methods. Interestingly, some of the methods are affected by the peak hours more than other methods. In particular, the methods based only on the planned timetable (Buffer-δ and End-Buffer-δ) seem to be greatly affected by both morning and afternoon rush hour. On the other hand, the methods that use recorded timetables for suggesting a journey are less affected by the afternoon rush hour than by the morning rush hour. We conjecture that the traffic behaves more "regularly" in the afternoon peak time, and this behaviour is captured in the recorded timetables used as input to these methods.

Standard Deviation vs. Time of the Day We also compared how the standard deviation of the arrival time varies through the day for the different methods. We show the relation of the methods in this aspect in Figures 27–32.

The behaviour of the majority of the considered methods suggests that the standard deviation of the arrival time is significantly greater during the morning rush hour than for the rest of the day. We suspect that this is due to some lines that are realised with low frequency, in particular lines coming from or going to the depot station. Such lines are realised only few times in the morning and then again only few times in the afternoon. Suppose that such a line was selected as a part of a suggested journey. If in the test instance, due to delays, the morning realisation of this line is missed, we need to wait for an afternoon realisation of the line and this may greatly influence the standard deviation of the arrival time. However, if a realisation of this line is missed in the afternoon, there might not be any more realisation of this line during the rest of the day, and the whole test case (i.e., the results of all methods) is ignored and does not the standard deviation of the arrival time. We conjecture that for the same reason, the standard deviation increases between 15:00 and 16:30 which is the time when certain lines are coming from the depot.

Figure 26 displays the results for the Mean-Risk-c approach based on delay information from 6 instances. As we saw in Figure 20, by increasing the parameter c, the standard deviation increases. Interestingly, for values of c above a certain threshold, the standard deviation increases rapidly, and unlike the other methods, Mean-Risk-c then reaches its peak in the afternoon, around 15:00 and

17:00. Currently we cannot explain this behaviour.

3.4.7 Influence of the Test Instance

In this series of experiments we investigated how the behaviour of the methods change in dependence on the test instance. We considered each one of the 7 instances as test instance. The delay information provided to the methods then consists either of the planned timetable, or the two closest recorded timetables (i.e., mostly the instances that correspond to the preceding and the succeeding date), or the six remaining instances. We set the latest allowed arrival time t_A to 18:00. We used each of the seven available recorded timetables as test instance, and for each of these we again performed 10000 experiments similar to the ones described in section 3.4.5. In our experiments, we considered the following aspects of the proposed journeys:

- Probability that the proposed journey arrives on time in the test instance,
- Standard deviation of the arrival time of the proposed journey in the test instance.

In particular, we study how the probability to arrive on time changes for different methods, depending on the chosen test instance.

Figure 33 shows how the average arrival rate changes in dependence of the test instance. It seems that the relative order of the methods does not change much and is comparable to the order shown in Figures 9 and 21. We also observe that all methods exhibit the same tendencies, i.e. there is no test instance that favours a special method. However, the choice of the test instance does influence the concrete probability to arrival on time significantly. For instance, if instance 3 is used as the test instance, then the arrival rates of all methods drop notably. It is possible that on the corresponding day, the traffic behaved differently than on the remaining days and this would explain the drop of the arrival rates for this test instance.

Next, Figure 34 shows how the standard deviation on the arrival time varies in dependence on the chosen test instance and the different methods. The situation differs from the one for the arrival rate shown in Figure 33. The relative order of the methods with respect to the standard deviation of the arrival time changes quite a lot. Also the behaviour of different methods changes substantially: For example, in test instance 3 the suggestions of the Norm-1 strategy have the least standard deviation while in instance 5, seven methods suggest a journey with a significantly smaller standard deviation than the journey proposed by Norm-1. At this moment, we cannot explain this behaviour.

Notice that it seems as if the standard deviation mostly increases when the probability to arrive on time decreases, and vice versa, although this behaviour is not always observable. This tradeoff between a higher chance to arrive on time and the standard deviation of the arrival time coincides with previous observations.

3.4.8 Maximising the Similarity

This section investigates how the similarity-based approach behaves on real-world data, and how the similarity of instances influences the quality of the predictions. First we notice that the maximum similarity $S_{\gamma_{OPT}}$ does not only depend on the two training instances T_1, T_2 but also on the origin and the destination of the route. At first glance, this might seem strange. However, we will now see why this is a reasonable and desirable property. Consider a really huge public transportation network. Imagine that there were two stops $s, t \in V$ that are close to each other, and that most st-routes have no delay at no time, neither in T_1 nor in T_2. Thus, the similarity of T_1 and T_2 with respect to the goal of getting from s to t should be high. On the other hand, imagine that there additionally exist two stops $s', t' \in V$ that are far away, and that many $s't'$-routes occurring in T_1 are not present in T_2 (e.g., due to accidents) and vice versa. Clearly, the similarity of T_1 and T_2 with respect to the goal of getting from s' to t' should be much lower, especially in comparison to the previous case.

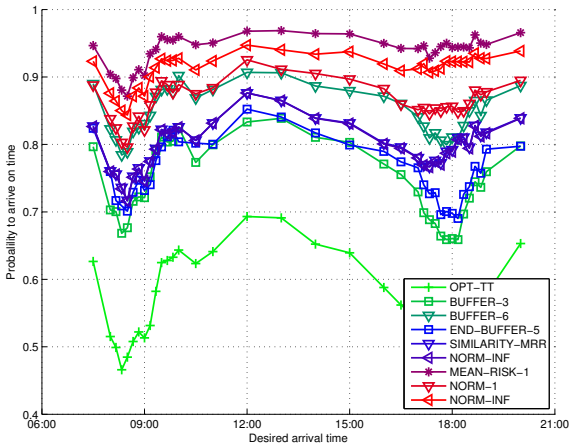

Figure 21: Arrival rate vs. desired arrival time: comparing various methods

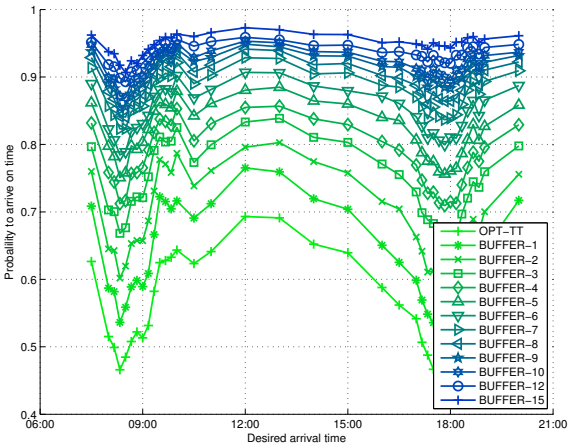

Figure 22: Arrival rate vs. desired arrival time: comparing different buffer times for transfers

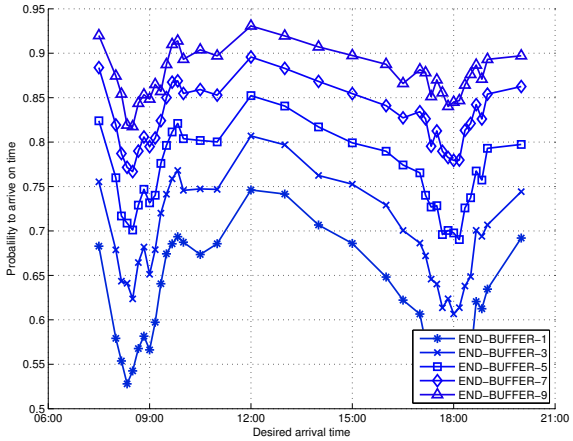

Figure 23: Arrival rate vs. desired arrival time: comparing different buffer times at the end

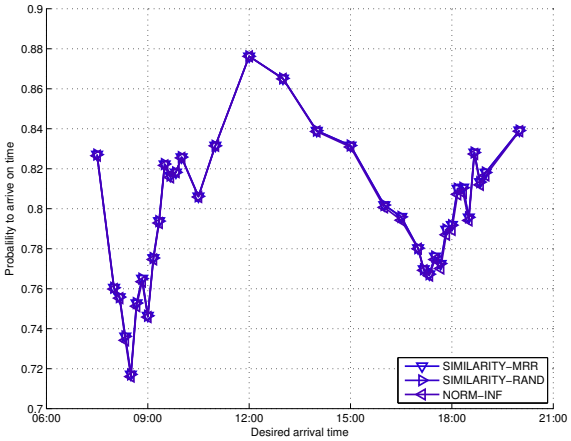

Figure 24: Arrival rate vs. desired arrival time: comparing similarity-based approach and first intersection based on 2 given instances

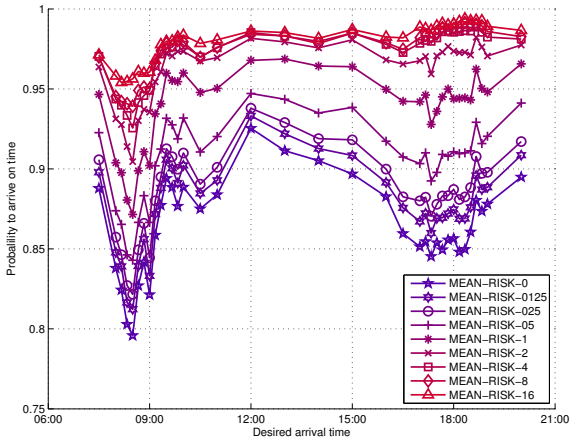

Figure 25: Arrival rate vs. desired arrival time: comparing mean-risk approach based on 6 given instances

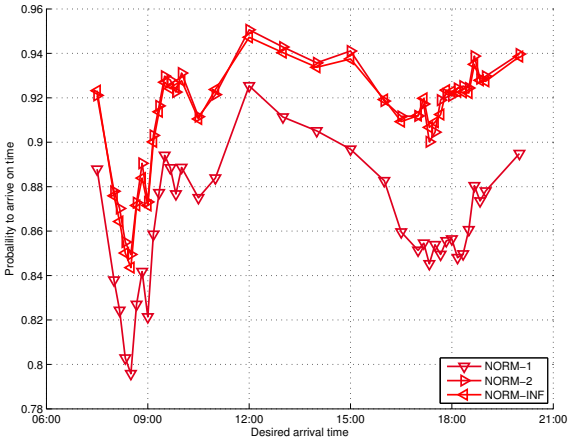

Figure 26: Arrival rate vs. desired arrival time: comparing norm-based approach based on 6 given instances

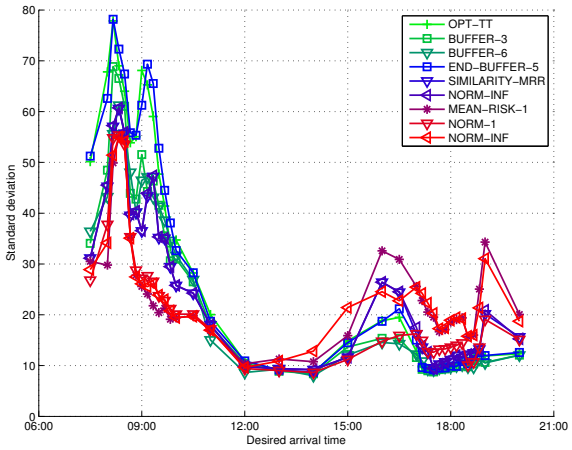

Figure 27: Standard deviation vs. desired arrival time: comparing various methods

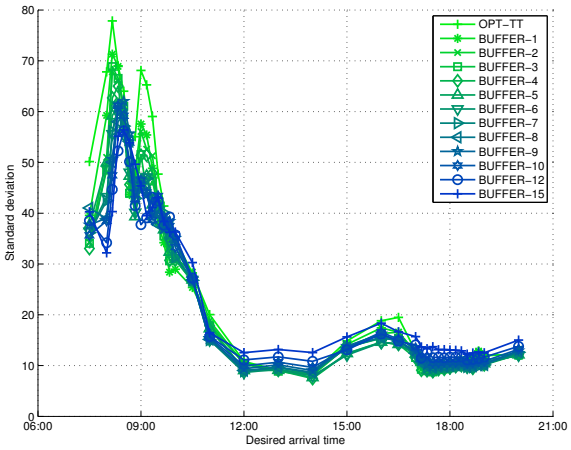

Figure 28: Standard deviation vs. desired arrival time: comparing different buffer times for transfers

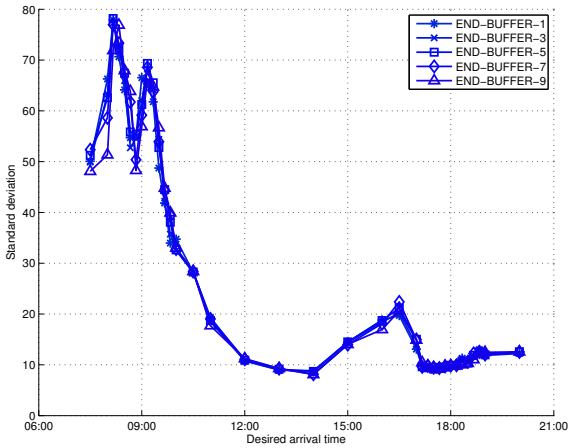

Figure 29: Standard deviation vs. desired arrival time: comparing different buffer times at the end

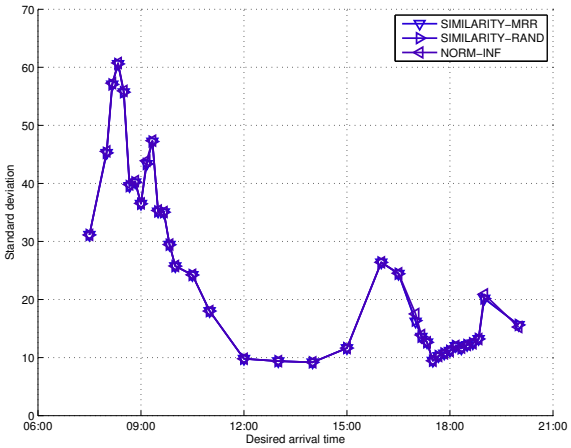

Figure 30: Standard deviation vs. desired arrival time: comparing similarity-based approach and first intersection based on 2 given instances

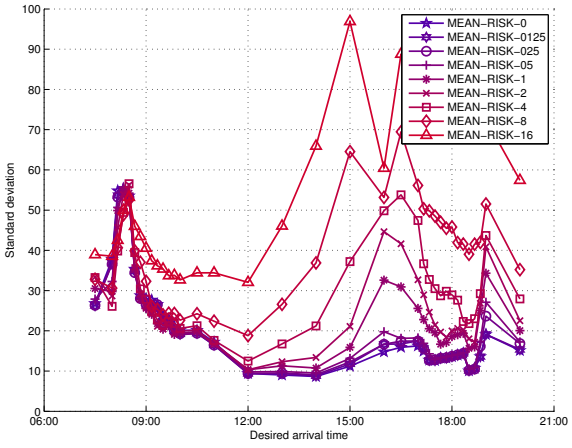

Figure 31: Standard deviation vs. desired arrival time: comparing mean-risk approach based on 6 given instances

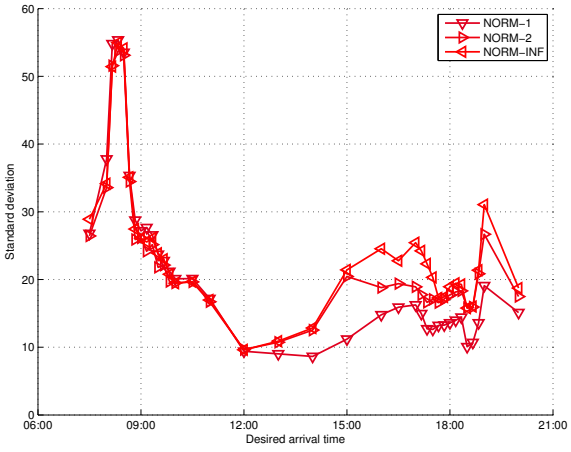

Figure 32: Standard deviation vs. desired arrival time: comparing norm-based approach based on 6 given instances

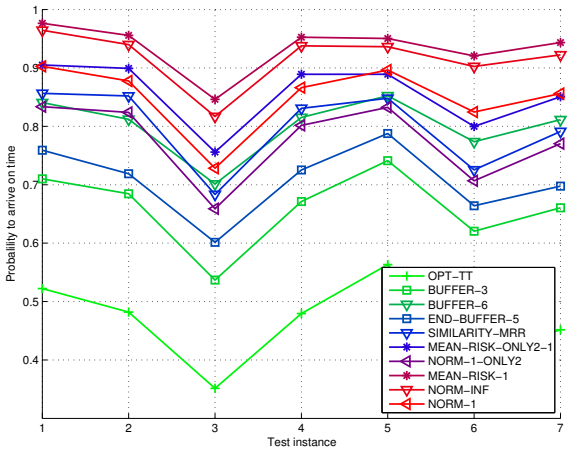

Figure 33: Arrival rate depending on test instance: comparing various methods

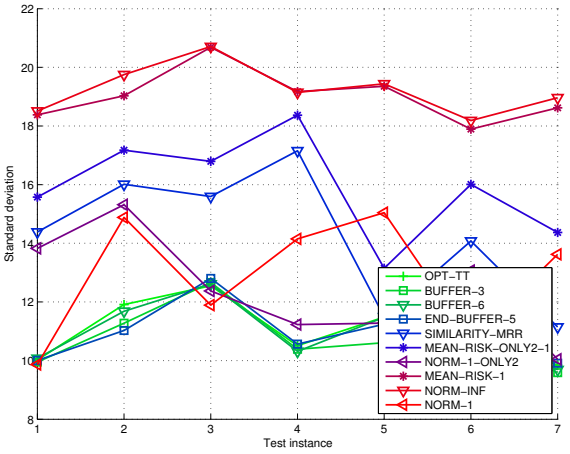

Figure 34: Standard deviation depending on test instance: comparing various methods

For the first series of experiments, we set the latest allowed arrival time t_A to 18:00, fixed a triple (T_1, T_2, T_3) of timetables and selected a set Σ of 10000 pairs of stops (s, t) uniformly at random from the set of all possible stop pairs (s, t) with $s \neq t$. Now for each of these pairs, we used the similarity-based approach with the (training) instances T_1 and T_2 to compute (one or more) robust routes and corresponding departure times. We selected the most frequent route from the intersection of the approximation sets. If the intersection was empty or the suggested route was not realised in T_3 (refer to Section 3.4.3 for situations when such a behaviour can occur), we discarded the pair (s, t) from Σ and did not generate an alternate one. As for the previous experiments, for every pair (s, t) we computed the average arrival time t_A^{st} of the route suggestions in T_3. Furthermore, let $S^{st}_{\gamma_{OPT}}$ be the value of the maximum similarity of T_1 and T_2 when the origin and destination are set to s and t, respectively. We now created a plot that contains the points $(S^{st}_{\gamma_{OPT}}, t_A^{st} - t_A)$ for each $(s, t) \in \Sigma$. Notice that $t_A^{st} - t_A \leq 0$ if and only if the suggested route(s) arrive on time. Thus, $t_A^{st} - t_A$ can be interpreted as the average *lateness* (in minutes) of the suggested routes for the stop pair (s, t).

Figure 35 shows the corresponding plot when the timetables of 18 and 25 April are used for training, and the timetable of the 2 May is used for testing. Figure 36 shows the plot when the timetables of 2 and 16 May are used for training, and the timetable of 23 May is used for testing. The figures indicate that a high similarity alone does not necessarily imply a higher chance to arrive on time. Figure 37 confirms this observation. For some number $z \in \mathbb{R}_0^+$, let Σ_z be the set of all stop pairs $(s, t) \in \Sigma$ with $S^{st}_{\gamma_{OPT}} \geq z$. For $z \in \{0, 25, 50, \ldots, 200\}$, Figure 37 shows the number of such pairs, the arrival rate and the standard deviation of the arrival time (in minutes) when only stop pairs from Σ_z are being considered. We can see that for none of the two selected testing/training triples, a high similarity increases the average arrival rate. Even worse, it seems that a high similarity even lowers the average arrival rate. It also seems that at least the standard deviation decreases a bit when the similarity is not too high. However, for pairs with high similarity, no strong statement about the behaviour of the similarity-based approach can be made. On the other hand one has to take into account that less than 3% of the pairs in Σ have a similarity above 100. This might be a reason why no clear behaviour is exhibited.

At first glance, the results shown in Figures 35–37 are rather disappointing. On the other hand, we just measured the similarity of the training instances T_1 and T_2, but we did not compare the similarity between each test training instance and the test instance T_3. Imagine that both training instances T_1 and T_2 were very similar to each other, but both were not very similar to T_3. In such a situation, a high similarity between T_1 and T_2 clearly does not help to obtain a good prediction

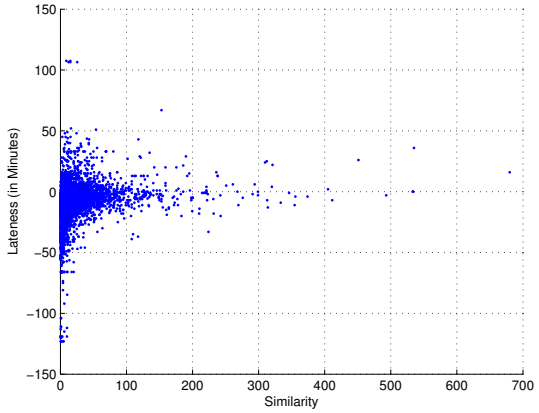

Figure 35: Relation between the similarity and the lateness in minutes. Timetables of 18 and 25 April were used for training, the timetable of 2 May was used for testing.

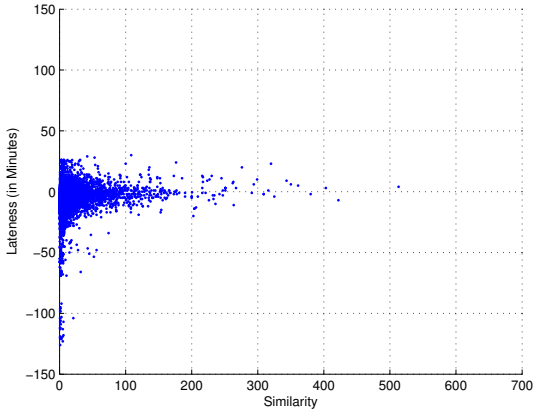

Figure 36: Relation between the similarity and the lateness in minutes. Timetables of 2 and 16 May were used for training, the timetable of 23 May was used for testing.

	18.4, 25.4, 2.5			2.5, 16.5, 23.5		
Similarity	#	Arrival Rate	Standard deviation	#	Arrival Rate	Standard deviation
≥ 0	9,268	85.66%	12.75	9,340	79.66%	11.64
≥ 5	1,520	72.50%	11.07	1,588	68.83%	7.81
≥ 50	578	69.72%	11.02	671	66.62%	7.56
≥ 75	302	69.21%	11.62	367	62.40%	7.12
≥ 100	187	65.78%	13.07	216	58.80%	7.78
≥ 125	126	65.87%	12.67	134	54.48%	7.68
≥ 150	89	66.29%	14.03	87	55.17%	7.95
≥ 175	67	64.18%	12.79	59	50.85%	8.62
≥ 200	51	68.63%	12.51	51	49.02%	8.53

Figure 37: Arrival rates and the standard deviation of the arrival time (in minutes) when only pairs (s, t) with a certain minimum similarity are considered. The lower bound on the similarity is given in the first column. The number of pairs having at least this similarity are given in the second and the fifth column. Columns 2–4 contain the results when the timetables of 18 and 25 April are used for training and the timetable of 2 May is used for testing, Columns 5–7 contain the results when the timetables of 2 and 16 May are used for training and the timetable of the 23 May is used for testing.

for T_3.

We decided to perform a second series of experiments in which we again generated 10000 pairs of stops (s,t) uniformly at random (as before). The experimental setup is similar to the one described in Section 3.4.5. However, this time we selected two triples of timetables, and these two triples did not stay the same over all experiments but were chosen individually for each stop pair (s,t). Let $\mathcal{T}$ be set of all available timetables (for the Zürich network, see Section 3.4.2 for a list of available historic timetables). Furthermore, let Υ be the set of all triples $(T_1,T_2,T_3) \in \mathcal{T}^3$ whose components are mutually different. For a given stop pair (s,t) and two timetables $T_1,T_2 \in \mathcal{T}$, let $S^{st}_{\gamma_{OPT}}(T_1,T_2)$ be the maximum similarity of T_1 and T_2 with respect to the origin s and the destination t. For each stop pair (s,t), we selected a triple whose minimum pairwise similarity is as large as possible,

$$(T_1^{max},T_2^{max},T_3^{max}) = \underset{(T_1,T_2,T_3)\in\Upsilon}{\arg\max} \min\left\{S^{st}_{\gamma_{OPT}}(T_1,T_2), S^{st}_{\gamma_{OPT}}(T_1,T_3), S^{st}_{\gamma_{OPT}}(T_2,T_3)\right\} \tag{15}$$

and, analogously, a triple whose maximum pairwise similarity is as small as possible,

$$(T_1^{min},T_2^{min},T_3^{min}) = \underset{(T_1,T_2,T_3)\in\Upsilon}{\arg\min} \max\left\{S^{st}_{\gamma_{OPT}}(T_1,T_2), S^{st}_{\gamma_{OPT}}(T_1,T_3), S^{st}_{\gamma_{OPT}}(T_2,T_3)\right\} \tag{16}$$

$(T_1^{min},T_2^{min},T_3^{min})$ can be seen as the three least similar instances while $(T_1^{max},T_2^{max},T_3^{max})$ are the three most similar ones. Now, for every stop pair (s,t), we used the instances T_1^{max} and T_2^{max} as input and T_3^{max} for testing, and for comparison, used alternatively T_1^{min} and T_2^{min} as input and T_3^{min} for testing. Figure 38 shows the results when the latest allowed arrival time t_A is set to 9:00, Figure 39 shows the results when t_A is set to 18:00. Notice that even though the mean-risk estimator as well as the norm-based estimators could handle more instances, they were given just the two mentioned instances.

Figures 38 and 39 essentially show that all methods benefit when the similarity of the three instances is high. However, notice that the arrival rate increases significantly especially for the prediction by the similarity-based approach, but also for the ones by the norm-based estimators. Also notice that the similarity-based approach outperforms all norm-based estimators when the similarity is low, which is a reasonable behaviour: for a low similarity, the routes in first intersection of the approximation set as well as the route that maximises the average departure time are too much influenced by the noise of the training instances. The similarity-based approach, however, still can let the approximation sets grow so that more stable solutions are contained. On the other hand, if the similarity is high, then there is so few noise in the data that the similarity is maximised as early as possible, which is reasonable. It also seems that there is only very little impact whether we choose a random route from the intersection, or a route with a maximum number of realisations.

Of course the results of this experiment cannot directly be used for designing an algorithm, since the testing instance is unknown. Nevertheless we believe that the results are interesting because they demonstrate the power of the similarity-based approach.

3.5 Conclusions

The deliverable described a more efficient algorithm for generating all routes, described various approaches for assessing the robustness of journeys and experimentally compared these algorithms on real-world data from Zürich. We established a ranking of the different methods and showed that it does not change substantially when the latest allowed arrival time t_A changes (the probability to arrive on time and the standard deviation on the arrival time change, though). For parametrisable algorithms such as the Buffer-δ or the Mean-Risk-c methods, we observed a clear trade-off between the departure time and the arrival rate. However, finding the right parameter is a highly nontrivial choice. On the other hand, the similarity-based method presented in D3.4 does not need parameter tuning and performs reasonably well when the training and the testing are similar enough. At least for the Zürich network, a high similarity of testing and training instances is not always

	Low Similarity		High Similarity	
	Arrival Rate	Standard deviation	Arrival Rate	Standard deviation
Opt-TT	49.44%	61.44	64.95%	63.39
Buffer-1	55.67%	55.07	73.67%	52.84
Buffer-2	61.66%	52.59	79.30%	49.70
Buffer-3	69.02%	47.41	83.27%	40.09
Buffer-4	74.26%	45.09	86.01%	43.14
Buffer-5	78.30%	43.51	88.56%	40.81
Buffer-6	81.34%	39.38	90.96%	36.84
Buffer-7	84.20%	39.42	92.64%	34.95
Mean-Risk-Only2-1	80.24%	41.12	95.55%	28.78
Similarity-MRR	70.94%	42.18	93.71%	32.05
Similarity-Random	70.93%	42.18	93.71%	32.05
Norm-1-Only2	68.76%	41.74	92.85%	32.01
Norm-2-Only2	70.60%	41.68	93.64%	31.97
Norm-Inf-Only2	70.58%	41.98	93.70%	32.05

Figure 38: Comparison of the arrival rates and the standard deviation of the arrival time (in minutes) for different methods. The latest allowed arrival time t_A was set to 9:00. The second and third column contain the results when the three least similar instances (average mutual maximum similarity 16.79) were used, the forth and the fifth column contain the results when the three most similar instances (average mutual minimum similarity 31.59) were used.

	Low Similarity		High Similarity	
	Arrival Rate	Standard deviation	Arrival Rate	Standard deviation
Opt-TT	37.16%	10.99	52.66%	11.29
Buffer-1	41.83%	10.80	59.98%	11.38
Buffer-2	47.96%	10.45	66.39%	10.98
Buffer-3	55.54%	10.42	72.44%	11.21
Buffer-4	61.33%	10.39	77.10%	11.10
Buffer-5	66.17%	10.23	80.96%	11.46
Buffer-6	71.65%	10.34	85.04%	11.73
Buffer-7	75.46%	10.45	88.50%	11.88
Mean-Risk-Only2-1	79.38%	15.56	94.00%	19.02
Similarity-MRR	68.42%	13.32	91.22%	16.87
Similarity-Random	68.40%	13.32	91.22%	16.87
Norm-1-Only2	64.04%	13.33	90.11%	16.69
Norm-2-Only2	67.66%	13.32	91.24%	17.45
Norm-Inf-Only2	67.79%	13.33	91.22%	17.61

Figure 39: Comparison of the arrival rates and the standard deviation of the arrival time (in minutes) for different methods. The latest allowed arrival time t_A was set to 18:00. The second and third column contain the results when the three least similar instances (average mutual maximum similarity 13.28) were used, the forth and the fifth column contain the results when the three most similar instances (average mutual minimum similarity 28.25) were used.

guaranteed. It would be interesting to compare the Zürich network to other networks to see whether the observed behaviour comes from the method itself, or is an artefact of the data.

All algorithms have a reasonable running time when the Zürich network is used as input. However, we notice that this network is rather small in comparison to networks of other cities. It is not a priori clear whether the running time of the methods scales well. Since our methods require historic delay data as input and such data is not easy to obtain, up to now we did not consider other networks. Nevertheless we think that the fast running time of the methods on the Zürich network indicates that the algorithms can also be applied in real-world scenarios for medium-sized networks other than Zürich.

4 Final assessment of algorithms for context-aware multi-modal daily routes for tourists

4.1 Brief overview of D3.5 algorithmic approaches

The main aim of the D3.5 ("Context-aware multi-modal daily routes for tourists and their empirical assessment") has been the development of models and algorithmic solutions for context-aware multi-modal daily route planning problems for tourists visiting multiple points of interests (POIs), optimized for mobile devices. Those route planning problems, known in the literature as tourist trip design problems (TTDP), involve deriving personalized recommendations for daily sightseeing itineraries for tourists visiting any urban destination.

In particular, a TTDP [33] refers to a route-planning problem for tourists interested in visiting multiple POIs. TTDP solvers derive daily tourist tours, i. e., ordered visits to POIs, which respect tourists' constraints and POIs' attributes. The main objective of the problem discussed is to select POIs that match tourist preferences, thereby maximizing tourist satisfaction ("profit"), while taking into account a multitude of parameters and constraints (e. g., distances among POIs, visiting time required for each POI, POIs visiting days/hours, entrance fees, weather conditions) and respecting the time available for sightseeing in daily basis.

The orienteering problem (OP) [64] serves as the baseline optimization problem for modeling TTDP. The OP seeks for a tour that maximizes the total collected profit while maintaining the travel cost under a given value. Clearly, the OP may be used to model the simplest version of the TTDP wherein the POIs are associated with a profit (i. e., user satisfaction) and the goal is to find a single tour that maximizes the profit collected within a given time budget (time allowed for sightseeing in a single day). Extensions of the OP have been successfully applied to model more complex versions of the single tour TTDP. The OP with Time Windows (OPTW) considers visits to locations within a predefined time window (this allows modeling opening days/hours of POIs). The Time-Dependent OP (TDOP) considers time dependency in the estimation of time required to move from one location to another and therefore, it is suitable for modeling multi-modal transports among POIs. The Team Orienteering Problem (TOP) is the extension of the OP to multiple tours. The TOP with Time Windows (TOPTW) and the Time-Dependent TOPTW (TDTOPTW) have been used to model different versions of the multiple tour TTDP.

In D3.5 we proposed algorithmic approaches that tackled variants of TTDP. First, we considered multiple tours via POIs with specific opening days/hours, assuming constant travel times among POIs (i. e., exclusively walking transfers). Hence, we modeled TTDP as a TOPTW problem and designed two efficient algorithmic solutions to deal with it. Building upon that, we then additionally took into account time-dependent (i. e., multimodal) travel times in our TTDP modeling. To treat time dependency, we modeled TTDP as a TDTOPTW problem and implemented several algorithms dealing with it. All our prototyped algorithms have been evaluated and tested upon both existing and new test instances. We have also used validation scenarios comprising real POI sets compiled from the Athens (Greece) area and calculated multimodal travel times based on the metropolitan transit network of Athens. A more elaborate overview of the D3.5 algorithmic approaches is provided in the sequel.

TOPTW algorithmic approaches. We presented CSCRatio and CSCRoutes, two cluster-based approaches to the TTDP. The main incentive behind these approaches is to favor visits to topology areas featuring high density of good candidate nodes. Furthermore, they both favor solutions with reduced number of long transfers among nodes, which are associated with public transportation transfers in typical urban settings (such transfers are costly, time consuming and usually less attractive to tourists than short walking transfers). The comparison of CSCRatio over the ILS algorithm[3]

[3]ILS (Iterated Local Search) represents a fair compromise in terms of speed versus deriving routes of reasonable quality (on average, less than 5% gap from the best known solution on publicly available test instances). Among the

[65] demonstrated that CSCRatio achieves higher quality solutions in comparable execution time (especially when considering limited itinerary time budget), while also reducing the average number of transfers. As regards the comparison of CSCRoutes over ILS, this confirmed the prevalence of the former in situations where the reduction of inter-cluster transfers is of critical importance. The lower number of transfers in CSCRoutes is achieved at the expense of slightly lower quality solutions. Furthermore, CSCRoutes achieved the best performance results with respect to execution time, compared to ILS and CSCRatio. Notably, the performance gap of the algorithms over ILS increased when tested on realistic TTDP instances, wherein nodes typically feature wide, overlapping time windows and are located nearby each other, while the daily time budget is 5-10h.

TDTOPTW algorithmic approaches. We proposed two cluster-based heuristics (the Time Dependent CSCRoutes (TDCSCRoutes) and the SlackCSCRoutes) for solving the TDTOPTW which make no assumption on periodic service schedules. The main design objectives of the two algorithms are to derive high quality TDTOPTW solutions (maximizing tourist satisfaction), while minimizing the number of transit transfers and executing fast enough to support online web and mobile applications. The prototyped algorithms have been tested in terms of various performance parameters (solutions quality, execution time, number of transit transfers, etc) upon real test instances compiled from the wider area of Athens, Greece. The performance of the algorithms has been compared against two variants that use precalculated average travel times (among the individual time dependent, real travel times) between POIs, the AvgILS and the AvgCSCRoutes. AvgILS refers to the average travel time approach proposed by Garcia et al. [32]. AvgCSCRoutes uses CSCRoutes to construct routes based on pre-computed average travel times. With respect to the overall collected profit, TDCSCRoutes has been shown to perform marginally better. On the other hand, SlackCSCRoutes achieved a fair compromise among all the performance aspects. In practical applications, comprising very large datasets, AvgCSCRoutes could be the most suitable choice as it efficiently derives solutions of reasonably good quality. Nevertheless, its suitability largely depends on the high frequency of public transit services, so that average travel times represent a good guess.

In this deliverable, we build upon the algorithmic approaches summarized above. Our focus has been to investigate increasingly complex TTDP formulations which capture realistic tourist requirements, thereby further advancing the state of the art (among all known research prototypes and commercial tools). The two main threads of our recent research in the framework of D3.6 are based on the following observations:

- Users typically seek stop overs at affordable and conveniently located restaurants along their tours. Existing algorithmic approaches (tour planning software tools) overlook this requirement and derive itineraries exclusively comprising visits to attractions.
- (TD)(T)OP(TW) algorithmic approaches derive tours, essentially comprising successive visits to vertices, i. e., public or supervised sites (e. g., squares, parks, museums, archaeological sites, etc). This approach overlooks the emphasis of tourists on the actual route followed to reach POIs. Most tourists particularly appreciate walking routes via zones of physical/scenic or architectural/historical/cultural value. That is, most visitors would trade a fastest route via a road segment with heavy car traffic for a longer, yet more attractive, route option. Notably, tour planners that integrate "scenic routes" in their tour recommendations simplistically consider the routes as points, modeling the scenic routes walking time similarly to POIs visiting time. This implies that, the user is expected to start/end the scenic route on that particular location within the allocated visiting time in order not to invalidate the remainder of the tour.

The above discussed issues have been addressed in the following ways:

many existing TOPTW solvers, ILS has been known (until recently) to be the most suitable algorithmic approach for TTDP problems, which have strict real-time requirements.

- We have extended our TDTOPTW algorithmic solutions so as to allow users to schedule lunch/coffee breaks through recommending restaurants/cafes based on both their price range and their location so as not to require long detours away from attraction areas.
- To allow modeling scenic routes, i. e., to assign profits to arcs in addition to vertices ("typical" POIs) we have firstly investigated the Arc Orienteering Problem (AOP) and the Mixed Orienteering Problem (MOP). AOP is a single route arc routing problem with profits where arcs are associated with profits and travel times, while MOP is the extension of both the OP and the AOP, where both nodes and arcs are associated with profit. We have obtained approximation algorithms for both the AOP and the MOP, presented in this deliverable. Thereafter, we investigated the Mixed Team Orienteering Problem with Time Windows (MTOPTW), i. e., the extension of the MOP to multiple tours, which has not been studied so far in the literature. Given the hardness of this problem and the real-time requirements of TTDP, we have focused on (meta)heuristic approaches to tackle MTOPTW.

The remainder of this Section is structured as follows: Subsection 4.2 discusses extensions to our TDTOPTW algorithmic solutions to allow incorporating lunch breaks in multimodal tour planning. Subsection 4.3 presents our Approximation algorithms for the AOP and the MOP while Subsection 4.4 presents two metaheuristic algorithms for the MTOPTW and their empirical assessment upon real data related to the city of Athens, Greece. Finally, Subsection 4.5 concludes this Section.

4.2 Incorporating lunch breaks in multimodal tour planning

Tourists commonly chose restaurants on the basis of hard/soft constraints and preferences (e. g., price range, cuisine, customer reviews, etc) and their location since they prefer not to considerably deviate from their sightseeing routes. Given the numerous restaurant options typically in offer, the selection of a suitable place for lunch break may be even more cumbersome than scheduling visits to attractions. However, existing tourist tour planners exclusively consider visits to attractions ignoring the need for lunch/rest breaks.

We address this issue by scheduling lunch breaks in affordable restaurants located nearby the tourist tour. From a tour planning point of view, restaurants may be considered as a separate set of POIs with identical profit; among these POIs it is compulsory to visit only one. The methodology discussed in the sequel has been incorporated in our TDTOPTW SlackCSCRoutes algorithm.

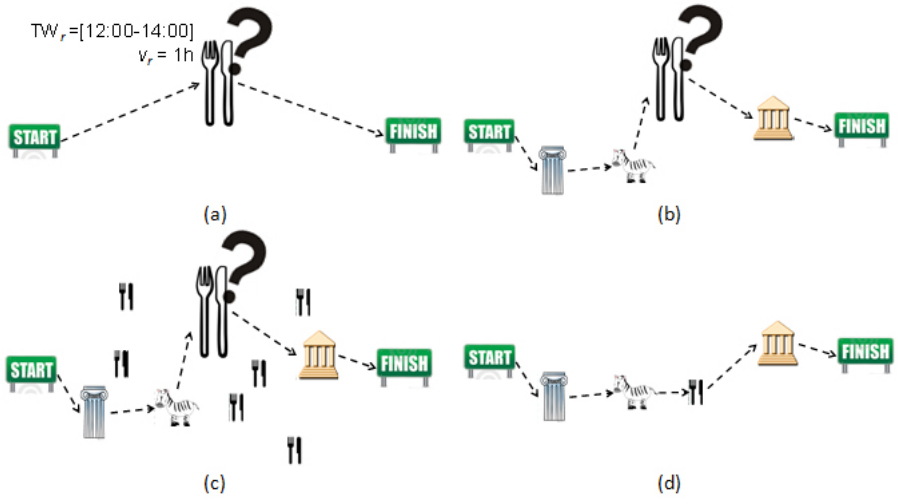

Figure 40: Process of incorporating lunch breaks.

The route creation process starts with inserting a "dummy" node (see Figure 40a) with its time window and visiting time properties provided by the user (the user indicates her/his preferred time span and duration of lunch break). Thereafter, SlackCSCRoutes executes as normal, accommodating visits to attractions prior/after the visit to the restaurant (see Figure 40b). It is noted that the travel cost between the dummy node and an attraction is set equal to the shortest time dependent travel cost between the attraction and a restaurant. At the end of the process, all available restaurants within the specified budget are considered (see Figure 40c). The one requiring the shortest time to reach is chosen to visit at the same position held by the dummy node (see Figure 40d). In the case that the derived route is infeasible, nodes included in the route are iteratively removed (in profit ascending order) until route feasibility is attained. The above discussed lunch scheduling logic has been integrated in the mobile (Android) multimodal tourist tour planning application which has been prototyped in the framework of WP5 of eCOMPASS (see Figure 41).

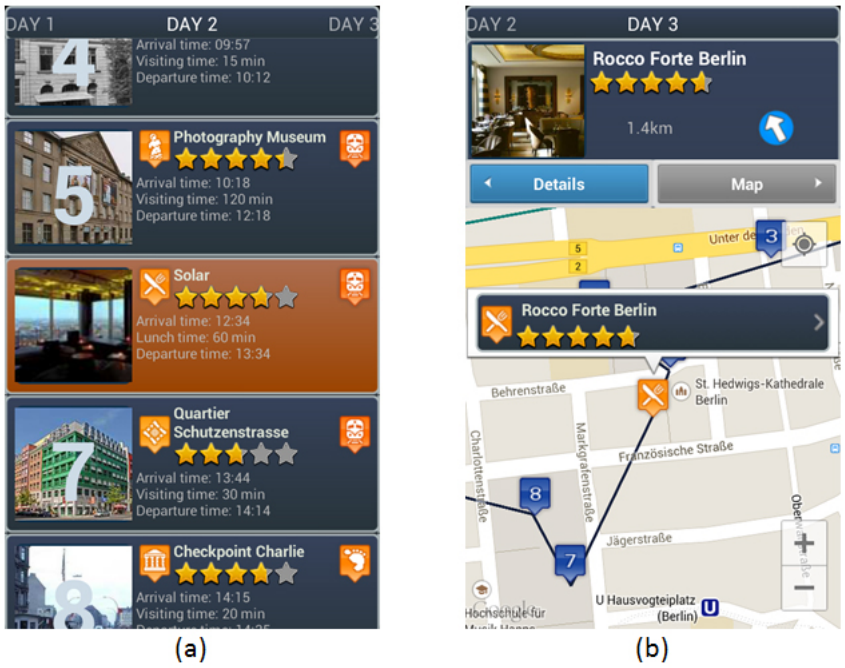

Figure 41: Screenshots taken from the eCOMPASS mobile (Android) multimodal tourist tor planning application: (a) lunch break shown in list view (highlighted with different background color); (b) lunch break shown in map view (indicated by different marker).

4.3 The Arc Orienteering Problem (AOP)

The Arc Orienteering Problem (AOP) is a single route arc routing problem with profits introduced by Souffriau et al. in [62]. Given a directed graph $G = (V, A)$ whose arcs are associated with profits and travel times, two nodes $s, l \in V$, and a time budget B, the problem entails finding an $s - l$ walk of total length at most B so as to maximize the sum of the profits of the arcs visited by the walk. Note that the profit of each arc in the walk is collected only at the first time it is traversed. The AOP is the arc routing version of the Orienteering Problem (OP), an NP-hard problem, named after a sport game called orienteering [39, 63]. In the OP the nodes (instead of the arcs) are associated with profits and the goal is to find a walk from s to l with length at most B such that the total profit of the visited nodes is maximized. The OP as well as its extension to multiple routes, the Team Orienteering Problem (TOP), and many other extensions and variants

have been extensively studied in the literature. In the past decade a significant number of studies have been conducted wherein approximation approaches, metaheuristics and exact methods have been employed to tackle these problems (see [33] and [64]for a survey). One of the most common application of the OP and its extensions is to model different versions of the Tourist Trip Design Problem (TTDP) [66], a route-planning problem which deals with deriving near optimal routes for tourists visiting a destination with several points of interest (POIs) each associated with a profit.

The AOP is applicable to TTDP variants whose modeling requires profits to be associated with the arcs of the network as some links may be more beneficial to be traversed than others. As an example we may consider the derivation of personalized bicycle trips. Based on the biker's personal interests, starting and ending point and the available time budget, a personalized trip can be composed using arcs that better match the cyclist's profile. Similarly, AOP solvers may favor detours via riverside or pedestrian roads against shorter routes via high-traffic or unsafe zones for tourists moving among POIs. The extension of the AOP to multiple routes, introduced by Archetti et al. in [4] and named as Team Orienteering Arc Routing Problem (TOARP), may also find applications to TTDP variants. For example, consider the selection of paths of higher scenic value (among the many available between pairs of POIs) as well as the exclusion of paths including environmentally burdened road segments in favor of longer detours through pedestrian zones.

Although numerous research works concern the OP as well as many extensions and variants of the OP, there is very limited body of literature concerning AOP and TOARP. To the best of our knowledge, this literature includes the work of Souffriau et al. [62] which uses the AOP to model and provide a heuristic solution to the problem of planning cycle trips in the province of East Flanders, the work of Archetti et al. in [4] that proposes a formulation of the problem and a branch-and-cut algorithm and the work of Archetti et al. in [2] which introduces a metaheuristic approach to TOARP.

The combination of the OP and the AOP is proposed in [64] under the name Mixed Orienteering Problem (MOP). In the MOP, profits are associated with the nodes as well as with the arcs of the graph. The problem is very interesting in the context of tourist trip planning as it can be used to formulate TTDP variants where certain routes may be of tourist interest, in addition to attractions. The only relevant research works concern the one-period Bus Touring Problem (BTP) [17], and the Outdoor Activity Tour Suggestion Problem (OATSP) [50].

Herein we first overview related work (Subsection 4.3.1) and then we present approximation algorithms for the AOP in directed and undirected graphs. Specifically, in Subsection 4.3.2 we give an inapproximability result for the AOP and propose a polylogarithmic approximation algorithm for the problem using the polylogarithmic approximation algorithm for the OP in [55]. In Subsection 4.3.3 we present a $(6 + \epsilon + o(1))-$approximation algorithm for the AOP in undirected graphs and a $(4 + \epsilon)-$approximation algorithm for the unweighted version of the problem, using the $(2 + \epsilon)-$approximation algorithm for the unweighted OP in [16]. Finally, we obtain approximation algorithms for the Mixed Orienteering Problem (MOP), the extension of both the OP and the AOP, where both nodes and arcs are associated with profit.

4.3.1 Related work

Souffriau et al. in [62] use the AOP to model and solve the problem of planning cycle trips in the province of East Flanders. Their solution approach is based on a Greedy Randomized Adaptive Search Procedure (GRASP), while experimental results are based on instances generated from the East Flanders network.

Archetti et al. in [4] propose a formulation for the AOP and study a relaxation of its associated polyhedron. Also, they develop a branch-and-cut algorithm for solving the problem. Archetti et al. in [2] propose a metaheuristic approach for the AOP. Experimental results show that the algorithm gives an average percentage error with respect to the optimal solution which is lower than 1%.

The Undirected Capacitated Arc Routing Problem with Profits (UCARPP), the arc routing counterpart of the capacitated TOP, is considered in [3]. In this problem a profit and a nonnegative

demand is associated with each arc and the objective is to determine a path for each available vehicle in order to maximize the total collected profit, without violating the capacity and time limit constraints of each vehicle. The authors consider an application where carriers can select potential customers for transporting their goods. Another potential application is the creation of personalized bicycle trips. An exact approach for solving the UCARPP along with several heuristics were proposed in [3]. The problem was also studied by Zachariadis and Kiranoudis in [67] where a local search procedure was given.

To the best of our knowledge, the only research works relevant to the MOP, concern the one-period Bus Touring Problem (BTP) [17], and the Outdoor Activity Tour Suggestion Problem (OATSP) [50]. In the BTP the objective is to maximize the total profit of the tour by selecting a subset of nodes to be visited and arcs to be traveled both having associated profits, given a constraint on the total touring time. The profit of recurrently visited nodes and arcs is only counted once. In [17] a heuristic approach is employed to solve the BTP. The OATSP, introduced recently by Maervoet et al. [50], involves finding attractive closed paths in a transportation network, tailored for a specific outdoor activity mode such as hiking and mountain biking. Total path attractiveness is evaluated as the sum of the average arc attractiveness and the profits of the nodes along the path. The problem involves finding a closed path of maximal attractiveness given a target path length and tolerance. That is, the OATSP requires a target path length instead of a maximal travel time required by the BTP. This gives rise to a path length window constraint. In [50] an efficient heuristic solution to the OATSP is presented.

To the best of our knowledge, no approximation algorithms for the AOP or the MOP have been presented in the literature. On the other hand, there is a significant number of research works on the approximability of the OP. As mentioned in the introduction, OP is NP-hard ([39], [46]) and it also known to be APX-hard [9]. The basic idea for approximating the OP was presented by Blum et al. in [9], [10] where the min-excess $s-t$ path problem (given two nodes s, t and an integer k, find an $s-t$ path of minimum-excess [4] that visits at least k nodes) was defined. It was shown that an approximation for the min-excess path problem implies an approximation for the OP. Then, the min-excess path problem can be approximated using algorithms for the k-stroll problem (find a minimum length $s-t$ walk that visits at least k nodes). Blum et al. obtained a 4-approximation algorithm for the OP in undirected graphs by using a $(2+\epsilon)$-approximation for the k-stroll path problem. In fact, most subsequent approximation algorithms for OP follow the framework of [9], [10] which reduces the OP to the k-stroll problem via the min-excess path problem. The best known approximation algorithm for the OP in undirected graphs is due to Chekuri et al. [15] who obtained a $(2+\epsilon)$-approximation algorithm with running time $n^{O(1/\epsilon^2)}$ by giving a bi-criteria approximation for k-stroll problem with respect to the path length and the number of nodes visited. Using the same approach, they also obtained an $O(\log^2 OPT)$ approximation algorithm for the OP in directed graphs, where OPT denotes the number of nodes in an optimal solution. The best known approximation algorithm for the OP in directed graphs is due to Nagarajan and Ravi [55]. They gave an $O(\frac{\log^2 n}{\log\log n})$-approximation algorithm for the OP in directed graphs employing a bi-criteria approximation solution for k-stroll based on an LP approach.

4.3.2 Approximation algorithms for the AOP

In this Subsection we first obtain an inapproximability result for the AOP. Then, we propose an approximation algorithm for the problem.

The OP in directed graphs is reduced to the AOP. Given an OP instance an AOP instance is created as follows: Assign zero profit to each arc and for each node u add a node u', the arcs $(u,u'),(u',u)$ of zero travel cost and $\frac{p_u}{2}$ profit and remove u's profit. Then, an OP solution yields an equal length and at least the same profit AOP solution and vice versa. Using the results in [39] and [10], we get the following theorem.

[4] The excess of an $s-t$ path is the difference of the path length from the length of the shortest $s-t$ path.

Theorem 1. *The AOP is NP-hard and hard to approximate within* $\frac{1481}{1480}$.

Theorem 2. *An* $f(n)-$*approximation algorithm for the OP in directed graphs, where* n *is the number of nodes, yields an* $f(m+2)-$*approximation algorithm for the AOP, where* m *is the number of arcs.*

Proof. Given an instance of the AOP $(G=(V,A),t,p,B)$, $|A|=m$, we construct an instance of the OP in the directed network $N=(V',A')$ with $V'=\{s,l\}\cup\{(u,v):(u,v)\in A\}$, $|V'|=n'=m+2$, and $A'=\{(s,(s,u)),((v,l),l):(s,u),(v,l)\in A\}\cup\{((u,v),(v,w)):(u,v),(v,w)\in A\}$. The travel times of the arcs in N are defined as follows: $t'_{((u,v),(v,w))}=\frac{t_{(u,v)}+t_{(v,w)}}{2}$, $t'_{(s,(s,u))}=\frac{t_{(s,u)}}{2}$ and $t'_{((v,l),l)}=\frac{t_{(v,l)}}{2}$. The profit of each node (u,v) equals to $p'_{(u,v)}=p_{(u,v)}$ and $B'=B$. Then, a solution of the AOP instance yields a solution of the OP instance of equal total profit and length and vice versa. □

The theorem yields a $O(\frac{\log^2 m}{\log\log m})-$approximation algorithm for the AOP, applying Nagarajan and Ravi's algorithm [55] to the metric closure of the constructed OP instance and transforming the solution to an AOP solution.

4.3.3 Approximation Algorithms for the AOP in Undirected Graphs

In this Subsection we study the AOP in undirected graphs. Similarly to the previous section the problem is NP-hard and hard to approximate within $\frac{1481}{1480}$ via a reduction from the OP in undirected graphs. We obtain a constant factor approximation algorithm for the problem by reducing it to the Unweighted OP (UOP) in undirected graphs, the restriction of the OP with nodes of unit profit. First, we reduce the AOP to the special case with polynomially bounded positive integer profits using a similar technique with [16, 45].

Lemma 3. *A* $\rho-$*approximation algorithm for the AOP in undirected graphs with polynomially bounded positive integer profits yields a* $(\rho+o(1))-$ *approximation algorithm for the AOP in undirected graphs.*

Proof. Given an AOP instance $I=(G=(V,E),t,p,B)$, we construct an instance $I'=(G'=(V',E'),t,p',B)$ with polynomially bounded positive integer profits. First, we guess the edge of highest profit ($p_{\max}$) in the optimal walk and remove all higher profit edges. Then, we set $p'_e=\lfloor\frac{n^3p_e}{p_{\max}}\rfloor+1$ for each edge e. A feasible walk W, consisting of the distinct edges $e_1,e_2,\ldots,e_k$ has profit $\text{profit}(W)=\sum_{j=1}^{k}p_{e_j}$ in I and $\text{profit}'(W)=\sum_{j=1}^{k}p'_{e_j}>\frac{n^3\text{profit(W)}}{p_{\max}}$ in I'. Hence, $\text{OPT}'>\frac{n^3}{p_{\max}}\text{OPT}$, where $\text{OPT}(\text{OPT}')$ is the optimum in $I(I')$. On the other hand, $\text{profit}'(W)=\sum_{j=1}^{k}p'_{e_j}\leq\sum_{j=1}^{k}(\frac{n^3p_{e_j}}{p_{\max}}+1)\implies\frac{n^3}{p_{\max}}\text{profit}(W)\geq\text{profit}'(W)-m\geq\text{profit}'(W)-m\frac{\text{OPT}}{p_{\max}}$, so $\text{profit}(W)\geq\frac{p_{\max}}{n^3}\text{profit}'(W)-\frac{m}{n^3}\text{OPT}$. Then, a $\rho-$approximation solution W of I' has $\text{profit}'(W)\geq\frac{\text{OPT}'}{\rho}$, so $\text{profit}(W)\geq\frac{1}{\rho}\frac{p_{\max}}{n^3}\text{OPT}'-\frac{m}{n^3}\text{OPT}>(\frac{1}{\rho}-\frac{m}{n^3})\text{OPT}$. □

Theorem 4. *A* ρ*-approximation algorithm for the UOP in undirected graphs yields a* 3ρ*-approximation algorithm for the AOP in undirected graphs with polynomially bounded positive integer profits.*

Proof. Given an AOP instance, each edge e such that the shortest path from s to l passing through e exceeds the time budget is removed from the graph. Then, we construct an UOP instance by splitting each edge $\{u,v\}$ into $p_{uv}+1$ edges as follows: Each node of the AOP instance is a node of the UOP instance (basic node) and for each edge $\{u,v\}$ of the AOP instance, the UOP instance includes the auxiliary nodes $\{u,v\}_1,\{u,v\}_2,\cdots,\{u,v\}_{p_{uv}}$ and the edges $\{u,\{u,v\}_1\}$, $\{\{u,v\}_1,\{u,v\}_2\}$, $\cdots$,

$\{\{u,v\}_{p_{uv}-1},\{u,v\}_{p_{uv}}\}$, $\{\{u,v\}_{p_{uv}},v\}$. The travel times of $\{u,\{u,v\}_1\}$ and $\{\{u,v\}_{p_{uv}},v\}$ are set to $\frac{t_{\{u,v\}}}{2}$, while the travel time of the remaining edges is zero. The time budget of the UOP instance is set equal to the time budget of the AOP instance.

An AOP solution yields an equal length UOP solution of at least the same profit by replacing each edge $\{u,v\}$ by the segment $(u,\{u,v\}_1,\cdots,\{u,v\}_{p_{uv}},v)$. On the other hand, we will show that any UOP solution yields an AOP solution of at least a third of the former's profit ($p_{\text{AOP}} \geq \frac{p_{\text{UOP}}}{3}$). A sequence of nodes $(u,\{u,v\}_1,\{u,v\}_2,\cdots,\{u,v\}_{p_{uv}},v)$ is called an *appropriate* segment, i. e., a segment representing the traversal of the edge $\{u,v\}$ in the AOP instance, while $(u,\{u,v\}_1,\cdots,\{u,v\}_{i-1},\{u,v\}_i,\{u,v\}_{i-1}\cdots,\{u,v\}_1,u)$ is called an *inappropriate* segment, i. e., a segment representing the partial traversal of the edge $\{u,v\}$ of the AOP instance. For each inappropriate segment we consider that $i = p_{uv}$, otherwise we may extend it to the equal length and higher profit segment with $i = p_{uv}$.

In an UOP solution, if $p_{\text{AS}}(p_{\text{IS}})$ is the profit gained by the appropriate (resp., inappropriate) segments, then $p_{\text{UOP}} = p_{\text{AS}} + p_{\text{IS}}$. Note that p_{AS} equals to the number of visited basic nodes plus the number of auxiliary nodes visited in the appropriate segments while p_{IS} equals to the number of auxiliary nodes visited in the inappropriate segments, since any basic node visited in an inappropriate segment has already been counted in p_{AS}. If all segments are appropriate, an AOP solution is obtained replacing the segments by their representing edges. This yields an AOP solution of $p_{\text{AOP}} > \frac{p_{\text{UOP}}}{2}$, since p_{UOP} equals to p_{AOP} plus the number of visited basic nodes, apart from s, l. Hence, p_{UOP} is less than p_{AOP} plus the number of traversed edges, i. e., $p_{\text{UOP}} < 2p_{\text{AOP}}$.

If however inappropriate segments exist, let $\text{IS} = \{s_1, s_2, \ldots, s_k\}$ be the set of inappropriate segments of the UOP solution. Let also $p_1, p_2, \ldots, p_k$ be the profits collected by traversing them ($\sum_{i=1}^{k} p_i = p_{\text{IS}}$), assuming w.l.o.g that $p_1 \geq p_2 \geq \cdots \geq p_k$ and $t_1, t_2, \ldots, t_k$ be the travel times associated with them. If $p_1 \geq \frac{p_{\text{UOP}}}{3}$, then an AOP solution of $p_{\text{AOP}} \geq \frac{p_{\text{UOP}}}{3}$ is obtained returning the shortest path from s to l passing through the edge represented by s_1.

Otherwise, we shall replace some of the inappropriate segments by their representing edge traversed in both directions, i. e., the inappropriate segment

$$(u,\{u,v\}_1,\cdots,\{u,v\}_{p_{uv}-1},\{u,v\}_{p_{uv}},\{u,v\}_{p_{uv}-1},\cdots,\{u,v\}_1,u)$$

shall be replaced by the sequence of nodes (u,v,u) in the AOP instance. A subset RS of IS, with $\sum_{s_j \in \text{RS}} 2t_j \leq \sum_{i=1}^{k} t_i = \sum_{s_j \in \text{IS}} t_j$ will be called a *replaceable subset*, i. e., replacing the appropriate segments by their representing edges and the segments in the replaceable set by their representing edges traversed in both directions, we obtain a feasible AOP solution. RS is a *maximal replaceable subset*, if inserting a segment ($s_m \notin \text{RS}$) into the set would violate the time constraint, i. e., $\sum_{s_j \in \text{RS}} 2t_j + 2t_m > \sum_{i=1}^{k} t_i$. Consider a maximal replaceable subset MRS of segments. We distinguish between the following cases: (i) $p_{\text{MRS}} \geq \frac{p_{\text{IS}}}{3}$ or $p_{\text{AS}} + p_{\text{MRS}} \geq \frac{2p_{\text{UOP}}}{3}$, where p_{MRS} is the total profit of segments in MRS. Then an AOP solution is obtained consisting of the edges represented by the appropriate segments (contributing at least $\frac{p_{\text{AS}}}{2}$ profit) and the sequences that replace the segments in MRS (contributing p_{MRS} profit), with profit at least $\frac{p_{\text{AS}}+p_{\text{IS}}}{3} = \frac{p_{\text{UOP}}}{3}$ or greater than $\frac{p_{\text{AS}}+p_{\text{MRS}}}{2} \geq \frac{p_{\text{UOP}}}{3}$, hence $p_{\text{AOP}} \geq \frac{p_{\text{UOP}}}{3}$. (ii) The set of inappropriate segments $\text{MRS}^c = \text{IS}\backslash\text{MRS}$ has $p_{\text{MRS}^c} > \frac{p_{\text{UOP}}}{3}$. Note that MRS^c contains at least two segments, otherwise $p_1 \geq p_{\text{MRS}^c} > \frac{p_{\text{UOP}}}{3}$, a contradiction. Furthermore, $p_{\text{MRS}^c} > \frac{2p_{\text{IS}}}{3}$, hence removing the lowest profit segment $s_m \in \text{MRS}^c$, $\text{MRS}^c\backslash\{s_m\}$ has profit $p_{\text{MRS}^c\backslash\{s_m\}} \geq \frac{p_{\text{MRS}^c}}{2} > \frac{p_{\text{IS}}}{3}$. Since $\sum_{s_j \in \text{MRS}} 2t_j + 2t_m > \sum_{i=1}^{k} t_i$ then $\sum_{s_j \in \text{MRS}^c\backslash\{s_m\}} 2t_j < \sum_{i=1}^{k} t_i$, hence $\text{MRS}^c\backslash\{s_m\}$ is replaceable. Then, we apply the same technique with (i) using $\text{MRS}^c\backslash\{s_m\}$ instead of MRS, obtaining an AOP solution of $p_{\text{AOP}} > \frac{p_{\text{UOP}}}{3}$. □

Using Lemma 3, Theorem 4 and the $(2+\epsilon)-$approximation algorithm for the UOP by Chekuri et al. [16] we obtain a $(6+\epsilon+o(1))-$approximation algorithm for the AOP in undirected graphs with execution time $n^{O(\frac{1}{\epsilon^2})}$.

The unweighted AOP (**UAOP**) is the special case of the AOP with edges of unit profit. Similarly to Theorem 4, a $\rho-$approximation algorithm for the UOP in undirected graphs yields a $2\rho-$approximation algorithm for the UAOP in undirected graphs. If the optimal UAOP solution contains less than $\rho+2$ nodes, it is found by exhaustive search since ρ is a constant. Otherwise, $\mathrm{OPT_{UOP}} \geq \mathrm{OPT_{UAOP}} + \rho$. Then, an UOP solution of $p_{\mathrm{UOP}} \geq \frac{\mathrm{OPT_{UOP}}}{\rho}$ yields an UAOP solution of $p_{\mathrm{UAOP}} \geq \frac{p_{\mathrm{UOP}}-1}{2} \geq \frac{\mathrm{OPT_{UAOP}}}{2\rho}$ by replacing its appropriate and its half shortest inappropriate segments. Using Chekuri et al.'s algorithm [16] we obtain a $(4+\epsilon)-$approximation algorithm.

The Mixed Orienteering Problem (**MOP**) [17, 64] extends both the OP and the AOP, assigning profit to both nodes and arcs. The MOP is reduced to the AOP in a similar way with the reduction from the OP to the AOP, retaining though the arcs' profit. As a result, any approximation algorithm for the AOP yields an approximation algorithm for the MOP.

4.4 The Mixed Team Orienteering Problem with Time Windows (MTOPTW)

In this Subsection we introduce the Mixed Team Orienteering Problem with Time Windows (MTOPTW), i. e., the extension of the MOP to multiple tours, and present the first algorithmic approaches to tackle it. The problem can be used to formulate realistic TTDP variants whose modeling requires multiple tourist tours, profits to be associated to both POIs (nodes of the network) and routes (arcs of the network) as certain routes may be more interesting to be traversed than others, while both POIs and routes are associated with visiting/traversing time windows. We define the problem on windy graphs as follows: Consider a complete windy undirected graph $G=(V,E)$, where $V=\{u_1,u_2,\ldots,u_N\}$ denotes the vertex set and E the edge set. A travel cost is assigned to each link between two vertices, i. e., each ordered pair of vertices (u,v) has a travel cost $T_{u,v}$ which might be different from $T_{v,u}$. Each vertex u is associated with a visit duration T_u, while visiting a vertex u (or traversing an edge $\{u,v\}$) offers a profit P_u (respectively, $P_{u,v}$). Each vertex u (edge $\{u,v\}$) is associated with an opening time O_u^d (respectively, $O_{u,v}^d$) and a closing time C_u^d (respectively, $C_{u,v}^d$) for each different day of the week $d\in\{0,1,\ldots,6\}$, The visit at a vertex u (or the traversal of an edge $\{u,v\}$) at a specific day d can only start after its opening time and end before its closing time. Furthermore, an integer K is given denoting the number of the walks that will be constructed, and for each walk W_i a starting vertex sl_i and an ending vertex el_i are given, $i=0,1,\ldots,K-1$ as well as a starting time st_i and an ending time $et_i, i=0,1,\ldots,K-1$.

A feasible solution of the MTOPTW consists of K walks $W_0,W_1,\ldots,W_{K-1}$ with $W_i=(w_0^i,w_1^i,\ldots,w_{l_i-1}^i)$ such that $w_0^i=sl_i$, $w_{l_i-1}^i=el_i$, the arrival time at sl_i equals to st_i, the arrival time at el_i is at most et_i, and the visit at each vertex w_m^i (the traversal of each edge $\{w_m^i,w_{m+1}^i\}$) satisfies its time window, i. e., the visit (traversal) starts after its opening time and ends before closing time. The profit of the solution is equal to the sum of the profits of the visited vertices and the traversed edges. If a vertex is visited (or an edge is traversed) more than once, its profit is counted only once. The goal of the MTOPTW is to construct the feasible solution of the highest profit.

Without loss of generality we may assume that each vertex u with $P_u>0$ is not connected with an edge of positive profit, i. e., there is no v such that $P_{u,v}>0$, otherwise we introduce a dummy vertex u', that is a clone of u, with $P_{u'}=P_u$ and the same attributes with u (i. e., the same time windows and costs for it and its adjacent edges) and remove the profit from u. Similarly, we may assume that for each edge $\{u,v\}$ with $P_{u,v}>0$ there is no node w such that $P_{u,w}>0$ or $P_{w,v}>0$. Based on these assumptions, we may only consider the feasible solutions that are sequences of profitable (i. e., with positive profit) nodes and edges. We will denote a profitable node and a profitable edge as a *profitable piece*. Then, we may represent a walk of the solution as $W_i=(p_0^i,p_1^i,\ldots,p_{m_i-1}^i)$, with $p_0^i=sl_i, p_{m_i-1}^i=el_i$ and $p_j^i, j=1,2,\ldots,m_i-2$ the included profitable pieces in walk W_i. We shall

denote this representation of the walk as the *piece representation*, while the representation of the walk as a sequence of nodes shall be denoted as the *node representation.* For example the walk W_i in Figure 42 consists of the starting/ending locations as well as the nodes u, v and w from which only u has a positive profit. Apart from u, the edge (v, w) is also associated with profit, hence the profitable pieces of W_i are the node u and the edge (v, w). Therefore, the node representation of the walk is $W_i = (sl_i, u, v, w, el_i)$ while its piece representation is $W_i = (p_0^i, p_1^i, p_2^i, p_3^i)$ with $p_1^i = u$ and $p_2^i = (v, w)$.

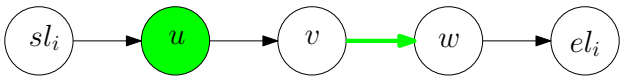

Figure 42: Illustration of the representation of a walk. Profitable pieces are colored green.

In this Subsection we propose two metaheuristic algorithms for the MTOPTW. Specifically, in 4.4.1 an Iterated Local Search metaheuristic, similar to the one presented by Vansteenwegen et al. [65] for the TOPTW, is given. In 4.4.2 we present a Simulated Annealing metaheuristic for the problem. The experimental results compiled from executing the two algorithms are given in 4.4.3.

4.4.1 Iterated Local Search Metaheuristic for the MTOPTW

The Iterated Local Search [11, 49] is a metaheuristic method widely used in combinatorial optimization problems in order to search the solution space extensively. The intuition of this metaheuristic is to iteratively reach a local optimum solution by applying local search and then perturb the solution. The solution is perturbed in order to escape from the local optimum and reach a different local optimum in the next iteration. In this way a lot of local optimum solutions are found, hence the method is expected to produce better results than a simple local search method.

The neighborhood structure used in the inner local search procedure of the presented method is the **InsertPiece**. Considering a feasible solution of an instance of the MTOPTW, a neighboring solution is obtained by inserting a non-included profitable piece between two consecutive nodes of the walk. Equivalently, the **InsertPiece** neighborhood contains all the solutions that are obtained by inserting a non-included profitable piece between two consecutive nodes of the walk that are not connected with a profitable edge.

For example in Figure 43 we consider the neighboring solutions of the single walk solution with node representation $W_i = (sl_i, u, v, w, el_i)$. We consider that there are only two non-included profitable pieces, the edge $\{y, z\}$ and the node x. Figures 43(a) – 43(f) show the neighboring solutions that are produced by inserting the edge $\{y, z\}$ between two consecutive nodes of the walk. Figures 43(a) and 43(b) depict the two solutions obtained by inserting $\{y, z\}$ between sl_i and u, the former with direction from y to z and the latter from z to y. Similarly, Figures 43(c) and 43(d) depict the insertion of $\{y, z\}$ between u and $\{v, w\}$, while Figures 43(e) and 43(f) present the solution obtained by inserting $\{y, z\}$ after $\{v, w\}$. As far as the non-included node x is considered, the neighboring solutions are depicted in Figures 43(g), 43(h) and 43(i), where the insertion after sl_i, u and $\{v, w\}$ is considered, respectively. Note, that neither $\{y, z\}$ nor x can be inserted between v and w, since $\{v, w\}$ is a profitable edge.

Inspired by Vansteenwegen et al.'s [65] article, each included node in a walk of the solution is associated with its arrival (arrive), starting (start) and leaving (leave) time, as well as the maximum time the arrival at the node can take place (maxArrive), such that the walk remains feasible. Considering the walk $W_i = (w_0^i, w_1^i, \ldots, w_{l_i-1}^i)$ in its node representation, that takes place in day d, the time attributes of each included node are given by the recursive formulas:

$\text{arrive}(w_0^i) = \text{start}(w_0^i) = \text{leave}(w_0^i) = st_i$ and for each $k = 0, 1, \ldots, l_i - 2$
$\text{arrive}(w_{k+1}^i) = \max(\text{leave}(w_k^i), O^d_{w_k^i, w_{k+1}^i}) + T_{w_k^i, w_{k+1}}$,
$\text{start}(w_{k+1}^i) = \max(\text{arrive}(w_{k+1}^i), O^d_{w_{k+1}^i})$ and $\text{leave}(w_{k+1}^i) = \text{start}(w_{k+1}^i) + T_{w_{k+1}^i}$.

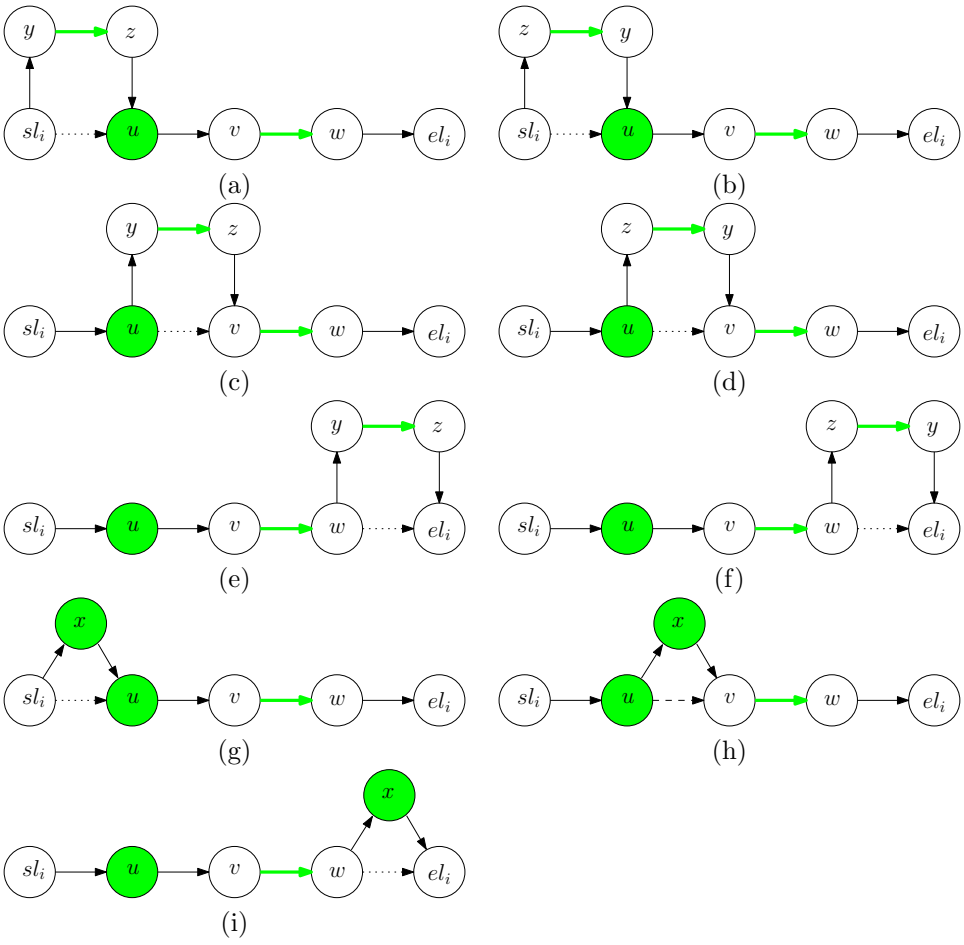

Figure 43: Illustration of the **InsertPiece** Neighborhood

The maxArrive time of the nodes is calculated recursively from the ending location to the start as follows:

$\text{maxArrive}(w^i_{l_i-1}) = et_i$ and for each $k = l_i - 2, l_i - 3, \ldots, 1, 0$
$\text{maxArrive}(w^i_k) = \min(C^d_{w^i_k}, C^d_{w^i_k,w^i_{k+1}} - T_{w^i_k}, \text{maxArrive}(w^i_{k+1}) - T_{w^i_k} - T_{w^i_k,w_{k+1}})$.

Using the previously introduced time attributes of the included nodes, checking if the insertion of a non-included profitable piece after an included piece (inclPiece) in a walk W_i is feasible requires constant time, i. e., time independent of the size of the walk. To see this, consider that the included piece's last node is w^i_k, i. e., the included piece is either the node w^i_k or the edge (w^i_{k-1}, w^i_k) and that the walk takes place at day d. In case that the candidate non-included profitable piece is the edge $\{y, z\}$, we have to examine both directions, i. e., from y to z and from z to y. The insertion of $\{y, z\}$ with the direction from y to z after an included piece with last node w^i_k (see Figure 44(a)) is feasible if and only if the following conditions are satisfied:

1. $\text{leave}(w^i_k) \leq C^d_{w^i_k,y}$
2. the arrival time at y is at most C^d_y
3. the leaving time from y is at most $C^d_{y,z}$
4. the arrival time at z is at most C^d_z
5. the leaving time from z is at most $C^d_{z,w^i_{k+1}}$
6. the new arrival time at w^i_{k+1} is at most $\text{maxArrive}(w^i_{k+1})$

If the insertion is feasible, the difference between the new arrival time at w^i_{k+1} and the former one will be considered as the time shifted and is denoted as shift((y, z),inclPiece).

The pseudo code of a procedure that checks the feasibility of the insertion of the edge (y, z) after an included piece (inclPiece) with last node w^i_k follows (Algorithm 2).

In a similar way, we may check if the insertion of a non-included profitable node x is feasible after an included piece, just by skipping the time window feasibility of the second node, i. e., the node x can be inserted after w^i_k (see Figure 44(b)) if and only if the following conditions are satisfied:

1. $\text{leave}(w^i_k) \leq C^d_{w^i_k,x}$
2. the arrival time at x is at most C^d_x
3. the leaving time from x is at most $C^d_{x,w^i_{k+1}}$
4. the new arrival time at w^i_{k+1} is at most $\text{maxArrive}(w^i_{k+1})$

Similarly to the case of a candidate profitable edge, shift(x,inclPiece) denotes the difference between the new arrival time at w^i_{k+1} and the former one.

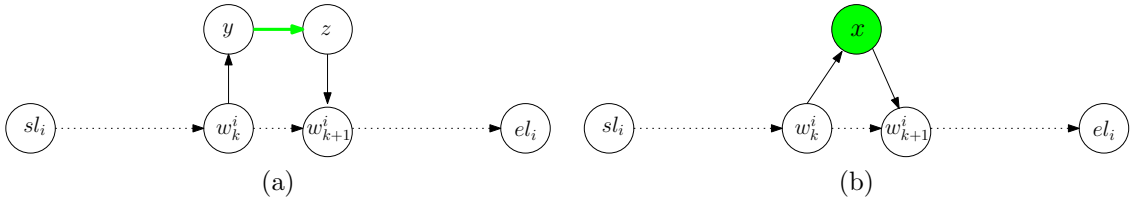

Figure 44: Illustration of insertion's feasibility

Algorithm 2: Insertion Feasibility of the edge (y, z) after inclPiece with last node w_k^i

feasibleInsertion $\leftarrow$ false; shift $\leftarrow 0$
if *leave*$(w_k^i) > C^d_{w_k^i,y}$ **then**
  return feasibleInsertion;shift
end
firstArrive $\leftarrow$ max(leave(w_k^i),$O^d_{w_k^i,y}) + T_{w_k^i,y}$
if *firstArrive* $> C^d_y$ **then**
  return feasibleInsertion;shift
end
firstLeave $\leftarrow$ max(firstArrive,$O^d_y) + T_y$
if *firstLeave* $> C^d_{y,z}$ **then**
  return feasibleInsertion;shift
end
secondArrive $\leftarrow$ max(firstLeave,$O^d_{y,z}) + T_{y,z}$
if *secondArrive* $> C^d_z$ **then**
  return feasibleInsertion;shift
end
secondLeave $\leftarrow$ max(secondArrive,$O^d_z) + T_z$
if *secondLeave* $> C^d_{z,w^i_{k+1}}$ **then**
  return feasibleInsertion;shift
end
newArrive $\leftarrow$ max(secondLeave,$O^d_{z,w^i_{k+1}}) + T_{z,w^i_{k+1}}$
if *newArrive* $\leq$ *maxArrive*(w^i_{k+1}) **then**
  feasibleInsertion $\leftarrow$ true
  shift $\leftarrow$ newArrive $-$ arrive(w^i_{k+1})
end
return feasibleInsertion;shift

The local search move applied by the ILS algorithm is the **insertBestPieceIn** method. This method takes as a parameter the id of a walk (walkId). Then, the "best" non-included profitable piece is inserted in walk with id walkId, in the position of the lowest shift. If pieces with shift ≤ 0 exist, then we consider as the "best" non-included profitable piece the one with the highest profit. Otherwise, we consider as the "best" non-included profitable piece the one with the highest ratio $\frac{\text{profit}}{\text{shift}}$. In more detail, the method considers all candidate non-included profitable pieces. For each candidate piece, every possible insert position is examined, i. e., the insertion between every consecutive pair of included pieces, and the position of the lowest shift is stored. When all candidate pieces and possible insertion positions have been examined, the best insertion takes place as follows: if there exist some candidate piece with shift ≤ 0, then the candidate piece of highest score and of lowest shift ≤ 0 is inserted in its best position and the time attributes of all inserted nodes are recalculated. Otherwise, if the insertion of a candidate piece was feasible, then the node of the highest ratio $\frac{\text{profit}}{\text{shift}}$ is inserted and the time attributes are updated. The pseudocode of the **insertBestPieceIn** method is listed below (Algorithm 3).

Algorithm 3: insertBestPieceIn (walkId)

```
1  for each non-included candidate profitable piece candPiece do
2  |  tempBestShift ← ∞
3  |  tempBestIncluded ← ∅
4  |  for each included piece incPiece in walk with id walkId do
5  |  |  if the insertion of candPiece after incPiece is feasible then
6  |  |  |  if shift(candPiece, incPiece) < tempBestShift then
7  |  |  |  |  tempBestShift ← shift(candPiece, incPiece)
8  |  |  |  |  tempBestIncluded ← incPiece
9  |  |  |  end
10 |  |  end
11 |  end
12 end
13 if ∃ candidate profitable piece with tempBestShift ≤ 0 then
14 |  insert the candidate piece of the highest profit after its tempBestIncluded piece
15 |  Update the times of all the nodes in walk with id walkId
16 else
17 |  if ∃ candidate profitable piece with tempBestShift < ∞ then
18 |  |  insert the candidate with the highest profit/tempBestShift after its tempBestIncluded piece
19 |  |  Update the times of all the nodes in walk with id walkId
20 |  end
21 end
```

The ILS algorithm escapes from the current local optimum applying the **perturb** method. In this method a randomly selected chain of consecutive pieces is removed from each walk of the solution. In a more detailed analysis, for each walk of the solution the number of pieces that will be removed (numberOfRemoved) is chosen randomly. Then, the starting position of the removed pieces (startRem) is selected randomly in the range $[1, m_i - 1 - \text{numberOfRemoved}]$ where m_i denotes the number of pieces included in the walk. Finally, the numberOfRemoved consecutive pieces starting from the position startRem are removed from the walk and the times associated to all the nodes of the walk are recalculated. The pseudocode of the method follows (Algorithm 4).

The ILS algorithm loops for a number of iterations (numberOfIterations) that is given as a parameter. Inside the loop, the list walkIdsToInsert is initialized containing all the ids of the constructed walks, i. e., the ids in the range $[0, K - 1]$. Then, an inner loop is executed as long as the walkIdsToInsert is not empty. Inside the inner loop, each walk with id in the walkIdsToInsert is

Algorithm 4: perturb

```
for each walk W_i do
    m_i ← the number of included profitable pieces in W_i
    numberOfRemoved ← a random number in the range [0, m_i]
    startRem ← a number randomly created in the range [1, m_i − 1 − numberOfRemoved]
    endRem ← startRem + numberOfRemoved −1
    remove all included profitable pieces from startRem (p^i_startRem) to endRem (p^i_endRem)
    update the times of all the nodes in W_i
end
```

considered and the best insertion for this walk is applied. If no insertion was feasible, then the id of this walk is removed from the walkIdsToInsert. When the inner loop is over, the current solution is considered. If its profit is the largest found so far, the current solution becomes the best found and its profit becomes the best profit (bestProfit) found during the method. The last step inside the loop is the perturb step. When the first loop is over, the best found solution as well as the best found profit are returned. The pseudocode of the ILS algorithm follows (Algorithm 5).

Algorithm 5: ILS

```
for numberOfIterations iterations do
    walkIdsToInsert ← [0, 1, ..., K − 1]
    iteratorOfWalkIdsToInsert ← iterator of walkIdsToInsert
    while walkIdsToInsert is not empty do
        walkId ← the value of iteratorOfWalkIdsToInsert
        insertBestPieceIn(walkId)
        if insertion did not happen then
            remove walkId from walkIdsToInsert
        end
        iteratorOfWalkIdsToInsert ← next of iteratorOfWalkIdsToInsert
    end
    if profit > bestProfit then
        bestProfit ← profit
        bestSolution ← solution
    end
    perturb()
end
return bestSolution;bestProfit
```

4.4.2 A Simulated Annealing Metaheuristic for the MTOPTW

The Simulated Annealing [44, 14, 11] is a metaheuristic method that escapes from a local optimum by allowing moves that result in inferior solutions. An initial local optimum solution is usually constructed. Then, moves that result in inferior solutions are allowed with a probability that is high in the early stages of the algorithm, in order to search the space in more depth. This probability is reduced along the execution of the method until it becomes negligible in order to allow only improving solutions and find new (possibly better) local optima. In the general scheme of the Simulated Annealing an auxiliary parameter is used, the temperature (T). The likelihood of inferior resulting moves is usually a function inversely proportional to T and proportional to the decrease of the solution's value, i. e., it may be $\exp(\frac{\Delta P}{T})$, where ΔP is the difference of the value of the resulting

solution from the initial. The temperature is initially set to the maximum temperature which is high enough to allow moves to worse solutions with high probability, and is decreased after a number of iterations. The temperature is usually decreased, multiplied by a cooling factor after a number (coolingIterations) of iterations. In order to add diversification, we may allow a lot of schemes, i. e., when the temperature becomes too low, we may reinitialize it to the maximum temperature and start a new scheme again.

In our setting the initial solution of the Simulated Annealing procedure will be obtained from the **SimultaneousWalkConstruction** procedure. The **SimultaneousWalkConstruction** produces the same solution with the ILS algorithm executed for 1 iteration. In the **SimultaneousWalkConstruction** the list of ids of walks is considered. Then, while the list is not empty, each walk id is obtained and the best profitable piece is inserted in the walk with this id. If no insertion was feasible, this id is removed from the list. When, the list of walk ids becomes empty, or equivalently, no insertion is feasible anymore in a walk, the method returns the solution obtained. The pseudocode of the **SimultaneousWalkConstruction** procedure follows (Algorithm 6).

Algorithm 6: SimultaneousWalkConstruction

```
1  walkIdsToInsert ← [0, 1, ..., K − 1]
2  iteratorOfWalkIdsToInsert ← iterator of walkIdsToInsert
3  while walkIdsToInsert is not empty do
4  |  walkId ← the value of iteratorOfWalkIdsToInsert
5  |  insertBestPieceIn(walkId)
6  |  if insertion did not happen then
7  |  |  remove walkId from walkIdsToInsert
8  |  end
9  |  iteratorOfWalkIdsToInsert ← next of iteratorOfWalkIdsToInsert
10 end
11 return solution
```

In the Simulated Annealing procedure, apart from the **InsertPiece** Neighborhood we also consider the **Replace** neighborhood. The **Replace** neighborhood of a feasible MTOPTW solution consists of all the solutions that can be obtained by replacing an included profitable piece in a walk by a non-included profitable piece. For example, in Figure 45 the neighboring solutions of the walk with node representation $W_i = (sl_i, u, v, w, el_i)$ in the **Replace** neighborhood are presented, considering that the only non-included profitable pieces are the edge $\{y, z\}$ and the node x. Figures 45(a) and 45(b) depict the replacement of u by the edge $\{y, z\}$, considering both directions that the edge can be traversed, i. e., from y to z and from z to y, respectively. Similarly, Figures 45(c) and 45(d) depict the replacement of $\{v, w\}$ by the edge $\{y, z\}$, while Figures 45(e) and 45(f) depict the replacement of u and $\{v, w\}$ by the non-included profitable node x, respectively.

Note that the replacement of an included profitable piece p_k^i by a non-included one candPiece, is equivalent to the removal of p_k^i followed by the insertion of candPiece between p_{k-1}^i and p_{k+1}^i.

The Simulated Annealing procedure takes into account five parameters: the number of schemes (numberOfSchemes), the maximum temperature (maxTemperature), the number of iterations executed in each scheme (schemeIterations), the cooling factor (coolingFactor) and the iterations needed in order to update the temperature (coolingIterations). The initial solution of the Simulated Annealing procedure is obtained by the **SimultaneousWalkConstruction** method (Algorithm 6). Then, the Simulated Annealing method loops for numberOfSchemes schemes. In each scheme, the temperature (T) initially becomes equal to the maxTemperature and an inner loop is executed for schemeIterations iterations. In the inner loop, a non-included profitable piece (candPiece) is randomly selected as well as a walk and an included piece (inclPiece). Then, if candPiece can be inserted after inclPiece without removing any included piece, the insertion takes place. If the

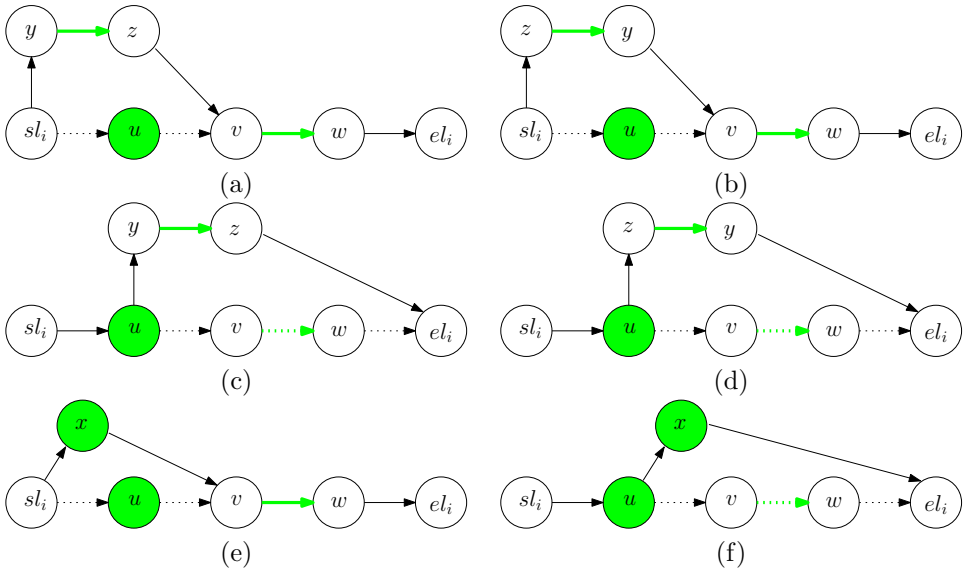

Figure 45: Illustration of **Replace** Neighborhood

insertion wasn't feasible, then the replacement of the inclPiece by candPiece is examined. If the replacement is feasible, then a randomly computed real number (prob) is obtained in the range $[0, 1]$ and if prob $< \exp(\frac{\text{diff}}{T})$, where diff is the difference of profits of candPiece and inclPiece, then candPiece replaces inclPiece. If either the insertion or the replacement results in a solution with the highest profit found, then the solution and its profit are stored as the bestSolution and bestProfit, respectively. Furthermore, if the inner loop has been executed for coolingIterations iterations since the last time the temperature was updated, then the temperature T gets the value $T \cdot$ coolingFactor. When, the first loop is completed, the algorithm returns the best solution and profit found. The pseudocode of the Simulated Annealing algorithm is listed in the sequel (Algorithm 7).

In order to have an effective algorithm, the parameters applied to the algorithm have to depend on the particular solution. In our setting we want to allow a small portion of the local optimum solution to change. For this reason we want the temperature to decrease after a portion of the included profitable pieces of the solution. This is important, because if we allow the coolIterations to be much greater than the size of the solution we may end up with a very different solution, when the temperature is high. On the other hand, if we consider coolIterations to be negligible compared to the solution's size, the solution won't escape from the local optimum. For this reason, we introduce a Solution Based Simulated Annealing (SBSA) that executes the Simulated Annealing with parameters dependending on the initial solution. Based on this goal, new parameters are introduced, namely the parameters coolItFactor and schemeItFactor. Furthermore, the number of profitable pieces in the initial solution (profPieces) is determined at the beginning of the method. The parameter coolItFactor is used to obtain a reasonable amount of cooling iterations, i. e., the coolIterations is set equal to profPieces $\cdot$ coolItFactor. The schemeItFactor is used to compute a reasonable number of iterations per scheme that will be a multiple of the cooling iterations, i. e., the schemeIt is set equal to the coolIteration $\cdot$ schemeItFactor. Finally, one more parameter is introduced, the numberOfIterations, that denotes the total number of iterations that will be executed during the SBSA. Based on this, the numSchemes is set equal to $\frac{\text{numberOfIterations}}{\text{schemeIt}}$. The maxTemperature and coolingFactor parameters of the Simulated Annealing are solution independent. Then, we use the parameters previously obtained for executing the Simulated Annealing. The pseudocode of the SBSA is given in Algorithm 8.

Algorithm 7: Simulated Annealing

```
solution ← SimultaneousWalkConstruction
for numberOfSchemes iterations do
    T ← maxTemperature
    for 1 ≤ itForScheme ≤ schemeIterations do
        candPiece ← a non-included profitable piece randomly selected
        wId ← a random number in [0, 1, ..., K − 1]
        inclPiece ← a randomly selected included profitable piece or the starting location of W_wId
        tryReplace ← true
        if the insertion of candPiece after inclPiece is feasible then
            insert candPiece after inclPiece
            tryReplace ← false
        end
        if candPiece can replace inclPiece and tryReplace then
            diff ← profit of candPiece minus profit of inclPiece
            prob ← a random real in the range [0, 1]
            if prob < exp(diff/T) then
                replace incPiece by candPiece
            end
        end
        if profit > bestProfit then
            bestProfit ← profit
            bestSolution ← solution
        end
        if itForScheme mod coolingIterations = 0 then
            T ← T · coolingFactor
        end
    end
end
return bestSolution;bestProfit
```

Algorithm 8: Solution Based Simulated Annealing (SBSA)

```
solution ← SimultaneousWalkConstruction
profPieces ← the total number of profitable pieces included in solution
coolIteration ← profPieces · coolItFactor
schemeIt ← coolIteration · schemeItFactor
numSchemes ← numberOfIterations / schemeIt

return Simulated Annealing with the parameters previously obtained
```

4.4.3 Assessment upon real data

Test Instances

To evaluate and compare the proposed algorithms, we have created test instances containing data related to the city of Athens, Greece. Our topology contains 18 scenic routes and 136 points of interest (POIs) that have been compiled from various tourist portals [5] and web services offering open APIs [6]. We have also included 100 hotels in our topology. The hotels are used as starting and ending locations of daily tourist walks. The travel costs between the locations of the topology were calculated using the OpenTripPlanner project [7].

The attributes of the POIs(time windows, visiting times and profits) are chosen as follows:

- 20% of POIs are open all day long in both weekdays and weekends, i. e., their time window is 0-1439, while they have profit and visiting time between 15 -30.
- 20% of the POIs have time windows 540 - 960 for weekdays and are closed in weekends, while their profit and visiting time is between 30 - 60.
- 20% of POIs have time windows 540 - 960 for weekdays and weekends, while their profit is between 70 - 100 and the visiting time is between 60 - 120.
- 20% of POIs have time windows 840 - 1140 for weekdays and weekends, while their profit and visiting time is between 30 - 60.
- 20% of POIs have time windows 480 - 780 for weekdays and are closed in weekends, while their profit is between 30 - 60.

The attributes of the scenic routes (time windows and profits) are chosen as follows:

- 25% of them are open all day long (their time windows is between 0 - 1439) for each day of the week.
- 25% of them have time windows 540 - 960 for weekdays and weekends.
- 25% of scenic routes have time windows 480 - 840 for weekdays and weekends.
- 25% of the scenic routes have time windows 540 - 960 for weekdays and are closed in weekends.
- All scenic routes have profit between 10 and 50.

Solutions of $1, 2, 3$ and 4 walks are required. For each number of walks, 100 test instances are considered, each with starting/ending location one of the hotels. In more detail, pref100 - pref199 preferences are asking for solutions of 1 walk, while pref2*(pref200-pref299), pref3*(pref300-pref399) and pref4*(pref400-pref499) are asking for $2, 3$ and 4 walks, respectively. Considering the starting/ending time of each walk in an instance, we assign the starting time to a randomly chosen time between 480 - 600, while we assign a randomly chosen number between 840 - 1080 to the ending time. Note, that for instances requiring more than 1 walk, different walks have the same starting/ending location, however they usually have different starting/ending times.

Results

All computations were carried out on a personal computer Intel Core i3 with 2.30 GHz processor and 4 GB RAM. Our tests aim at comparing the presented ILS and SBSA algorithm. The algorithms are compared with respect to the obtained profit and the execution time. Mostly preferred solutions are those associated with high profit values (higher profit values denote higher quality solutions)

[5] http://www.tripadvisor.com/, http://index.pois.gr/
[6] https://developers.google.com/places/documentation/
[7] http://www.opentripplanner.org/

and reduced execution time (as this denotes improved suitability for real-time applications). Both algorithms have been coded in C++.

The numberOfIterations parameter in ILS takes the values $100, 200, 400$ and 800 and the corresponding results are reported as ILS_100, ILS_200, ILS_400 and ILS_800. Considering the SBSA algorithm the parameters maxTemperature and coolingFactor are always set equal to 10 and 0.5, respectively, while the parameters numberOfIterations, coolItFactor and schemeItFactor are associated with the values numberOfIterations= 0.5 and 1 million, the coolItFactor takes the values $0.8, 0.5$ and 0.25 and the schemeItFactor takes the values 8 and 10. The results of the SBSA with specified values for the parameters numberOfIterations, coolItFactor and schemeItFactor are indicated as SBSA_numberOfIterations_coolItFactor_schemeItFactor; for example, for numberOfIterations equal to 0.5 million, coolItFactor equal to 0.8 and schemeItFactor equal to 8 the results obtained are indicated with SBSA_0.5M_0.8_8.

The algorithms have been employed upon the test instances described previously. Since, they are both randomized, each algorithm is executed 5 times for each instance. Table 9 illustrates the experimental results compiled from the executed algorithms for each value of the parameters. The results obtained are averaged with respect to the number of requested walks. In more detail, each algorithm with its associated parameter values is shown in the first column. The next four pairs of columns present the experimental results obtained for instances requesting $1, 2, 3$ and 4 walks, respectively. Each pair of columns shows the average profit obtained and the average execution time of each algorithm for the 5 executions of each instance asking for a specific number of walks. So, the average profit and execution time (in ms) for instances requesting 1 walk (pref100 - pref199) are given in the first pair of columns of Table 9, while the average results obtained for 2(pref200 - pref299), 3(pref300 - pref399) and 4(pref400 - pref499) walks are given in the second, third and fourth pair of columns of Table 9, respectively.

Table 9: Experimental Results

	1 walk		2 walks		3 walks		4 walks	
Algorithm	Profit	Time(ms)	Profit	Time(ms)	Profit	Time(ms)	Profit	Time(ms)
ILS_100	816.698	295.728	1359.972	340.59	1865.546	387.194	2286.534	402.608
ILS_200	821.578	590.264	1365.354	678.658	1872.24	771.248	2291.818	803.022
ILS_400	823.754	1179.93	1368.992	1354.696	1876.248	1542.638	2296.038	1604.118
ILS_800	826.342	2353.616	1375.468	2708.656	1880.298	3080.624	2299.984	3203.474
SBSA_0.5M_0.8_8	818.59	271.05	1333.24	275.958	1815.88	278.432	2233.714	279.096
SBSA_0.5M_0.5_8	818.16	271.778	1332.56	276.204	1816.134	278.806	2233.43	279.62
SBSA_0.5M_0.25_8	819.644	272.708	1333.426	277.302	1816.378	279.238	2233.71	279.954
SBSA_0.5M_0.8_10	820.736	268.846	1335.488	274.206	1818.05	276.85	2236.38	277.52
SBSA_0.5M_0.5_10	820.58	269.698	1334.888	274.414	1819.804	276.796	2235.72	277.838
SBSA_0.5M_0.25_10	818.574	271.284	1334.864	275.728	1819.41	277.968	2235.454	278.124
SBSA_1M_0.8_8	821.072	525.35	1333.036	534.368	1815.194	537.652	2234.246	538.258
SBSA_1M_0.5_8	820.6	526.03	1334.716	535.174	1815.97	538.54	2234.072	538.6
SBSA_1M_0.25_8	820.324	530.034	1334.178	536.93	1815.918	539.41	2233.732	539.794
SBSA_1M_0.8_10	820.066	522.626	1336.474	531.182	1818.038	534.66	2235.464	535.366
SBSA_1M_0.5_10	820.944	523.026	1336.128	531.81	1819.59	534.704	2237.192	535.798
SBSA_1M_0.25_10	820.3	527.324	1335.462	533.492	1820.054	536.534	2236.73	537.258

Based on the experimental results, we can easily conclude that increasing the algorithms' execution time, i. e., increasing the allowed number of iterations, improves slightly the solutions' quality. To see this, consider the ILS algorithm. The obtained average profit from 200 iterations is better the corresponding profit obtained from 100 iterations less than 0.7% for all number of walks, while the execution time is doubled. The same statement holds when comparing the results for 400 and 200 iterations as well as for 800 and 400 iterations. As far as the SBSA algorithm is considered, the profit obtained by one million iterations exceeds the profit obtained by half a million iterations by at most 0.5% for each walk, while the execution time is twice the one in the case of half a million iterations.

As far as the parameters coolItFactor and schemeItFactor are concerned, the results indicate that

the parameter coolItFactor does not significantly influence the output of SBSA algorithm. For example, for half a million iterations and schemeItFactor equal to 10 the difference of SBSA_0.5M_0.8_10's profit and SBSA_0.5M_0.25_10's profit is less than 0.27% and this is the biggest difference, when the rest of the parameters are the same. Apart from coolItFactor, schemeItFactor seems to influence the results slightly more, however the difference in profit is always less than 0.5%.

The comparison results between ILS and SBSA indicate that ILS obtains higher quality solutions in more execution time. The solutions obtained for the case of one walk by both ILS and SBSA are of similar profit, i. e., the ILS_100 produces slightly lower quality solutions than all the SBSA algorithms, while ILS_200, ILS_400, ILS_800 produce marginally better solutions, however in prolonged execution time. When, two or more walks are considered, ILS prevails over SBSA even with 100 iterations. For example, ILS_100's profit for two walks is higher than the profit obtained by any SBSA algorithm by at least 1.7%, while for three and four walks the average solutions' profit is at least 2.47% and 2.19% higher than the profit obtained by any SBSA algorithm. However, ILS requires more execution time than SBSA. ILS' execution time increases when the number of walks is increased, without modifying the number of iterations. This happens because in the perturbation step ILS removes a chain of included pieces from each walk. This results in increased execution time when the required number of walks is large, since reaching local optimum requires inserting pieces to all the walks. On the other hand, the execution time of the SBSA algorithm does not depend on the number of the constructed walks, but only on the number of iterations. This happens because in each step of SBSA, only one walk is mutated, hence the execution time is independent of the number of walks.

4.5 Conclusions

Our work on tourist tour planning in the framework of D3.6 has been towards tackling incrementally complex TTDP formulations, addressing realistic tourist requirements. Along this line, our research output has been twofold. Firstly, we extended our TDTOPTW algorithmic approaches (as presented in D3.5) so as to allow scheduling lunch breaks at appropriately located restaurants. Secondly, we investigated the AOP and MOP problems which allow incorporating scenic routes (in addition to point POIs) in our tour planning logic. We also investigated the MTOPTW problem and proposed novel efficient metaheuristics to tackle it.

We presented the first approximation algorithms for AOP in directed and undirected graphs. Specifically, we proposed a polylogarithmic approximation algorithm for the problem in directed graphs, a $(6 + \epsilon + o(1))-$approximation algorithm for the AOP in undirected graphs and a $(4 + \epsilon)-$approximation algorithm for the unweighted version of the problem. Moreover, we obtained approximation algorithms for the MOP, the extension of both the OP and the AOP, where both nodes and arcs are associated with profit. In the sequel, we introduced the MTOPTW , i. e., the extension of the MOP to multiple tours. The problem can be used to formulate realistic TTDP variants whose modeling requires multiple tourist tours, profits to be associated to both POIs (nodes of the network) and routes (arcs of the network) as certain routes may be more interesting to be traversed than others, while both POIs and routes are associated with visiting/traversing time windows. We proposed the first algorithmic approaches to tackle the MTOPTW: an Iterated Local Search metaheuristic and a Simulated Annealing metaheuristic. The main design objectives for both algorithms were to derive high quality solutions (maximizing the profit gained by visiting the POIs and traversing the routes) while executing fast enough to support online web and mobile applications. We evaluated and compared the proposed algorithms on test instances based on real data related to the city of Athens, Greece.

5 Discussion and Final Remarks

Multimodal public transport routing. Following the submission of D3.3 and D3.4, (M21 of eCOMPASS onwards), we continued our research investigating new algorithmic approaches in multimodal route planning. In this deliverable, we have complemented our algorithms with multi-core and SIMD parallelization techniques. Furthermore, we have adapted our multimodal multi-criteria route planning prototype to Berlin data sources. We delivered eco-friendliness assessment based on substantiated CO_2 consumption estimates. The multi-criteria optimization approach enables us to present the user several good alternative travel plans, highlighting eco-friendly travel choices while also supplying more traditional results. While we have learned from our user study in beginning of the eCOMPASS project that users would vastly prefer quick journeys over eco-friendly journeys, we are confident that by increasing *eco-awareness* (by showing the different trade-offs between quick and less consuming journeys), we enable end-users to establish more eco-friendly traveling styles.

The algorithms using methods from stochasticity and machine learning have been implemented and extensively tested on the network of Zürich. At least for that network, the running time was fast and we were able to compute journeys that arrive on time with a high probability. However, these algorithms compute their predictions using historic delay data, which is not available for the network of Berlin. This is the reason why these algorithms were not included in the prototyped implementations of eCompass. Moreover, the network of Berlin is substantially larger than the network of Zürich, and it is not clear whether the running time of the proposed algorithms scales well.

The success of our approaches has been demonstrated by our extensive experimental evaluation in Deliverables D3.3, D3.4, and D3.6, as well as by the use of our prototyped implementations for the applications developed in the framework of WP 5 and piloted in Berlin, Germany (see D5.3).

Outside of the main focus, we also did a preliminary investigation on integrating car-sharing into public transportation services. So far, we have considered only simple models and provided initial algorithm-theoretical investigations. For the details on our work we refer to [12]. The problems in this area deserve further study. We mainly focused on car sharing as an isolated problem, to gain a better understanding of the challenges that arise. However, the clear and necessary next steps are to study solutions that explicitly combine public transportation with car sharing. We believe that the car sharing service has the potential to offer the missing flexibility to public transportation, and thus further promote environmentally friendly means of transportation.

Multimodal tourist tour planning. In Deliverable D3.5 we introduced novel TOPTW and TDTOPTW algorithmic solutions addressing the basic requirements of TTDP applications. Our TDTOPTW metaheuristics approaches allow modeling multimodal transfers among POIs, hence, they better fit eCOMPASS purposes than TOPTW approaches. The main design objectives for our TDTOPTW algorithms have been to derive high quality TDTOPTW solutions (maximizing tourist satisfaction), while minimizing the number of transit transfers and executing fast enough to support online web and mobile applications. Our metaheuristics have been shown to perform better with respect to solutions quality compared to baseline approaches considering precalculated average travel times (for moving from a POI to another by public transit), especially when considering relatively large datasets (i. e., large number of POIs) and low frequency of public transit services (then, the average travel time approximation is not good enough). The high efficiency of our TDTOPTW metaheuristics (execution time well below 1 sec in most usage scenarios) has been evidenced by both our extensive experimentation with realistic datasets (see D3.5 - Section 4.5.2) and the use of the prototyped web and mobile client applications developed in the framework of WP5 and piloted in Berlin, Germany (see D5.3). Hence, we have completely succeeded with respect to the aims originally set for our eCOMPASS tourist tour planning product.

Following the submission of D3.5, (M21 of eCOMPASS onwards), we continued our research investigating new algorithmic approaches in tourist tour planning. Our effort has been directed

towards increasingly elaborate and complex TTDP formulations, trying to address further realistic requirements of tourists. Along this line, our main threads of research in M21-M36 of the project have tackled the following issues: (a) we extended our TDTOPTW SlackCSCRoutes algorithm so as to cater for scheduling lunch breaks through recommending affordable restaurants conveniently located along the tourist tour; (b) we investigated approaches which allow incorporating scenic routes into recommended tours; along this line we looked at AOP and MOP problems (obtaining approximation algorithms) and also developed two novel efficient metaheuristics to tackle MTOPTW.

MTOPTW solutions are appropriate to highlight the appeal of natural, cultural, architectural and historical assets of tourist destinations. Thus, we argue that MTOPTW captures the requirements of tourist destination visitors better than any other relevant approach found today in the TTDP-related literature. Extensions to our mobile (Android) tour planning application (see D5.3) are currently underway to bind it to the MTOPTW web service.

It should be noted that, aside the promotion of scenic assets in tourist destinations, our MTOPTW algorithms may be used to discourage access to dangerous areas through penalizing the attractiveness of dangerous road segments (e. g., assigning low or negative profit). Note that the profit may be time-dependent, e. g., to allow the designation of areas that may be dangerous only in the night time. Moreover, our algorithms could be easily extended to allow tour planning on a larger scale, e. g., to highlight scenic routes for drivers which could be desirable for road trips.

Admittedly, all the algorithmic solutions developed for tourist tour planning have not directly targeted CO_2 reduction per se. Namely, the CO_2 emissions derived from tourists transfers have not been regarded as an optimization criterion. Besides, the precalculated multimodal travel profiles (i.e. the time-dependent travel times among POIs) are derived using the fastest option among those discussed in D3.3 (although this could be easily substituted by the eco-friendliest option). The main intuition in our approach has been to improve the accessibility and mitigate the complexity of public transit networks, as perceived by tourists. Along this line, our solutions aim at facilitating the use of public transit, which the tourists would be reluctant to use otherwise. Also, the precomputed multimodal travel profiles only consider public transit and walking transportation modalities, excluding the option of private cars or taxis, thereby being inherently eco-friendly. However, an indication of the CO_2 emission savings (compared to the CO_2 emissions incident in the respective car transfers) of recommended walking or public transit transfers (separately for each leg of the tour or collectively for the while trip) could be displayed in future releases of our tourist tour planning web/mobile application products.

Certainly, our TTDP modeling could be refined so as to capture additional aspects of realistic tourist needs. The investigation of those aspects has not been possible within the time frame of eCOMPASS, hence, they will be considered in our future research:

- Consider cases that users deviate from their tour schedule so as to automatically adapt the remainder of their tour. Similarly, tour updates could be triggered in the occasion of transit services delays due to traffic congestion or unexpected incidents.
- Take into account the accessibility status of walking routes, transit services and POIs so as to derive tours accessible by individuals with motoring disabilities.
- Provide a 'standalone' version of eCOMPASS mobile application (i. e., requiring no Internet connection) to save roaming charges; to address this issue, besides integrating city map data into the application executable, the algorithmic solutions derived in WP 3 would be modified (e. g., reduce iterations searching for improved solutions) to execute fast on slower machines without compromising much the quality of derived solutions.

Final Remarks. This deliverable concludes our investigation in WP 3 into deriving optimized urban routes using public means of transportation targeting residents and tourists. We have researched novel methods for environment friendly routes in urban public transportation networks.

We have developed mathematical sound models for various (context-aware) route planning scenarios arising in the field of urban human mobility for city residents, commuters and tourists. We have provided several new algorithmic methods for multimodal routes in urban transportation with respect to multiple criteria and high robustness. All WP 3 approaches have had prototype implementation and extensive experimental evaluation on the performance and quality of derived results.

In particular, in this deliverable D3.6, we have assessed the success of the algorithmic solutions developed within WP3 upon real public transportation network data, discussing necessary modifications with respect to Deliverable D3.3.1, D3.3.2, D3.4.1, D3.4.2, D3.5.1, and D3.5.2 solutions, identifying the most applicable and technically most robust solutions. Our algorithmic solutions have considerably advanced the state of the art and allow for faster, more precise and eco-friendly route and tourist trip planning. These solutions were further integrated by WP 5 partners and successfully piloted in Berlin within the scope of WP 6.

Acknowledgement

We wish to thank the Athens Urban Transport Organisation (OASA), the Verkehrsverbund Berlin-Brandenburg (VBB), the Verkehrsbetriebe Zürich (VBZ) for providing real-world transportation data and useful information about the network structure. We especially thank VBZ for providing historic transit delay data.

References

[1] Ittai Abraham, Daniel Delling, Andrew V. Goldberg, and Renato F. Werneck. Hierarchical hub labelings for shortest paths. In *Proceedings of the 20th Annual European Symposium on Algorithms (ESA'12)*, volume 7501 of *Lecture Notes in Computer Science*, pages 24–35. Springer, 2012.

[2] C. Archetti, A. Corberan, I. Plana, J.M. Sanchis, and Speranza M.G. A matheuristic for the team orienteering arc routing problem. Working paper, Department of Economics and Management, University of Brescia, 2013.

[3] C. Archetti, D. Feillet, A. Hertz, and M. G. Speranza. The undirected capacitated arc routing problem with profits. *Computers & Operations Research*, 37(11):1860 – 1869, 2010.

[4] C. Archetti, M. G. Speranza, A. Corberan, J. M. Sanchis, and I. Plana. The team orienteering arc routing problem. *Transportation Science*, 0(0):null, 0.

[5] Hannah Bast. Car or public transport – two worlds. In *Efficient Algorithms*, volume 5760 of *Lecture Notes in Computer Science*, pages 355–367. Springer, 2009.

[6] Hannah Bast, Erik Carlsson, Arno Eigenwillig, Robert Geisberger, Chris Harrelson, Veselin Raychev, and Fabien Viger. Fast routing in very large public transportation networks using transfer patterns. In *Proceedings of the 18th Annual European Symposium on Algorithms (ESA'10)*, volume 6346 of *Lecture Notes in Computer Science*, pages 290–301. Springer, 2010.

[7] Annabell Berger, Daniel Delling, Andreas Gebhardt, and Matthias Müller–Hannemann. Accelerating time-dependent multi-criteria timetable information is harder than expected. In *Proceedings of the 9th Workshop on Algorithmic Approaches for Transportation Modeling, Optimization, and Systems (ATMOS'09)*, OpenAccess Series in Informatics (OASIcs), 2009.

[8] Annabell Berger, Martin Grimmer, and Matthias Müller–Hannemann. Fully dynamic speed-up techniques for multi-criteria shortest path searches in time-dependent networks. In *Proceedings of the 9th International Symposium on Experimental Algorithms (SEA'10)*, volume 6049 of *Lecture Notes in Computer Science*, pages 35–46. Springer, May 2010.

[9] A. Blum, S. Chawla, D. R. Karger, T. Lane, A. Meyerson, and M. Minkoff. Approximation algorithms for orienteering and discounted-reward tsp. In *Proceedings of the 44th Annual IEEE Symposium on the Foundations of Computer Science*, pages 46 –55, 2003.

[10] A. Blum, S. Chawla, D. R. Karger, Lane T., A. Meyerson, and M. Minkoff. Approximation algorithms for orienteering and discounted-reward tsp. *SIAM J. Comput.*, 37(2):653–670, 2007.

[11] C. Blum and A. Roli. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Comput. Surv.*, 35:268–308, September 2003.

[12] Katerina Böhmová, Yann Disser, Matús Mihalák, and Peter Widmayer. Interval selection with machine-dependent intervals. In *Algorithms and Data Structures - 13th International Symposium, WADS 2013, London, ON, Canada, August 12-14, 2013. Proceedings*, pages 170–181, 2013.

[13] Joachim M. Buhmann, Matús Mihalák, Rastislav Srámek, and Peter Widmayer. Robust optimization in the presence of uncertainty. In *Innovations in Theoretical Computer Science, ITCS '13, Berkeley, CA, USA, January 9-12, 2013*, pages 505–514, 2013.

[14] V. Cerny. Thermodynamical approach to the traveling salesman problem: An efficient simulation algorithm. *Journal of Optimization Theory and Applications*, 45(1):41–51, 1985.

[15] C. Chekuri, N. Korula, and M. Pál. Improved algorithms for orienteering and related problems. In *Proceedings of the 19th Annual ACM-SIAM symposium on Discrete Algorithms*, SODA '08, pages 661–670, 2008.

[16] C. Chekuri, N. Korula, and M. Pál. Improved algorithms for orienteering and related problems. *ACM Trans. Algorithms*, 8(3):23:1–23:27, 2012.

[17] R. Deitch and S.P. Ladany. The one-period bus touring problem: Solved by an effective heuristic for the orienteering tour problem and improvement algorithm. *European Journal of Operational Research*, 127(1):69 – 77, 2000.

[18] Daniel Delling. Time-dependent SHARC-routing. *Algorithmica*, 60(1):60–94, May 2011.

[19] Daniel Delling, Andrew V. Goldberg, Thomas Pajor, and Renato F. Werneck. Customizable route planning. In *Proceedings of the 10th International Symposium on Experimental Algorithms (SEA'11)*, volume 6630 of *Lecture Notes in Computer Science*, pages 376–387. Springer, 2011.

[20] Daniel Delling, Bastian Katz, and Thomas Pajor. Parallel computation of best connections in public transportation networks. *ACM Journal of Experimental Algorithmics*, 17(4):4.1–4.26, July 2012.

[21] Daniel Delling, Thomas Pajor, and Dorothea Wagner. Accelerating multi-modal route planning by access-nodes. In *Proceedings of the 17th Annual European Symposium on Algorithms (ESA'09)*, volume 5757 of *Lecture Notes in Computer Science*, pages 587–598. Springer, September 2009.

[22] Daniel Delling, Thomas Pajor, and Renato F. Werneck. Round-based public transit routing. In *Proceedings of the 14th Meeting on Algorithm Engineering and Experiments (ALENEX'12)*, pages 130–140. SIAM, 2012.

[23] Daniel Delling, Peter Sanders, Dominik Schultes, and Dorothea Wagner. Engineering route planning algorithms. In *Algorithmics of Large and Complex Networks*, volume 5515 of *Lecture Notes in Computer Science*, pages 117–139. Springer, 2009.

[24] Daniel Delling and Dorothea Wagner. Pareto paths with SHARC. In *Proceedings of the 8th International Symposium on Experimental Algorithms (SEA'09)*, volume 5526 of *Lecture Notes in Computer Science*, pages 125–136. Springer, June 2009.

[25] Daniel Delling and Renato F. Werneck. Faster customization of road networks. In *Proceedings of the 12th International Symposium on Experimental Algorithms (SEA'13)*, volume 7933 of *Lecture Notes in Computer Science*, pages 30–42. Springer, 2013.

[26] Julian Dibbelt, Thomas Pajor, Ben Strasser, and Dorothea Wagner. Intriguingly simple and fast transit routing. In *Proceedings of the 12th International Symposium on Experimental Algorithms (SEA'13)*, volume 7933 of *Lecture Notes in Computer Science*, pages 43–54. Springer, 2013.

[27] Julian Dibbelt, Thomas Pajor, and Dorothea Wagner. User-constrained multi-modal route planning. In *Proceedings of the 14th Meeting on Algorithm Engineering and Experiments (ALENEX'12)*, pages 118–129. SIAM, 2012.

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

[29] Yann Disser, Matthias Müller–Hannemann, and Mathias Schnee. Multi-criteria shortest paths in time-dependent train networks. In *Proceedings of the 7th Workshop on Experimental Algorithms (WEA'08)*, volume 5038 of *Lecture Notes in Computer Science*, pages 347–361. Springer, June 2008.

[30] Andrew Ensor and Felipe Lillo. Partial order approach to compute shortest paths in multimodal networks. Technical report, http://arxiv.org/abs/1112.3366v1, 2011.

[31] Marco Farina and Paolo Amato. A fuzzy definition of "optimality" for many-criteria optimization problems. *IEEE Transactions on Systems, Man, and Cybernetics, Part A*, 34(3):315–326, 2004.

[32] A. Garcia, P. Vansteenwegen, O. Arbelaitz, W. Souffriau, and M. T. Linaza. Integrating public transportation in personalised electronic tourist guides. *Computers & Operations Research*, 40(3):758 – 774, 2013.

[33] D. Gavalas, C. Konstantopoulos, K. Mastakas, and G. Pantziou. A survey on algorithmic approaches for solving tourist trip design problems. *J. of Heuristics*, 20(3):291–328, 2014.

[34] Robert Geisberger. Contraction of timetable networks with realistic transfers. In *Proceedings of the 9th International Symposium on Experimental Algorithms (SEA'10)*, volume 6049 of *Lecture Notes in Computer Science*, pages 71–82. Springer, May 2010.

[35] Robert Geisberger, Moritz Kobitzsch, and Peter Sanders. Route planning with flexible objective functions. In *Proceedings of the 12th Workshop on Algorithm Engineering and Experiments (ALENEX'10)*, pages 124–137. SIAM, 2010.

[36] Robert Geisberger, Peter Sanders, Dominik Schultes, and Daniel Delling. Contraction hierarchies: Faster and simpler hierarchical routing in road networks. In *Proceedings of the 7th Workshop on Experimental Algorithms (WEA'08)*, volume 5038 of *Lecture Notes in Computer Science*, pages 319–333. Springer, June 2008.

[37] Robert Geisberger, Peter Sanders, Dominik Schultes, and Christian Vetter. Exact routing in large road networks using contraction hierarchies. *Transportation Science*, 46(3):388–404, August 2012.

[38] General Transit Feed. `https://developers.google.com/transit/gtfs/`, 2010.

[39] B. L. Golden, L. Levy, and R. Vohra. The orienteering problem. *Naval Research Logistics (NRL)*, 34(3):307–318, 1987.

[40] HaCon - Ingenieurgesellschaft mbH. `http://www.hacon.de`, 1984.

[41] Pierre Hansen. Bricriteria path problems. In *Multiple Criteria Decision Making – Theory and Application –*, pages 109–127. Springer, 1979.

[42] Moritz Hilger, Ekkehard Köhler, Rolf H. Möhring, and Heiko Schilling. Fast point-to-point shortest path computations with arc-flags. In *The Shortest Path Problem: Ninth DIMACS Implementation Challenge*, volume 74 of *DIMACS Book*, pages 41–72. American Mathematical Society, 2009.

[43] Martin Holzer, Frank Schulz, and Dorothea Wagner. Engineering multilevel overlay graphs for shortest-path queries. *ACM Journal of Experimental Algorithmics*, 13(2.5):1–26, December 2008.

[44] S. Kirkpatrick, D. Jr. Gelatt, and M.P. Vecchi. Optimization by simmulated annealing. *Science*, pages 671–680, 1983.

[45] N. J. Korula. *Approximation algorithms for network design and orienteering.* PhD thesis, University of Illinois at Urbana-Champaign, 2010.

[46] G. Laporte and S. Martello. The selective travelling salesman problem. *Discrete Applied Mathematics*, 26(2-3):193 – 207, 1990.

[47] Dominique LaSalle and George Karypis. Multi-threaded graph partitioning. In *27th International Parallel and Distributed Processing Symposium (IPDPS'13)*. IEEE Computer Society, 2013.

[48] Sejoon Lim, Christian Sommer, Evdokia Nikolova, and Daniela Rus. Practical route planning under delay uncertainty: Stochastic shortest path queries. In *Robotics: Science and Systems VIII, University of Sydney, Sydney, NSW, Australia, July 9-13, 2012*, 2012.

[49] Helena Lourenco, Olivier Martin, and Thomas Stützle. Iterated local search. In F. Glover and G. Kochenberger, editors, *Handbook of Metaheuristics*, volume 57 of *International Series in Operations Research & Management Science*, pages 320–353. Springer New York, 2003.

[50] J. Maervoet, P. Brackman, K. Verbeeck, P. De Causmaecker, and G. Vanden Berghe. Tour suggestion for outdoor activities. In *Proceedings of the 12th International Symposium on Web and Wireless Geographical Information Systems (W2GIS'13)*, volume 7820 of *LNCS*, pages 54–63, 2013.

[51] Metropolitan Transportation Authority of the State of New York. `http://www.mta.info/`, 1966.

[52] Matthias Müller–Hannemann and Mathias Schnee. Finding all attractive train connections by multi-criteria pareto search. In *Algorithmic Methods for Railway Optimization*, volume 4359 of *Lecture Notes in Computer Science*, pages 246–263. Springer, 2007.

[53] Matthias Müller–Hannemann, Frank Schulz, Dorothea Wagner, and Christos Zaroliagis. Timetable information: Models and algorithms. In *Algorithmic Methods for Railway Optimization*, volume 4359 of *Lecture Notes in Computer Science*, pages 67–90. Springer, 2007.

[54] Matthias Müller–Hannemann and Karsten Weihe. Pareto shortest paths is often feasible in practice. In *Proceedings of the 5th International Workshop on Algorithm Engineering (WAE'01)*, volume 2141 of *Lecture Notes in Computer Science*, pages 185–197. Springer, 2001.

[55] V. Nagarajan and R. Ravi. The directed orienteering problem. *Algorithmica*, 60:1017–1030, August 2011.

[56] PTV AG – Planung Transport Verkehr. http://www.ptv.de, 1979.

[57] Evangelia Pyrga, Frank Schulz, Dorothea Wagner, and Christos Zaroliagis. Efficient models for timetable information in public transportation systems. *ACM Journal of Experimental Algorithmics*, 12(2.4):1–39, 2008.

[58] Peter Sanders. Algorithm engineering – an attempt at a definition. In *Efficient Algorithms*, volume 5760 of *Lecture Notes in Computer Science*, pages 321–340. Springer, 2009.

[59] Peter Sanders and Dorothea Wagner. Algorithm engineering. *Informatik Spektrum*, 36(2):187–190, April 2013.

[60] Frank Schulz, Dorothea Wagner, and Karsten Weihe. Dijkstra's algorithm on-line: An empirical case study from public railroad transport. *ACM Journal of Experimental Algorithmics*, 5(12):1–23, 2000.

[61] Christian Sommer. Shortest-path queries in static networks, 2012. submitted. Preprint available at `http://www.sommer.jp/spq-survey.htm`.

[62] W. Souffriau, P. Vansteenwegen, G. Vanden Berghe, and D. Van Oudheusden. The planning of cycle trips in the province of east flanders. *Omega*, 39(2):209 – 213, 2011.

[63] T. Tsiligirides. Heuristic methods applied to orienteering. *The Journal of the Operational Research Society*, 35(9):797–809, 1984.

[64] P. Vansteenwegen, W. Souffriau, and D. Van Oudheusden. The orienteering problem: A survey. *European Journal of Operational Research*, 209(1):1 – 10, 2011.

[65] P. Vansteenwegen, W. Souffriau, G. Vanden Berghe, and D. Van Oudheusden. Iterated local search for the team orienteering problem with time windows. *Computers & Operations Research*, 36:3281–3290, 2009.

[66] P. Vansteenwegen and D. Van Oudheusden. The mobile tourist guide: An or opportunity. *Operational Research Insight*, 20(3):21–27, 2007.

[67] E.E. Zachariadis and C.T. Kiranoudis. Local search for the undirected capacitated arc routing problem with profits. *European Journal of Operational Research*, 210(2):358 – 367, 2011.

[68] Lotfi A. Zadeh. Fuzzy logic. *IEEE Computer*, 21(4):83–93, 1988.