eCO-friendly urban Multi-modal 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

D2.4 – Final Assessment of Eco-friendly Vehicle Routing Algorithms

Workpackage:	WP2 – Algorithms for Vehicle Routing
Due date of deliverable:	30 November 2014
Actual submission date:	30 November 2014
Responsible Partner:	CTI
Contributing Partners:	CERTH, CTI, ETHZ, KIT

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: Algorithms, heuristics, experimental evaluation, route planning, traffic prediction, time-dependent shortest path, customizable contraction hierarchies, alternative routes, robust routes, fleets of vehicles.

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
1.0	13.11.2014	First draft sent to reviewers	Spyros Kontogiannis, CTI
1.0	17.11.2014	Reviewers send comments	Andreas Gemsa, KIT Stefanos Makris, CERTH
1.1	18.11.2014	Reviewers' comments incorporated (sent to PQB)	Spyros Kontogiannis, CTI
1.1	20.11.2014	PQB sends comments	
1.2	21.11.2014	PQB's comments incorporated	Spyros Kontogiannis, CTI
1.3	30.11.2014	Final (approved by PQB, sent to the Project Officer)	Christos Zaroliagis, CTI

Deliverable manager

- Spyros Kontogiannis, CTI

List of Contributors

- Julian Dibbelt, KIT
- Dimitris Gkortsilas, CTI
- Kalliopi Giannakopoulou, CTI
- Dionisis Kehagias, CERTH
- Spyros Kontogiannis, CTI
- Sandro Montanari, ETHZ
- Georgia Papastavrou, CTI
- Andreas Paraskevopoulos, CTI
- Ben Strasser, KIT
- Christos Zaroliagis, CTI

List of Evaluators

- Andreas Gemsa, KIT
- Stefanos Makris, CERTH

Summary

The purpose of this deliverable is to present the outcomes of the assessment of success for the algorithmic solutions developed within WP2 in the actual implementation environments of WP5, discuss possible modifications w.r.t. the original solutions, and identify the technically most robust solutions.

Contents

1 Introduction

This deliverable presents extensive experimental evaluation of the most mature algorithmic solutions that have been developed within eCOMPASS, with emphasis on the eco-footprint awareness for the provided solutions for private vehicles and fleets of vehicles. It describes how the algorithmic solutions developed for the problems related to WP2 were improved and extended in order to yield better solutions in a more efficient way.

1.1 Objectives and scope of D2.4

The goal of WP2 is to develop novel algorithmic methods for optimization of problems related to routing of private vehicles and fleets of vehicles in urban areas, considering the environmental impact as one of the main parameters of the optimization objective. This document contains extensive experimental evaluations for the most mature algorithmic solutions provided within eCOMPASS/WP2 for the last 20 months.

The present deliverable is the outcome of the following tasks:

- Task 2.2: Eco-friendly private vehicle routing algorithms. Task 2.2 aims at designing routing algorithms for private vehicles. The computed routes should be optimized also with respect to their environmental footprint and should take into consideration traffic prediction techniques as well. Furthermore, the trade-off between data precision and solution robustness is also investigated in the context of this task.

- Task 2.3: Eco-friendly routing algorithms for fleets of vehicles. Task 2.3 aims at designing routing algorithms for fleets of vehicles. The application scenario for this task is a transportation company wishing to schedule the delivery or collection of goods in the most efficient and environmentally-friendly way as possible.

The algorithms developed for Task 2.2 and Task 2.3 should be designed in such a way that the environmental impact of the computed routes is taken into account, while aiming at outperforming the state-of-art techniques for classical routing problems in terms of quality (i.e., precision) and efficiency. Since the environmental impact was not opted as a essential optimization criterion in the User Requirements Analysis of eCOMPASS (c.f. Deliverable D1.1), our strategic choice was to assure eco-awareness by assessing and reporting to the driver the eco-footprint of the proposed routes. Moreover, our Traffic Prediction mechanism provides the driver with a visualized forecast for the evolution of congestion, in order to possibly incentivise the selection of Alternative Routes which might then not be so much worse (e.g., by means of actual travel-time) as estimated by the (static) historic traffic data.

Furthermore, dynamic scenarios should be taken into account, wherein the input is not statically predetermined but depends on several factors, like the time at which a query has been issued, or the current road traffic conditions. In scenarios where deriving optimum solutions in an efficient manner is not feasible, the computation of approximate solutions is taken into account. Towards this direction, within eCOMPASS we have proposed algorithmic solutions for (i) proposing routes with respect to the (time-dependent travel-time metric), and (ii) for creating metric-independent orders for the vertices so that the Contraction Hierarchies technique works efficiently under various graph metrics.

All our experimental evaluations have demonstrated significant improvements in execution times, compared to existing state-of-art approaches in the literature. Moreover, the deviations of the proposed routes by our algorithms, from environmentally-optimal routes, is rather limited. Additionally, in cases where approximately optimal solutions are proposed (e.g., when optimizing travel-times for the time-dependent shortest path problem), the proposed routes, apart from being much more efficient to compute than the optimal routes, they also achieve extremely good stretch factors with respect to the chosen optimization criterion (e.g., travel-time), whereas the additional deviations

from the environmentally optimal routes is also negligible when compared to the deviations of the optimal routes for the chosen optimization criterion.

1.2 Structure of the Document

The remaining sections of this document present extensive experimental assessments of the algorithms developed within the scope of the project. Section 2 deals with the experimental evaluation of the most prominent traffic prediction techniques. Section 3 describes experiments on various oracles for efficiently answering shortest path queries under the time-dependent travel-time metric for the city of Berlin. Section 4 experiments on precomputing the graph of Western Europe so as to create a metric-independent order of the vertices, to be fed in the Contraction Hierarchies approach so that efficient responses to arbitrary queries are achieved, under various scalar metrics (e.g. distance and constant travel-times). Section 5 experiments on the computation of alternative routes under a static travel-time metric. Section 6 considers the issues arising in the computation of routes when the data is noisy or not completely reliable, namely, it addresses computation of so-called *robust routes*. Section 7 illustrates the eCOMPASS approach for the computation of routes for delivery companies that need to schedule the delivery of goods over fleets of vehicles. Finally, Section 8 concludes this document.

2 Traffic Prediction

This section presents the results of the experiments conducted for the evaluation of the performance of the traffic predictions algorithms that were developed within WP2. The details of the algorithms are included in deliverable D2.2.1.

2.1 Parametric Approach

The first time series-based technique that also comprises an improvement of the previously developed Lag-STARIMA technique, was evaluated against five benchmark methods: (a) a Random Forest algorithm (RF non-parametric) (b) k-Nearest Neighbor (kNN non-parametric) (c) Historic Average (naive) (d) a classic STARIMA model which uses the Euclidean distance to estimate the neighbor matrix W (parametric) and (e) a lag-STARIMA model which uses CoD for the same reason (parametric). The last two STARIMA models are applied on time series as a whole (not segmented as the enhanced version).

The overall metric that characterizes the efficiency of the prediction and concerns the end-users is the estimated travel time and the goal is to minimise the predicted travel time error. The predicted speeds are transformed into travel time (expressed in minutes per km) by inverting and multiplying by 60. These values are compared to the travel time values that correspond to the actual speed values for a specific road and time interval using the RMSE metric which is calculated by:

$$RMSE = \sqrt{\frac{1}{N}\sum_{j=1}^{N}(\hat{x}_{i,j} - x_{i,j})^2}, \tag{1}$$

where $x_{i,j}$ and $\hat{x}_{i,j}$ are the real and the predicted travel time values respectively for road i at interval j. The travel time is obtained by the formula $x_{i,j} = 60/V_{i,j}$ where $V_{i,j}$ is the speed value of road i at time interval j.

In order to implement the procedure described in the previous sections there are many parameters that need to be tuned. These are the threshold of the segmentation algorithm, the outlier detection filter threshold, the number of the clusters used for the imputation and the window size of the moving average filter that is used in order to smooth the time series. The values of these

Table 1: Model Parameters

Parameter Name	Parameter Value
Segment Maximum Threshold	40 km/h
Outlier detection filter	20 km/h
Number of clusters	200
Window size of the moving average filter	30 intervals

Table 2: Comparison of the Implemented Algorithm with the 2 benchmark methods for all days of the week.

Average RMSE	RF	kNN	Historic Average	STARIMA	Lag-STARIMA	SLS
Monday	2.81608	2.80217	2.73945	2.5786	2.5429	2.35826
Tuesday	2.92331	2.78204	2.73707	2.58007	2.56447	2.37272
Wednesday	2.94556	2.92513	2.81248	2.69253	2.66009	2.65601
Thursday	3.06745	3.10003	3.00367	2.95759	2.92381	2.78093
Friday	2.95898	2.963	2.90613	2.71091	2.67041	2.6289
Saturday	2.99586	2.77798	2.58505	2.44476	2.41561	2.42171
Sunday	3.09342	2.50968	2.32077	2.17002	2.13472	2.02932

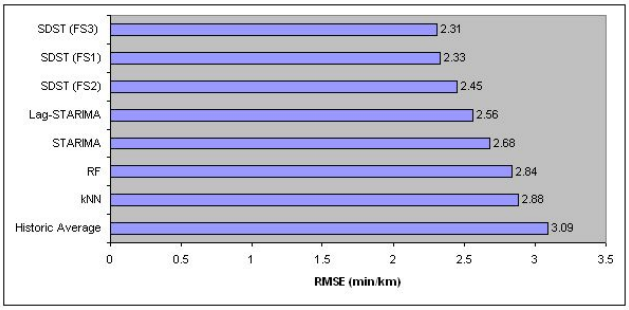


Figure 1: Comparison of the implemented algorithm with the benchmark methods for all days of the week

parameters were found mostly by trial and error but since the model was trained and tested for all days of the given dataset, which is two weeks in total, the values that were chosen are pretty robust. For example for the threshold of the segmentation algorithm, since periodicity of traffic is not optimal as noted before, intuitively a small threshold value that produces a large number of segments will be outperformed by a value that produces only a few segments that capture the general trend of the series. As a result, after a few tests, the value of 40 km/h was chosen as the segmentation threshold. A list of all the parameter values is shown in the following table:

The average RMSE for different forecasting periods and for all the roads of the Berlin dataset for each day of the week are presented on 2 and Fig. 1. The results indicate that SLS outperforms all its competitors (has a lower average RMSE) for all days of the week except of Saturday where is ranked second (first is Lag-STARIMA).

Table 3: Comparison of the benchmark algorithms for various forecasting periods

RMSE	5min	10min	15min	20min	25min	30min	35min	40min	45min	50min	55min	60min
Historic Average	3.06	3.06	3.02	3.05	3.07	3.08	3.12	3.13	3.13	3.15	3.15	3.14
kNN	2.84	2.84	2.82	2.83	2.85	2.85	2.89	2.92	2.93	2.94	2.95	2.92
RF	2.76	2.81	2.81	2.82	2.85	2.87	2.86	2.85	2.85	2.82	2.86	2.86
STARIMA	2.63	2.65	2.64	2.67	2.65	2.69	2.72	2.78	2.78	2.74	2.74	2.69
Lag-STARIMA	2.52	2.54	2.55	2.53	2.49	2.49	2.52	2.53	2.53	2.60	2.55	2.57
SDST(FS2, Naive Random Number generator)	2.44	2.46	2.42	2.40	2.41	2.41	2.43	2.46	2.48	2.52	2.50	2.45
SDST(FS1, Sophisticated Random Number generator)	2.31	2.32	2.30	2.29	2.29	2.29	2.31	2.33	2.35	2.37	2.34	2.35
SDST(FS3, Sophisticated Random Number generator)	2.29	2.29	2.27	2.26	2.27	2.27	2.30	2.32	2.34	2.36	2.33	2.35

2.2 Non-Parametric Approach

This section presents the results of the experimental evaluation procedure for the new non-parametric traffic prediction technique that was developed in WP2. We compared this non-parametric approach, namely Speed Dynamic Short-Term (SDST) forecasting technique to a set of techniques selected from the literature. In particular we used for benchmarking the following techniques: (non-parametric) kNN [1], Random Forest [3], and (parametric) STARIMA [4], Lag-STARIMA [2]. We have also included our technique with the three highest information gain feature sets: FS1, FS2, FS3 described in Table 2.

The overall accuracy of all benchmarked techniques is shown in Figure 2 that presents all techniques in ranked order based on their accuracy.

The metric used for evaluating the performance of the proposed and benchmark techniques is the Root Mean Square Error (RMSE), which was selected over other metrics, such as the Mean Average Precision Error (MAPE), since it is robust with near to zero values. The RMSE for a specific period forecasting (e.g. 5-min. forecasting) is defined by (1).

Table 3 depicts the predicted traffic values for all benchmarked techniques and for multiple five-minutes steps, up to 12 steps ahead (next 5, 10, 15, , 60 minutes). Traffic is expressed as average travel time (min/km). Figure 2 shows the average RMSE for different forecasting periods and for all roads in the Berlin dataset. As it seems from Table 3 the version of the proposed method (SDST) which uses the feature set FS3 and the sophisticated random number generator outperforms both all benchmark methods and the other versions (FS1 and FS2) of our technique when forecasting is concerned at each forecasting step.

This is also illustrated in Figure 2, which shows the comparison between all five benchmark algorithms and the three variations of SDST based on the average RMSE, i.e. the mean value of RMSE of all steps ahead and for the whole network.

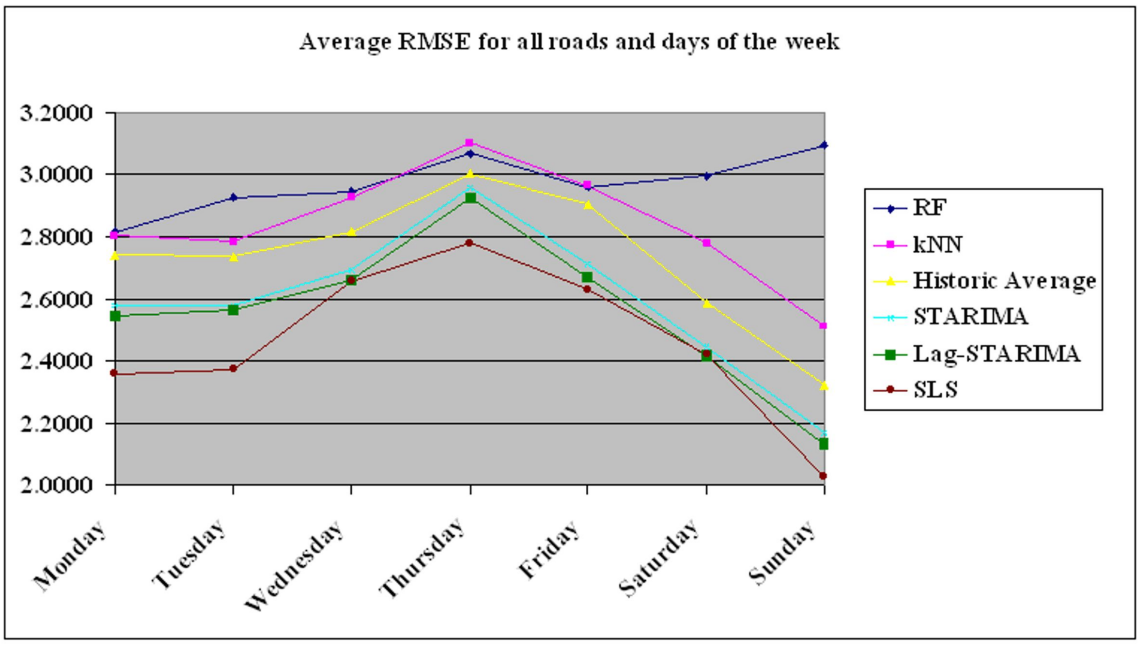


Figure 2: Comparison of the implemented algorithm with benchmark methods in terms of RMSE

3 Time-Dependent Shortest Paths

Distance oracles are succinct data structures encoding shortest path information among a carefully selected subset of pairs of vertices in a graph. The encoding is done in such a way that the oracle can efficiently answer shortest path queries for arbitrary origin-destination pairs, exploiting the pre-processed data and/or local shortest path searches. A distance oracle is exact (resp. approximate) if the returned distances by the accompanying query algorithm are exact (resp. approximate). A bulk of important work (e.g., [33, 32, 28, 29, 34, 35, 7]) is devoted to constructing distance oracles for *static* (i.e., *time-independent*), mostly undirected networks in which the arc-costs are fixed scalars, providing trade-offs between the oracle's space and query time and, in case of approximate oracles, also of the stretch. For an overview of distance oracles for static networks, the reader is deferred to [31] and references therein.

Considerable experimental work on routing in large-scale road networks has also appeared in recent years, with remarkable achievements that have been demonstrated on continental-size road-network instances. The goal is again to preprocess the distance metric and then propose query algorithms (known as *speedup techniques* in this framework) for responding to shortest path queries in time that is several orders of magnitude faster than a conventional Dijkstra run. An excellent overview of this line of research is provided in [8]. Once more, the bulk of the literature concerns static distance metrics, with only a few exceptions (e.g., [10, 17, 46]) that will be discussed later in more detail.

Modelling the Time-Variance of the Cost Metric. In many real-world applications, the arc costs may vary as functions of time (e.g., when representing the variability of arc traversal times, temporal unavailability of a particular arc, etc.) giving rise to *time-varying* network models. A striking example is route planning in road networks where the travel-time for traversing an arc uv (modelling a road segment) depends on the temporal traffic or availability conditions while attempting to traverse uv, and thus on the departure time from its tail u. Consequently, the min-cost path from an origin o to a destination d may vary with the departure-time t_o from the origin. In this work, we consider the *time-dependent network model*, in which every arc uv comes with an arc-traversal-time function $D[uv]$, whereas each path-traversal-time function is simply the *composition* of the corresponding arc-traversal-time functions of its constituent arcs. The *Time Dependent Shortest Path* (TDSP) problem concerns computing an $od-$path attaining the *earliest arrival time* at d, for an arbitrary triple (o, d, t_o) of an origin-destination pair of vertices $(o, d) \in V \times V$ and departure-time $t_o \in \mathbb{R}$ from the origin, in a time-dependent network model $(G = (V, A), (D[a] : \mathbb{R} \to \mathbb{R}_{>0})_{a \in A})$. The problem has been studied since a long time ago (see e.g., [12, 20, 47]). The shape of arc-travel-time functions and the waiting policy at vertices may considerably affect the tractability of the problem [47]. It is customary to consider as arc-travel-time functions the *continuous, piecewise linear (pwl) interpolants* of periodically sampled arc-travel-times. Regarding the waiting policy, a crucial assumption is that each arc obeys the *FIFO property*, according to which the earliest-arrival-time function of an arc uv is an increasing function of its departure time t_u from the tail u. Non-FIFO policies may lead to **NP**$-$hard cases [30]. On the other hand, in FIFO network models in which all the arc-travel-time functions possess the FIFO property, there is no need for waiting at either the origin or at intermediate nodes of the chosen path. Then, the problem can be solved in polynomial time by a straightforward variant of Dijkstra's algorithm (we call it `TDD`), which relaxes arcs by computing the arc costs "on the fly", when settling their tails [20].

Apart from the theoretical challenge, the time-dependent network model with FIFO-abiding, continuous, pwl arc-travel-time functions, is also much more appropriate with respect to handling the historic traffic data that the route planning vendors have to digest in order to provide their customers with fast route plans within milliseconds. For example, TomTom's *LiveTraffic* service[1] provides real-time estimations of average travel-time values, collected by periodically sampling the average speed of each road segment in a city, using the connected cars to the service as sampling devices. The crux is how to exploit all this historic traffic information in order to provide *efficiently* route plans that will adapt to the departure-time from the origin.

Related Work. Until recently, most of the previous work on the time-dependent shortest path problem concentrated on computing an optimal origin-destination path providing the earliest-arrival time at destination when departing at a *given* time from the origin, neglecting the computational complexity of providing succinct representations of the entire earliest-arrival-time (or equivalently for FIFO networks, shortest-travel-time) *functions* for *all* departure-times from the origin. Such representations, apart from allowing rapid answers to several queries for selected origin-destination pairs but for varying departure times, would also be valuable for the construction of *travel-time summaries* (a.k.a. *route planning maps*, or *search profiles*) from central vertices (e.g., *landmarks* or *hubs*) towards other vertices in the network, providing a crucial ingredient for the construction of oracles to support real-time responses to arbitrary queries $(o, d, t_o) \in V \times V \times \mathbb{R}$.

The complexity of succinctly representing earliest-arrival-time functions was first questioned in [13, 15, 14], but was solved only recently by a seminal work [22]. In particular, it was shown that, for FIFO-abiding pwl arc-travel-time functions, the problem has space-complexity $(1 + K) \cdot n^{\Theta(\log n)}$ for a *single* origin-destination pair, where n is the number of vertices and K is the total number of breakpoints of all the continuous, pwl arc-travel-time functions. Polynomial-time algorithms (or even PTAS) for constructing *point-to-point* $(1 + \varepsilon)$-approximate shortest-travel-time functions are provided in [22, 16], delivering point-to-point travel-time values at most $1 + \varepsilon$ times the true

[1] `http://www.tomtom.com/livetraffic/`

values. These functions indeed possess *succinct representations*, since they require only $\mathcal{O}(1+K)$ breakpoints per origin-destination pair. It is also easy to verify that K could be substituted by the number K^* of *concavity-spoiling* breakpoints of the arc-travel-time functions (i.e., breakpoints at which the arc-travel-time slopes increase). Of course, the succinctness in this representation heavily depends on the value of K^*. E.g., for $K^* \in \mathcal{O}(\text{polylog}(n))$, clearly these point-to-point approximation methods would work very well. Things become harder though for instances with more concavity-spoiling breakpoints, e.g. when $K^* \in \Omega(n)$.

Due to the above mentioned hardness of providing succinct representations of exact shortest-travel-time functions, the only realistic alternative is to use approximations of these functions for computing (in a preprocessing phase) *travel-time summaries* from properly selected vertices to all other vertices in the network, which is a crucial ingredient for constructing distance oracles in time-dependent networks.

Exploiting a PTAS (such as that in [22]) for computing travel-time summaries, one could provide a trivial oracle with query-time complexity $Q \in \mathcal{O}(\log\log(K^*))$, at the cost of an exceedingly high space-complexity $S \in \mathcal{O}\big((1+K^*)\cdot n^2\big)$, by precomputing and storing travel-time summaries from all possible origins. At the other extreme, one might use the minimum possible space complexity $S \in \mathcal{O}(n+m+K)$ for just storing the input, at the cost of suffering a query-time complexity $Q \in \mathcal{O}(m + n\log(n)[1+\log\log(1+K_{\max})])$ (i.e., respond to each query by running `TDD` in real-time using a predecessor search structure for evaluating continuous, pwl functions). $K_{\max}$ denotes the maximum number of breakpoints in an arc-travel-time function. The main challenge for a time-depenent oracle is thus to smoothly close the gap between these two extremes, i.e., to achieve a better (e.g., *sublinear*) query-time complexity, while consuming smaller space-complexity, e.g., $o\big(n^2\big)$, for succinctly representing travel-time summaries, while enjoying a small, e.g., close to 1, approximation guarantee (stretch factor). It would also be crucial to avoid the dependence on the amount of discncavity in the travel-time metric, as expressed by the value of K^*, at least for instances in which $K^* \in \Omega(n)$.

Providing distance oracles for time-dependent networks with *provably* good approximation guarantees, small preprocessing-space complexity and sublinear query time complexity, has only been recently investigated in [23, 24]. In particular, the *first* approximate distance oracle for sparse directed graphs with time-dependent arc-travel-times was presented in [24], providing $(1+\sigma)-$approximate travel-times in query-time that is *sublinear* in the network size, and preprocessing time and space that are *subquadratic* in the network size, when the total number of concavity-spoiling breakpoints in the instance is sufficiently small, e.g. when $K^* \in \mathcal{O}(\text{polylog}(n))$. The oracle uses a novel *one-to-all* method (called *Bisection* – `BIS`) to produce $(1+\varepsilon)-$approximate landmark-to-vertex travel-time summaries, for a randomly selected landmark set. It also guarantees either constant approximation ratio (a.k.a *stretch*) via the `FCA` query algorithm, or stretch at most $1+\sigma = 1+\varepsilon\cdot(1+\varepsilon/\psi)^{r+1}/[(1+\varepsilon/\psi)^{r+1}-1]$ via the `RQA` query algorithm, where ψ is a fixed constant depending on the characteristics of the arc-travel-time functions but is independent of the network size, and $r \in \mathcal{O}(1)$ is the recursion depth of `RQA`. In [23], another oracle is proposed, providing both constant and $(1+\sigma)-$approximate travel-times in query-time that is *sublinear* in the network size, and preprocessing time and space that are *subquadratic* in the network size, independently of the amount of disconcavity K^* in the network instance at hand. This is achieved by combining `BIS` with another *one-to-all* method (called *Trapezoidal* – `TRAP`) to produce $(1+\varepsilon)-$approximate landmark-to-vertex travel-time summaries.

A few time-dependent variants of well-known speedup techniques for road networks have also appeared in the literature (e.g., [10, 17, 46]). All of them were experimentally evaluated on synthetic time-dependent instances of the European and German road networks, with impressive performances. For example, in [10] methods are provided that respond to arbitrary queries of the German road network (4.7 million vertices and 10.8 million arcs) in less than $1.5ms$ and preprocessing space requirements of less than 1GB. A point-to-point travel-time summary (a.k.a. search profile) can also be constructed in less than $40ms$, when the departure times interval is a single

day. For point-to-point approximate travel-time summaries, with experimentally observed stretch at most 1%, the construction time is less than $3.2ms$. Their approach is based on the so-called *time-dependent Contraction Hierarchies* [36], along with several heuristic improvements both on the preprocessing step and on the query method.

Our Contribution. Our goal in this work is to provide a thorough experimental evaluation of the time-dependent distance oracles that were proposed and analysed in [24]. The main obstacle towards this direction is the dependence of the required preprocessing time and space on the number K^* of concavity-spoiling breakpoints in the raw traffic data.

Inspired by the theoretical analysis of [23], our first contribution is to propose a new time-dependent distance oracle whose preprocessing phase for computing landmark-to-vertex approximate travel-time summaries is solely based on a new approximation technique [23], the *trapezoidal* (`TRAP`) method. This method is significantly simpler than `BIS` and reduces dramatically the required space. In particular, `TRAP` avoids any kind of dependence on the number K^* of concavity-spoiling breakpoints, which are completely neglected during the preprocessing and need not be computed at all. Based on `TRAP`, we build new time-dependent distance oracles, which preprocess landmark-to-vertex approximate travel-time summaries for various landmark sets, and again employ the `FCA` and `RQA` query algorithms that were proposed in [24]. Additionally, we propose another quite simple query algorithm, `FCA`$^+$. Although the theoretical guarantee of its stretch factor is analogous to that of `FCA`, in practice it behaves very well, sometimes even better than `RQA`.

Our second contribution is an extensive experimental study of the above mentioned query algorithms for six different landmark sets, achieving remarkable speedups over `TDD` on truly real-world time-dependent data sets. In particular, we conduct our experimental evaluation on the historic traffic data for the city of Berlin, kindly provided to us by TomTom within [21]. The input instance is a directed graph with $478,989$ vertices and $1,134,489$ arcs. The provided raw traffic data for the arcs were stored as integer values for two different levels of resolution, one considering $10.3msec$, and another using $2.64sec$, as the time unit. We created six different landmark sets with 1000 or 2000 landmarks, which were chosen either randomly, or as the boundary vertices of appropriate METIS [5] or KaHIP [6] partitions of the Berlin graph. The speedups that we observed for our query algorithms over the average time of a `TDD` run, vary from 397 times (using 1000 randomly chosen landmarks and `FCA`), to 723 times (using 2000 randomly chosen landmarks and `FCA`), for $10.3ms$ resolution in the approximate travel-time summaries. In both cases the average relative error is less than 1.634%. Analogous speedups are observed if our quality measure is not the computational time, but the (machine-independent) number of settled vertices (a.k.a. Dijkstra rank) of the query algorithms. The best possible observed relative error is indeed much better than the theoretical bounds provided by the analysis of the query algorithms. In particular, it is as small as 0.382% for 1000 KaHIP landmarks, or 0.298% for 2000 KaHIP landmarks, for $10.3ms$ resolution in the approximate travel-time summaries. The corresponding speedups are 38 for the former, and 118 for the latter.

If we focus on the absolute response times, we manage to provide responses (via `FCA`) to arbitrary queries, in times less than $0.4ms$ for all landmark sets that we used, with relative error no more than 2.201%. For relative error at most 0.701%, we can provide answers in no more than $1.345ms$ using `FCA`$^+$, for all the considered landmark sets.

As for the preprocessed data, we create and succinctly store roughly $300K$ approximate travel-time summaries from a given landmark, in average sequential time less than $40sec$. That is, the amortized sequential time per approximate travel-time summary is no more than $0.134ms$.

Finally, with respect to the eco-footprint of the suggested routes we observed that our query algorithms, despite being significantly faster than `TDD`, they achieve not only extremely good approximation guarantees for the travel-time metric, but are also comparable with `TDD` by means of eco-footprint measurements. Of course, both the proposed routes by our query algorithms and the ones proposed by `TDD` are typically suboptimal for the eco-footprint metric, since this is not their

optimization criterion. On the other hand, the eco-footprint deviation of all these algorithms is not dramatic (roughly speaking, up to 15% deviation), whereas the average deviation of an optimum route with respect to eco-footprint has a much more significant deviation (more than 27%) from the optimal route with respect to the travel-time metric.

3.1 Preliminaries

We consider directed graphs $G = (V, A)$ with $|V| = n$ vertices and $|A| = m$ arcs, where each arc $a \in A$ is accompanied with a continuous, periodic, piecewise linear (pwl) *arc-travel-time* (or *arc-delay*) function defined as follows: $\forall k \in \mathbb{N}, \forall t \in [0, T),\ D[a](kT+t) = d[a](t)$, where $d[a] : [0, T) \to [1, M_a]$ such that $\lim_{t\uparrow T} d[a](t) = d[a](0)$, for some fixed integer M_a denoting the maximum possible travel time ever observed at arc a. Notice that the minimum arc travel time value in the entire network is also normalized to 1. Each arc-travel-time function $D[a]$ can be represented succinctly as a list of K_a breakpoints defining $d[a]$. Let $K = \sum_{a\in A} K_a$ be the number of breakpoints to represent all of them, $K_{\max} = \max_{a\in A} K_a$, and K^* be the number of *concavity-spoiling* breakpoints, i.e., those in which the arc-travel-time slopes increase. Clearly, $K^* \leq K$, and $K^* = 0$ for *concave* pwl arc-travel-time functions.

The *arc-arrival-time* functions are defined as $Arr[a](t) = t + D[a](t), \forall t \in [0, \infty)$. An assumption that we make is that each arc-arrival-time function is strictly increasing, in order to satisfy the strict FIFO property. The *path-arrival-time* function of a given path $p = \langle a_1, \ldots, a_k\rangle$ in G (represented as a sequence of arcs) is defined as the composition of the arc-arrival-time functions for the constituent arcs of p: $Arr[p](t) = Arr[a_k](Arr[a_{k-1}](\cdots(Arr[a_1](t))\cdots))$. The *path-travel-time* function is then $D[p](t) = Arr[p](t) - t$. Finally, between any origin-destination pair of vertices, $(o, d) \in V \times V$, $\mathcal{P}_{o,d}$ denotes the set of all $od-$paths in G, and the *earliest-arrival-time / shortest-travel-time* functions are defined as follows: $\forall t_o \geq 0$, $Arr[o, d](t_o) = \min_{p\in\mathcal{P}_{o,d}} \{Arr[p](t_o)\}$ and $D[o, d](t_o) = \min_{p\in\mathcal{P}_{o,d}} \{D[p](t_o)\} = Arr[o, d](t_o) - t_o$.

For any arc $a = uv \in A$ and any departure-times subinterval $[t_s, t_f) \subseteq [0, T)$, we consider the free-flow and maximally-congested travel-times for this arc, defined as follows:

- *Free-flow* arc-travel-time:

$$\underline{D}[uv](t_s, t_f) := \min_{t_u\in[t_s,t_f)} D[uv](t_u)\,.$$

- *Maximally-congested* arc-travel-time:

$$\overline{D}[uv](t_s, t_f) := \max_{t_u\in[t_s,t_f)} D[uv](t_u)\,.$$

We also denote $\overline{D}[uv] := \overline{D}[uv](0, T)$ and $\underline{D}[uv] := \underline{D}[uv](0, T)$. When $[t_s, t_f) = [0, T)$, we refer to the (static) *free-flow* and *full-congestion* travel-time metrics $\underline{D}$ and $\overline{D}$, respectively. These definitions also extend naturally to path-travel-times and shortest-travel-times between arbitrary origin-destination pairs of vertices.

For a point $(o, t_o) \in V \times [0, T)$ and $\beta \in \mathbb{N}$, let $B[o](t_o; \beta)$ be the set of the first β vertices settled by `TDD`, when growing a ball from (o, t_o). Analogously, $\underline{B}[o](\beta)$ and $\overline{B}[o](\beta)$ are the corresponding sets under the free-flow and fully-congested metrics $\underline{D}$ and $\overline{D}$, respectively.

For an arbitrary pair $(o, d) \in V \times V$ of origin-destination vertices, a succinctly represented $(1+\varepsilon)$-*upper-approximation* of $\Delta[o, d]$, is a continuous pwl function, hopefully with a *small* number of breakpoints, such that $\forall t_o \geq 0,\ D[o, d](t_o) \leq \Delta[o, d](t_o) \leq (1 + \varepsilon) \cdot D[o, d](t_o)$.

We adopt two assumptions from [24] and one additional assumption from [23], on the kind of shortest-travel-time functions that may appear in the time-dependent network instance at hand. All of them are quite natural and justified in urban-traffic road networks. Indeed, we conducted an experimental analysis on the real-world instance of Berlin that we had at our disposal, which

verified the validity of the assumptions. Technically, these assumptions allow the *smooth transition* from static metrics on undirected graphs towards time-dependent metrics on directed graphs. For a more thorough justification, the reader is deferred to [24, 23].

The first assumption asserts that the partial derivatives of the shortest-travel-time functions between any origin-destination pair are bounded in a fixed interval $[\Lambda_{\min}, \Lambda_{\max}]$.

Assumption 3.1 (Bounded Travel-Time Slopes) *There are* constants $\Lambda_{\min} \in [0, 1)$ *and* $\Lambda_{\max} \geq 0$ *s.t.:* $\forall (o, d) \in V \times V,\ \forall t_1 < t_2,\ \left(D[o, d](t_1) - D[o, d](t_2)\right) / (t_1 - t_2) \in [-\Lambda_{\min}, \Lambda_{\max}]$.

It is mentioned that the lower-bound of -1 in the shortest-travel-time function slopes is indeed a direct consequence of the strict FIFO property, which is typically assumed to hold in several time-dependent networks and allows for the use of time-dependent variants of classical shortest-path computation techniques, such as Dijkstra's and Bellman-Ford algorithms. Our experimental analysis on the historic traffic data for the city of Berlin, in which the maximum value of $\Lambda_{\max}$ in a series of $10,000$ randomly chosen origin-destination pairs was always less than 0.19.

The second assumption asserts that for any given departure time, the shortest-travel-time from o to d is not more than a *constant* $\zeta \geq 1$ times the shortest-travel-time in the opposite direction (but not necessarily along the reverse path). This is quite natural in road networks. E.g., it is most unlikely that a trip in one direction is more than, say, 10 times longer than the trip in the opposite direction for the same departure time. The assumption was also confirmed by our experimental analysis on the historic traffic data for the city of Berlin, in which the maximum value of ζ in a series of 10000 randomly chosen origin-destination pairs was always less than 1.5.

Assumption 3.2 (Bounded Opposite Trips) *There is a constant* $\zeta \geq 1$ *such that:* $\forall (o, d) \in V \times V,\ \forall t \in [0, T),\ D[o, d](t) \leq \zeta \cdot D[d, o](t)$.

One last assumption concerns the relation of the Dijkstra ranks (i.e., number of settled vertices, up to termination) of cocentric balls in the network, with respect to the (static) *free-flow* metric implied by the time-dependent instance at hand:

Assumption 3.3 (Growth of Free-Flow Balls) *For any vertex* $\ell \in V$ *and positive integer* $F \in \mathbb{N}$*, assume growing a free-flow Dijkstra ball* $\underline{B}[\ell](F)$ *around* ℓ*, of size* F*. Let* $\underline{R}[\ell] = \max\{\underline{D}[\ell, v] : v \in \underline{B}[\ell](F)\}$ *be the* free-flow radius *in* $\underline{B}[\ell](F)$*. Also let* $\overline{R}[\ell] = \max\{\overline{D}[\ell, v] : v \in \underline{B}[\ell](F)\}$ *be the* full-congestion radius *in* $\underline{B}[\ell]$*. Finally,* $\underline{B}' = \{v \in V : \underline{D}[\ell, v](0, T) \leq \overline{R}[\ell]\}$ *is the free-flow ball with radius* $\overline{R}[\ell]$ *around* ℓ*. Then it holds that* $|\underline{B}'[\ell]| \in \mathcal{O}(F \cdot \text{polylog}(F))$.

This assumption has also been experimentally tested in the Berlin instance, for various initial ball sizes. In all cases the scaling factor of the ball size was less than 2.

3.2 Time-Dependent Oracles

In [23] time-dependent distance oracles are proposed and theoretically analysed, which preprocess the travel-times metric using both the `BIS` method (for nearby destinations) proposed in [24] and the novel `TRAP` method (for faraway destinations) to approximate shortest travel-time functions, and then use one of two query algorithms (`FCA` or `RQA`) for efficiently responding to arbitrary queries. The novelty of these oracles is that they assure subquadratic storage space and sublinear query complexity, irrespectively of the degree of disconcavity of the travel-time metric, measured by the value of K^*. In this work we experimentally evaluate these oracles exploiting exclusively the `TRAP` method for creating travel-time summaries, and also experiment with an additional query algorithm, called `FCA`$^+$, that we propose.

All the oracles start by selecting a subset $L \subset V$ of *landmarks*. This can be done either randomly (e.g., by deciding for each vertex i.u.r with probability $\rho \in (0, 1)$ whether it belongs to L), or by selecting L from the vertices in the cut sets provided by some graph partitioning algorithm. In this work we consider appropriate METIS and KaHIP partitions of the Berlin graph. After L is

determined, a preprocessing phase is performed in which, $\forall \ell \in L$ and $\forall v \in V$, all ℓ-to-v $(1+\varepsilon)-$upper-approximating travel-time functions (we call them *approximate travel-time summaries*) are computed and stored, based on the `TRAP` method. Consequently, one of the three different query algorithms, `FCA`, `FCA`$^+$, or `RQA` is used for providing in sublinear time *guaranteed* approximations of the actual shortest travel time values, for arbitrary queries $(o, d, t_o) \in V \times V \times [0, T)$. In a final step, a path-construction routine is run to provide an *od*-path with actual path-travel-time at most equal to the predicted one. In this section, we briefly review the above mentioned ingredients of our oracles.

3.2.1 Approximate Travel-Time Functions via the Trapezoidal Method

We briefly present here the novel preprocessing step of our oracles which, based on the `TRAP` method, constructs $(1+\varepsilon)-$upper-approximations of shortest travel-time functions (cf. [23] for a detailed presentation and analysis). The performance of this new preprocessing phase is practically *independent* of the degree of disconcavity of the instance as expressed by K^*.

`TRAP` splits the entire period $[0, T)$ into small, consecutive subintervals of length $\tau > 0$ each. It then provides a crude approximation of the unknown shortest-travel-time functions in each interval, solely based on Assumption 3.1 concecning the boundedness of the shortest travel-time slopes in the instance. After sampling the travel-time values of each destination $v \in V$, for a given origin $u \in V$, we consider each pair of consecutive sampling times $t_s < t_f$ and the semilines with slopes $\Lambda_{\max}$ from t_s and $-\Lambda_{\min}$ from t_f. The considered upper-approximating function $\overline{D}[u, v]$ within $[t_s, t_f)$ is then (a refinement of) the lower-envelope of these two lines. Analogously, a lower-approximating function $\underline{D}[u, v]$ is the upper-envelope of the semilines that pass through t_s with slope $-\Lambda_{\min}$, and from t_f with slope $\Lambda_{\max}$. Depending on the value of the absolute error and the minimum possible value of $\underline{D}[u, v]$ in this interval, we can decide whether $\overline{D}[u, v]$ is a $(1+\varepsilon)$-upper-approximating function of $D[u, v]$. Any destination vertex that has such a $(1+\varepsilon)$-upper-approximating function for each subinterval of $[0, T)$, clearly has a $(1+\varepsilon)$-upper-approximating function for the entire period as well. The proof of correctness of `TRAP` is provided in [23].

The problem with the trapezoidal approximation is that, by construction, it is not possible to provide $(1+\varepsilon)$-approximate travel-time functions for "nearby" destination vertices, which are too close to the origin. In [23] these "nearby" vertices of each landmark are either handled by the `BIS` method [24], or are left to be handled by local `TDD` searches "on the fly". Here we resolve this issue exclusively with `TRAP`, starting with a large subinterval length, and then recursively dividing by 2 the lengths of those subintervals containing vertices which have not been sufficiently approximated yet, until all landmark-to-vertex $(1+\varepsilon)$-approximate travel-time summaries have been successfully created. This proved to be extremely space- and time-efficient in practice.

3.2.2 Query Algorithms

For efficiently responding to arbitrary origin/destination/departure-time queries (o, d, t_o), three approximation algorithms are considered. The first one, called `FCA`, is a simple *sublinear*-time constant-approximation algorithm, which works as follows. It grows a ball $B_o \equiv B[o](t_o) = \{x \in V : D[o, x](t_o) \leq D[o, \ell_o](t_o)\}$ from (o, t_o), by running `TDD` until either d or the closest landmark $\ell_o \in \arg\min_{\ell \in L}\{D[o, \ell](t_o)\}$ is settled. It then returns either the exact travel-time value, or the approximate travel-time value via ℓ_o, achieving a $1+\varepsilon+\psi$ approximation guarantee as was shown in [24], where ψ is a constant depending on ε, ζ, and $\Lambda_{\max}$, but not on the size of the network.

The second query algorithm, called `FCA`$^+$, is a variant of `FCA` which keeps growing a `TDD` ball from (o, t_o) until either d or a given number N of landmarks is settled. `FCA`$^+$ returns the smallest via-landmark approximate travel-time value, along all these settled landmarks. The approximation guarantee is the same as that of `FCA`, but in practice it performs quite well, in certain cases even better than `RQA`, as it will be demonstrated in the experimental evaluation.

The third algorithm, called RQA, is indeed a PTAS for computing shortest travel-time functions. In particular, it improves the approximation guarantee of the chosen $od-$path to $1+\sigma = 1+\varepsilon\cdot[(1+\varepsilon/\psi)^{r+1}]/[(1+\varepsilon/\psi)^{r+1}-1]$, by exploiting carefully a number $r \in \mathbb{N}$ (called the *recursion budget*) of recursive accesses to the preprocessed information, each of which produces (via calls to FCA) additional candidate $od-$paths sol_i. RQA works as follows. As long as the destination vertex within the explored area around the origin has not yet been discovered, and there is still some remaining recursion budget, it "guesses" (by exhaustively searching for it) the next vertex w_k of the boundary set of touched vertices (i.e., still in the priority queue) along the unknown shortest $od-$path. Then, it grows an outgrowing TDD ball from the new center $(w_k, t_k = t_o + D[o, w_k](t_o))$, until it reaches the closest landmark ℓ_k to it, at travel-time $R_k = D[w_k, \ell_k](t_k)$. This new landmark offers an alternative $od-$path $sol_k = P_{o,k} \bullet Q_k \bullet \Pi_k$ by a new application of FCA, where $P_{o,k} \in SP[o, w_k](t_o)$, $Q_k \in SP[w_k, \ell_k](t_k)$, and $\Pi_k \in ASP[\ell_k, d](t_k + R_k)$ is the approximate suffix subpath provided by the preprocessed data of the oracle (in case of the TRAP scenario, it has to be computed "on-the-fly". Observe that sol_k uses a longer (optimal) prefix-subpath P_k which is then completed with a shorter approximate suffix-subpath $Q_k \bullet \Pi_k$. This is exactly the main idea behind its analysis for improving the provided approximation guarantee. RQA finally responds with a $(1+\sigma)-$approximate travel-time to the query in *sublinear* time, for any constant $\sigma > \varepsilon$.

A more detailed presentation of FCA and RQA, along with the proofs of correctness and their time complexities, are provided in [24]. As for the approximation guarantee of FCA$^+$, it is straightforward to observe that, at least theoretically, it is as small as that of FCA, whereas its time complexity is comparable to that of RQA.

3.2.3 Heuristic Improvements

The TRAP approximation method introduces at least one intermediate (possibly two) breakpoint per interval that does not yet meet the required approxmation guarantee. This is certainly unnecessary for intervals in which the actual shortest-travel-time functions are almost constant. To avoid the blow-up of the preprocessing space required, we heuristically make an arbitrary "guess" that we have to deal with an "almost constant" shortest-travel-time function $D[\ell, v]$ within a given interval $[t_s, t_f)$, if the following holds: $D[\ell, v](t_s) = D[\ell, v](t_f) = D[\ell, v]\left((t_s + t_f)/2\right)$. This is justified by the fact that $D[\ell, v]$ is a continuous pwl function, along with the fact that already $t_f = t_s + \tau$ for some small value $\tau > 0$. Of course, one could easily construct artificial examples for which this criterion is violated, e.g., by providing a properly chosen periodic function with period τ. On the other hand, one can easily tackle this by considering a *randomly perturbed* sampling period $\tau + \delta$, for some arbitrarily small but positive random variable δ.

Another improvement that we adopt is that, rather than splitting the entire period $[0, T)$ in a flat manner into *equal-size* intervals, we start with a coarse partitioning based on a large length and then in each inteval and for each destination vertex we check for the provided approximation guarantee by TRAP. All the vertices which are already satisfied by this guarantee with respect to the current interval, become inactive for this and all its subsequent subintervals. We then proceed by splitting in the middle every subinterval that contains at least one still active destination vertex, and repeating the check for all active vertices within the new subintervals.

3.3 Experimental Evaluation

The purpose of this section is to provide all the details of the experimental evaluation that we conducted on the three query algorithms that we propose, and for the TRAP-based preprocessing phase executed on six properly chosen landmark sets, of either $1,000$ or $2,000$ landmarks each.

3.3.1 Preprocessing The Road Instance

The Berlin instance, kindly provided by TomTom within [21], consists of a directed graph with $478,989$ vertices and $1,134,489$ arcs. We focused only on the strongly connected component of this input graph, consisting of $473,253$ vertices and $1,126,468$ arcs. $924,254$ of the arcs have constant arc-travel-times. For the remaining $202,214$ arcs, continuous pwl arc-travel-time functions are provided, concerning an entire weekday (Tuesday). The maximum arc-travel-time slope is 0.0166667, whereas the minimum slope is -0.0133333. The succinct representation of these functions requires a total number of $3,234,213$ breakpoints. We substituted each maximal path in the network consisting of intermediate vertices with no intersections (i.e., would have degree 2 in the simple, undirected version of the graph), with a single shortcut arc of arc-travel-time function equal to the corresponding (exact) path-travel-time function. This resulted in a reduced graph consisting of $299,693$ vertices and $950,504$ arcs.

We generated two data formats suitable as input for our query algorithms. The first concerns the arc-travel-time functions and the second the preprocessed travel-time summaries (i.e., landmark-to-vertex $(1+\varepsilon)$-approximate shortest travel-time functions).

Arc-Travel-Time Functions. The raw-traffic data set is provided as a collection of arrays with average speed estimations. Each row of such an array corresponds to a particular arc indicating a road segment. The columns provide a partition of the entire one-day period into 288 timeslots of 5-minutes each. The arc-travel-time value of an arc $a = uv$ for a timestlot i is computed as $length/[S_{(a,i)} \times (free\ flow\ speed)_a]$, where $free\ flow\ speed$ denotes the top speed that can be achieved with zero congestion along a, while $S_{(a,i)}$ denotes a scale factor dependent on the road traffic status of timeslot i. Therefore, for arc a a sequence $\langle$ (departure-time$_i$, arc-travel-time$_i$)$_{i\in[288]}$ $\rangle$ of breakpoints is created, where departure-time$_i$ is the starting point of the corresponding timeslot, and arc-travel-time$_i$ is the estimated time to traverse it when the departure time is exactly departure-time$_i$.

In order to avoid wasting space, for each arc and arc-travel-time value per timeslot, consecutive timeslots having the same arc-travel-time value were merged. Optionally, one could perform a broader merging of consecutive timeslots having absolute difference in arc-travel-time values less than a small constant (e.g. $< 1min$ resolution bound). However, in our experiments we chose to preserve the maximum possible resolution of the raw-traffic data. This proved to be extremely efficient by means of approximation guarantees, for different levels of resolution for the approximate travel-time summaries. The eventual space required for all the raw-traffic data provided as input, is roughly 225MB.

The arc-travel-time function $d[a](t)$ is simply the continuous, pwl interpolant of all the breakpoints corresponding to arc a. $D[a](t)$ is then the periodic repetition of $d[a](t)$.

Preprocessed Landmark Information. In order to create all the landmark-to-vertex $(1+\varepsilon)$-approximate shortest travel-time summaries, we call `TRAP`, which is a one-to-all approximation method, once per landmark. Upon completion of this preprocessing phase, we collect the $(1+\varepsilon)$-approximate travel-time summaries for all the landmark-vertex pairs in the Berlin graph. For each such pair $(\ell, v) \in L \times V$, we store a sequence $\langle$ ($Dep[\ell]_i$, $Arr[\ell, v]_i$)$_i$ $\rangle$ of breakpoints, where $Dep[\ell]_i$ denotes a departure-time from landmark ℓ and $Arr[\ell, v]_i$ denotes the corresponding earliest-arrival-time at v. The interpolation of all these breakpoints produces the overall $(1+\varepsilon)$-upper-approximating travel-time summary (continuous, pwl function) $\Delta[\ell, v](t)$. With $|L|$ landmarks and p breakpoints (on average) per approximate travel-time summary, the preprocessing space required for storing all the landmark-to-vertex approximate travel-time summaries is $\mathcal{O}(|L|pn)$.

Our approach is focused on achieving a cost-effective storage of these summaries, while keeping a sufficient precision. The key is that some specific features can be exploited in order to reduce the required space. The main observation is that, for a one-day time period, departure-times and arrival-times have a bounded value range. In particular, when the considered precision of the traffic

data is within seconds we handle time-values as integers in the range $[0\ ,\ 86,399]$, for milliseconds as integers in $[0\ ,\ 86,399,999]$, etc.

Any (real) time value within a single-day period, represented as a floating-point number t_f, can thus be converted to an integer t_i with fewer bytes and a given unit of measure. For a unit measure (or scale factor) s, the resulting integer is $t_i = \lceil t_f/s \rceil$. In this manner, t_i needs size $\lceil \log_2(t_f/s)/8 \rceil$ bytes. The division t_f/s has quotient π and remainder υ. Thus, $t_f = s \cdot \pi + \upsilon$ and $t_i = \lceil (s \cdot \pi + \upsilon)/s \rceil = \lceil \pi + \upsilon/s \rceil$, with $\upsilon < s$. Therefore, converting t_f to t_i results to an absolute error of at most $2s$. In the reverse process, for extracting the stored value, the conversion is $t_f^{'} = t_i \cdot s$. In our experiments, for storing the time-values of approximate travel-time summaries, we have considered two different resolutions:

(a) $2.64sec$ resolution, corresponding to a scale factor $s = 1.32$ (when counting time in seconds), requiring 2 bytes per time-value, and

(b) $10.3ms$ resolution, corresponding to a scale factor $s = 5.15$ (when counting time in milliseconds), requiring 3 bytes per time-value.

3.3.2 Experimental Setup

All algorithms were implemented using C++ (gcc, version 4.6.3). To support all graph-operations we used the PGL library [25]. All experiments were executed by a CPU of 3.40GHz×8, using 16GB of RAM, on Ubuntu 12.04 LTS. All our algorithms are executed sequentially. Exploitation of parallelism is left for future implementations and is anticipated to reduce dramatically the execution times, particularly for the preprocessing phase and the query algorithm `RQA` for which parallelism would apply quite naturally.

3.3.3 Measurements and Evaluation of Speedups and Approximation Guarantees

We now proceed with the presentation and discussion of our findings in the experimental evaluation that we conducted on the data set of Berlin, for the three query algorithms and the six landmark sets that we considered.

Preprocessing Phase: Creation of Approximate Travel Time Summaries. Our preprocessing phase took as input six different landmark sets for the Berlin graph: R_{1000} and R_{2000} correspond to $1,000$ and $2,000$ landmarks chosen uniformly at random from the entire vertex set. M_{1000} and M_{2000} correspond to $1,021$ and $2,072$ landmarks chosen as the boundary vertices of appropriate METIS partitions. K_{1000} and K_{2000} correspond to $1,016$ and $2,024$ landmarks chosen as the boundary vertices of appropriate KaHIP partitions.

For the production of the approximate travel-time summaries for each of the landmark sets, a total amount of less than 13 hours (for small sets) and 26 hours (for large sets) of sequential computational time was consumed. In particular, the average time per landmark, for producing its approximate travel-time summaries towards *all* possible destinations is less than $43sec$, and the amortized time for constructing a single landmark-to-vertex approximate travel-time summary is less than $0.1435ms$.

The required storage space is less than 35MB per landmark for $2.64sec$ resolution, and 55MB per landmark for the $10.3ms$ resolution.

Query Phase: Responding to Arbitrary Shortest-Path Queries. The query execution times and relative errors of the produced solutions, for all possible landmark sets and the two different resolutions that we consider for the approximate travel-time summaries, are presented in Tables 4 and 5. Moreover, Table 6 presents the speedups of the query algorithms, measured by the machine-independent criterion of Dijkstra-rank, i.e., the number of settled vertices during

execution. All reported values are averages over $1,000$ randomly chosen queries from the Berlin instance. We note that for RQA the recursion budget was set to 1. For fairness of comparison, the parameter N (number of landmarks) in FCA$^+$ was set equal to the number of landmarks settled by RQA.

It should be noted that for the query algorithms we only count the required computational time for providing an upper bound on the earliest-arrival-time at the destination. In particular, we exclude the time required for the construction time of a path with the discovered guarantee (which is anyway negligible) and the time required for accessing from the hard disk the approximate travel-time summaries of the involved landmarks. The latter is done for two reasons: First, we wish our comparison to be as independent as possible of the characteristics of the machine, and in particular of the size of the main memory. For example, the reported times would be as they appear in Tables 4 and 5 in exactly the same machine but with sufficiently large main memory. Second, our main quality measure is the achieved speedup versus the average performance of TDD. Clearly, TDD produces no disk I/O accesses when being executed, and the comparison would be misleading for the query algorithms, simply due to poor hardware characteristics. We wish to have a clear comparison of the algorithms themselves, which is irrelevant of the hardware platform.

Apart from query-times, we also report the observed *relative errors* of the produced solutions. The relative error for a given $od-$path p is the percentage of surplus from the exact shortest travel-time (as computed by TDD), i.e.:

$$100 \cdot [\text{travel time of } p - \text{shortest travel time from } (o, t_o) \text{ to } d]/[\text{shortest travel time from } (o, t_o) \text{ to } d]$$

With respect to the observed query times, in all cases FCA is the fastest query method, but with the highest relative error, compared to the other two methods. For example, it returns answers with relative error 1.634% in $0.195ms$ (i.e., a speedup more than 397 over the runtime of TDD), for R_{1000} and $10.3ms$-resolution. The response time for R_{2000} and $10.3ms$-resolution is $0.107ms$ (i.e., speedup more than 723) with relative error 1.065%. Similar performance is observed also for the cases of preprocessing with $2.64sec$-resolution. For the other two query algorithms, FCA$^+$ is always faster than RQA, the latter being at most two times slower than the former. This can be justified by the fact that FCA$^+$ grows a unique Dijkstra ball from the origin, and thus acts like a label-setting algorithm. On the other hand, RQA may visit and update the labels of the same vertices more than once, since at the second level of the recursion the labels of the settled nodes are not always shortest travel-times from the origin, but shortest travel-times via particular parents. On the other hand, it should be noted that RQA is amenable to parallelization due to its recursive flavor. This is anticipated to speedup significantly the average query time in forthcoming implementations of RQA, which will then be comparable to that of FCA.

With respect to the relative error, we observe that for all the random landmark sets FCA$^+$ provides smaller values, of 0.449% for 1000 random landmarks and 0.389% for 2000 random landmarks. For the rest of the landmark sets, RQA is the best option with respect to the relative error, achieving values 0.314% for 1000 KaHIP landmarks and 0.298% for 2000 KaHIP landmarks. That is, the oriented expansion of the Dijkstra tree provided by RQA performs better in cases of landmark sets created from well structured partitions, whereas the brute-force expansion of FCA$^+$ is better for randomly chosen landmarks in the network.

As for the machine-independent performance of Dijkstra-ranks (cf. Table 6), we observe that the reported average speedups of our query algorithms, compared to a typical TDD run, are even better. For example, using FCA on R_{1000} and R_{2000} produce speedups larger than 429 and 889 times respectively. This is indeed quite encouraging, since all the proposed query algorithms are based on label-settings, just as TDD, and this performance measure is truly machine independent.

A final remark is the sensitivity of our algorithms to the choice of resolution for the values of the approximate travel-time summaries that are created during the preprocessing phase. Observe that if the performance measure is the Dijkstra-rank, then the choice of resolution, which only affects the approximate values of the landmark-to-destination travel-times, is irrelevant of the rank measure,

because the Dijkstra balls grow over the raw traffic data for which we have preserved the maximum possible accuracy. Even when we account for computational times of the query algorithms, we observe that the difference in the relative errors is rather negligible, and in a few cases the coarser resolution of $2.64sec$ results in smaller relative-error values. This is due to the path reconstruction method that we use, which also takes into account the values of the approximate landmark-to-vertex travel-time values. The main reason for this insensitivity in the chosen resolution is that it is only the last part of the chosen path that is indeed affected, by only a small additive term of few seconds, or even milliseconds.

	TDD		FCA		FCA$^+$		RQA	
	Time (ms)	Rel.Error (%)	Time (ms)	Rel.Error (%)	Time (ms)	Rel.Error (%)	Time (ms)	Rel.Error (%)
R_{1000}	77.424	0	**0.195**	1.634	1.345	**0.449**	1.692	0.575
M_{1000}			**0.381**	2.201	1.313	0.698	2.349	**0.483**
K_{1000}			**0.362**	2.165	1.223	0.506	2.015	**0.382**
R_{2000}			**0.107**	1.065	0.71	**0.389**	0.771	0.445
M_{2000}			**0.152**	1.115	0.582	0.336	0.7	**0.314**
K_{2000}			**0.148**	1.405	0.599	0.367	0.655	**0.298**

Table 4: Query performances for $10.3ms$-resolution of the raw traffic data.

	TDD		FCA		FCA$^+$		RQA	
	Time (ms)	Rel.Error (%)	Time (ms)	Rel.Error (%)	Time (ms)	Rel.Error (%)	Time (ms)	Rel.Error (%)
R_{1000}	77.424	0	**0.198**	1.634	1.345	**0.449**	1.712	0.574
M_{1000}			**0.381**	2.199	1.287	0.7	2.09	**0.487**
K_{1000}			**0.348**	2.171	1.197	0.512	1.834	**0.381**
R_{2000}			**0.108**	1.065	0.694	**0.382**	0.769	0.442
M_{2000}			**0.156**	1.116	0.589	0.346	0.767	**0.314**
K_{2000}			**0.148**	1.401	0.591	0.366	0.721	**0.295**

Table 5: Query performances for $2.64sec$-resolution of the raw traffic data.

	TDD		FCA		FCA$^+$		RQA	
	Rank	Speedup	Rank	Speedup	Rank	Speedup	Rank	Speedup
R_{1000}	149397	1	**348**	**429.302**	2628	56.848	4261	35.061
M_{1000}			713	209.533	2517	59.355	5304	28.167
K_{1000}			657	227.393	2353	63.492	4660	32.059
R_{2000}			**168**	**889.268**	1251	119.422	1820	82.086
M_{2000}			252	592.845	1039	143.789	1646	90.764
K_{2000}			247	604.846	1002	149.099	1522	98.158

Table 6: Query performances with respect to the numbers of settled vertices.

3.3.4 Methodology and Measurements for Assessing the Eco-Footprint

We have calculated the CO_2 emissions of any computed route considering an average private car, in particular a 5-door Opel-Astra, which has an average fuel-consumption of 4.2 litres/ 100 km. Based on the information provided by the EN 16258:2012 – *Methodology for Calculation and Declaration of Energy Consumption and GHG Emissions of Transport Services (Freight and Passenger)*, published

by CEN[2], we considered the greenhouse gas (GHG) emissions (calculated as CO_2 equivalents) and specifically the well-to-wheels value, measured as $kg\ CO_2e\ /\ km$. Table 7, extracted from the EN 16258 standard, was used in our calculations. Since the fuel-consumption of the vehicle is considered

	Standardised energy consumption				Greenhouse gas emissions (calculated as CO_2 equivalents)			
	Tank-to-wheels (e_T)		Well-to-wheels (e_W)		Tank-to-wheels (g_T)		Well-to-wheels (g_W)	
	MJ/ kg	*MJ/ l*	*MJ/ kg*	*MJ/ l*	*kg CO_2e/ kg*	*kg CO_2e/ l*	*kg CO_2e/ kg*	*kg CO_2e/ l*
Petrol	43.2	32.2	50.5	37.7	3.25	2.42	3.86	2.88
Ethanol	26.8	21.3	65.7	52.1	0.00	0.00	1.56	1.24
Petrol E5 (5 vol.% Ethanol)	42.4	31.7	51.4	38.4	3.08	2.30	3.74	2.80
Petrol E10 (10 vol.% Ethanol)	41.5	31.1	52.2	39.1	2.90	2.18	3.62	2.72
Diesel	43.1	35.9	51.3	42.7	3.21	2.67	3.90	3.24
Biodiesel	36.8	32.8	76.9	68.5	0.00	0.00	2.16	1.92
Diesel D5 (5 vol.-% biofuel)	42.8	35.7	52.7	44.0	3.04	2.54	3.80	3.17
Diesel D7 (7 vol.-% biofuel)	42.7	35.7	53.2	44.5	2.97	2.48	3.76	3.15
Compressed natural gas	45.1	n/a	50.5	n/a	2.68	n/a	3.07	n/a
Liquefied petroleum gas	46.0	25.3	51.5	28.3	3.10	1.70	3.46	1.90
Jet kerosene[1)]	44.1	35.3	52.5	42.0	3.18	2.54	3.88	3.10
Heavy fuel oil (HFO)[2)]	40.5	39.3	44.1	42.7	3.15	3.05	3.41	3.31
Marine diesel oil (MDO)	43.0	38.7	51.2	46.1	3.24	2.92	3.92	3.53
Marine gas oil (MGO)	43.0	38.3	51.2	45.5	3.24	2.88	3.92	3.49

[1)] Without allowing for a possible higher effect on the climate from air traffic at cruising height. – [2)] HFO = Heavy fuel oil (heavy fuel oil for ships).

Source: EN 16258.

Example: a vehicle needs 406l to drive from A->B

TTW GHG emissions: $G_T = F \times g_T = 406\ l \times 2.67\ kg\ CO_2e/l = 1{,}084\ kg\ CO_2e$
WTW GHG emissions: $G_W = F \times g_W = 406\ l \times 3.24\ kg\ CO_2e/l = 1{,}316\ kg\ CO_2e$

Table 7: Factors for the calculation of energy consumption and greenhouse gas emissions (calculated as CO_2 equivalents) in accordance with EN 16258

to be an average (constant) value, the amount of the resulting CO_2 emissions depends solely on the length of the computed route. As a result, we adopt the GHG emissions of the shortest path with respect to the distance (rather than travel-times) metric, from an origin o to a destination d, as the baseline in our experiments.

The corresponding shortest-distance *od*-route is of course expected to provide the minimum eco-

[2] http://www.transport2020.org/newsitem/cen-publishes-european-standard-for-calculation-of-ghg-emissions

footprint, but on the other hand it is typically a suboptimal route with respect to travel-times. The average deviation of the eco-footprint for a proposed route from o to d is compared with respect to this baseline GHG emission, and it is provided for all the time-dependent algorithms that we experimentally tested, namely, `TDD`, `FCA`, `FCA`$^+$, and `RQA`. In all our experiments we consider Diesel to be the type of fuel used by the vehicle, which means that the well-to-wheel value is $gw = 3.24\text{CO}_2e$. In each case, the computation of the total CO_2e emissions of a path p turns out to be rather simple:

$$\text{CO}_2e(p) = \text{total fuel consumption} \cdot gw = distance(p) \cdot \text{fuel consumption per } km \cdot gw$$

Table 8 demonstrates the comparison of all these algorithms for the most prominent (with respect to the observed approximation guarantee with respect to travel-times) KaHIP K_{2000}landmark set. Table 9 demonstrates the comparison of all these algorithms for the most prominent (with respect to the observed approximation guarantee with respect to travel-times) KaHIP K_{2000}landmark set.

	Shortest-Distance-Dijkstra	`TDD`	`FCA`	`RQA`	`FCA`$^+$
time ($msec$)	77.583	92.883	0.151	0.747	0.616
Dijkstra rank	215,297	201,209	352	2,260	1,452
travel time deviation (%)	27.115	0	1.354	0.308	0.327
eco-footprint (CO_2e)	19.17	21.828	22.003	21.854	21.864
eco-footprint deviation (%)	0	14.729	16.017	15.006	15.061

Table 8: Eco-footprint reports for 1000 random queries and the K_{2000} landmark set.

For both landmark sets we observe that, as expected, the route achieving the (exact) shortest travel time, reported by `TDD`, is already suboptimal with respect to the baseline eco-footprint of the shortest-distance route. Quite interestingly, the proposed query algorithms `FCA`, `FCA`$^+$ and `RQA`, which definitely achieve extremely good approximations of travel times, also demonstrate performance with respect to the eco-footprint which is pretty close to that of `TDD`. On the other hand, the optimal route with respect to the eco-footprint is a rather prohibitively bad approximation of the optimum travel-times achieved by `TDD`.

	Shortest-Distance-Dijkstra	`TDD`	`FCA`	`RQA`	`FCA`$^+$
time ($msec$)	77.583	92.883	0.105	0.832	0.754
Dijkstra rank	215,297	201,209	235	2,567	1,766
travel time deviation (%)	27.115	0	1.209	0.384	0.402
eco-footprint (CO_2e)	19.17	21.828	21.916	21.847	21.858
eco-footprint deviation (%)	0	14.729	15.632	14.873	14.917

Table 9: Eco-footprint reports for 1000 random queries and the R_{2000} landmark set.

4 Fast, Dynamic and Highly User-Configurable Route Planning

We perform extensive experiments to determine the performance of Customizable Contraction Hierarchies on real world networks. For an algorithmic description, see Deliverable D2.2.1 and [44]. Our experiments show that the performance is completely independent of the metric used, as long as it is constant per weight. This makes it a prime technique to cope with complex personalized shortest path metrics that besides static travel time take vehicle restrictions, user preferences such aus avoid highways, current traffic conditions, or eco-friendly energy-consumption into account.

Table 10: Instances. We report the number of vertices and of directed arcs of the benchmark graphs. We further present the number of edges in the induced undirected graph.

Instance	# Vertices	# Arcs	# Edges
Karlsruhe	120 412	302 605	154 869
Europe	18 010 173	42 188 664	22 211 721

Figure 3: All vertices in the PTV-Europe graph.

4.1 Experiments

Compiler and Machine We implemented our algorithms in C++, using g++ 4.7.1 with -O3 for compilation. The customization and query experiments were run on a dual-CPU 8-core Intel Xeon E5-2670 processor (Sandy Bridge architecture) clocked at 2.6 GHz, with 64 GiB of DDR3-1600 RAM, 20 MiB of L3 and 256 KiB of L2 cache. The order computation experiments (see Table 11) were run on a single core of an Intel Core i7-2600K CPU processor.

Instances We consider two large instances of practical relevance (see Table 10): The Europe graph was made available by PTV[3] for the DIMACS challenge [43]. The vertex positions are depicted in Figure 3. It is the standard benchmarking instance used by road routing papers over the past few years. Note that besides roads it also contains a few ferries to connect Great Britain and some other islands with the continent. The Europe graph analyzed here is its largest strongly connected component (a common method to remove bogus vertices). It is directed, and we consider two different weights. The first weight is the travel time and the second weight is the straight line distance between two vertices on a perfect sphere. The Karlsruhe graph is a subgraph of the PTV graph for a larger region around Karlsruhe. We consider the largest connected component of the graph induced by all vertices with a latitude between 48.3° and 49.2°, and a longitude between 8° and 9°. Table 10 reports the instance sizes.

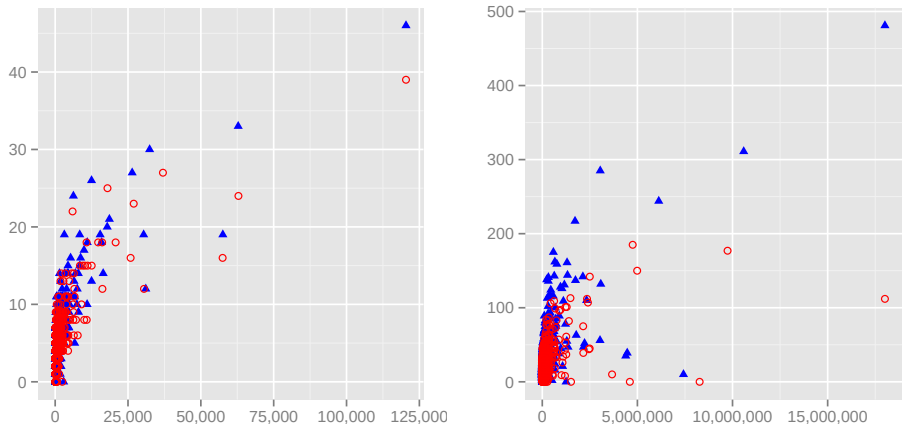
4.1.1 Orders

We analyze three different vertex orders: 1) The greedy order is an order in the spirit of [45]. 2) The Metis graph partitioning package contains a tool called `ndmetis` to create ND-orders. 3) KaHIP provides just graph partitioning tools. As far as we know tools to directly compute ND-orders are planned by the authors but not yet finished. We therefore implemented a very basic program on top of it. For every graph we compute 10 bisections with different random seeds using the "strong" configuration. We recursively bisect the graph until the parts are too small for KaHIP to handle and assign the order arbitrarily in these small parts. We set the imbalance for KaHIP to 20%. Note that our program is purely tuned for quality. It is certainly possible to trade much speed for a negligible (or even no) decrease in quality. Table 11 reports the times needed to compute

[3] `http://www.ptvgroup.com`

Table 11: Orders. Duration of order computation in seconds. No parallelization was used.

Instance	Greedy	Metis	KaHIP
Karlsruhe	4.1	0.5	1 532
Europe	813.5	131.3	249 082



(a) Karlsruhe (b) Europe

Figure 4: The amount of vertices in the separator (vertical) vs the number of vertices in the subgraph being bisected (horizontal). We only plot the separators for (sub)graphs of at least 1000 vertices. The red hollow circles is KaHIP and the blue filled triangles is Metis.

the orders. Interestingly, Metis outperforms even the greedy strategy. Figure 4 shows the sizes of the computed separators. As expected KaHIP results in better quality. The road graphs seem to have separators following a $\Theta(\sqrt[3]{n})$-law. On Karlsruhe the separator sizes steadily decrease (from the top level to the bottom level). The KaHIP separators on the Europe graph have a different structure on the top level. The separators first increase before they get smaller. This is because of the special structure of the European continent. For example the cut separating Great Britain and Spain from France is far smaller than one would expect for a graph of that size. In the next step KaHIP cuts Great Britain from Spain which results in one of the extremely thin cuts observed in the plot. Interestingly Metis is not able to find these cuts that exploit the continental topology.

4.1.2 CH Construction

Table 12 compares the performance of our specialized Contraction Graph datastructure to the dynamic adjacency structure (see [45]) to compute undirected and unweighted CHs. We do not report numbers for the hash-based approach (see [50]) as it is fully dominated. Our datastructure dramatically improves performance that it also requires less memory). However to be fair, our approach cannot immediately be extended to directed or weighted graphs (i. e., without employing customization).

4.1.3 CH Size

In Table 13 we report the resulting CH sizes for various approaches. Computing a CH on Europe *without witness search* with the greedy order is infeasible even using the Contraction Graph datastructure. This is even true if we only want to count the number of arcs: We aborted calculations

Table 12: Construction of the Contraction Hierarchy. We report the time in seconds required to compute the arcs in $G_\pi^\wedge$ given a KaHIP ND-order π. No witness search is performed. No weights are assigned (yet).

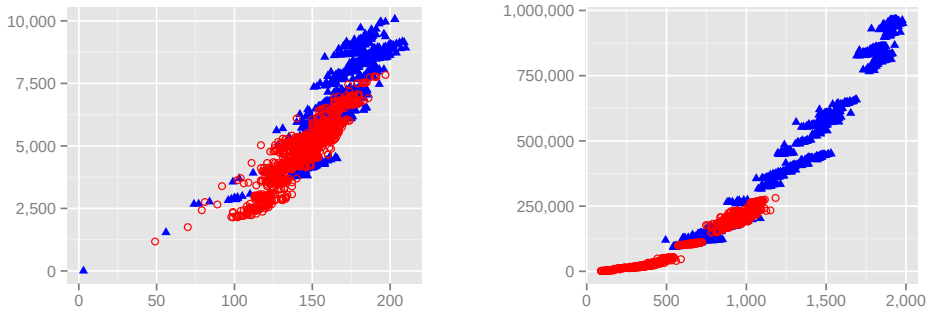
Instance	Dyn. Adj. Array	Contraction Graph
Karlsruhe	0.6	<0.1
Europe	305.8	15.5

Table 13: Size of the Contraction Hierarchies for different instances and orders. We report the number of undirected as well as upward directed arcs of the CH, as well as the number of supporting lower triangles. As an indication for query performance, we report the average search space size in vertices and arcs (both metric-independent undirected and upward weighted), by sampling the search space of 1000 random vertices. Metis and KaHIP orders are metric-independent. Greedy orders are metric-dependent. We report resulting figures after applying different variants of witness search. A heuristic witness search is one that exploits the metric in the preprocessing phase.

						Average search space size			
			# Arcs [$\cdot 10^3$]		# Triangles	undirected		upward	
	Order	Witness search	undir.	upward	[$\cdot 10^3$]	# Vertices	# Arcs	# Vertices	# Arcs
Karlsruhe	Greedy	none	21 926	17 661	37 439 858	5 870	15 786 622	5 246	11 281 564
		heuristic	—	244	—	—	—	108	503
		perfect	—	239	—	—	—	107	498
	Metis	none	478	463	2 590	164	6 579	163	6 411
		perfect	—	340	—	—	—	152	2 903
	KaHIP	none	528	511	2 207	143	4 723	142	4 544
		perfect	—	400	—	—	—	136	2 218
Europe	Greedy	heuristic	—	33 912	—	—	—	709	4 808
	Metis	none	70 070	65 546	1 409 250	1 291	464 956	1 289	453 366
		perfect	—	47 783	—	—	—	1 182	127 588
	KaHIP	none	73 920	69 040	578 248	652	117 406	651	108 121
		perfect	—	55 657	—	—	—	616	44 677

after several days. We can however say with certainty that there are at least 1.3×10^{12} arcs in the CH and the maximum upward vertex degree is at least 1.4×10^6. As the original graph has only 4.2×10^7 arcs, it is safe to assume that using this order it is impossible to achieve a speedup compared to Dijkstra's algorithm on the input graph. However, on the Karlsruhe graph we can actually compute the CH without witness search and perform a perfect witness search. The numbers show that the heuristic witness search employed by [45] is nearly optimal. Furthermore, the numbers clearly show that using greedy orders in a metric-independent setting (i. e., without witness search) results in unpractical CH sizes. However they also show that a greedy order exploiting the weight structure dominates ND-orders (for a more detailed discussion see below). In Figure 5 we plot the number of arcs in the search space vs the number of vertices. The plots show that the KaHIP order significantly outperforms the Metis order on the road graphs. Table 14 examines the elimination tree. Note that the height of the elimination tree corresponds[4] to the number of vertices in the

[4]The numbers in Table 13 and Table 14 deviate a little because the search spaces in the former table are sampled while in the latter we compute precise values.



(a) Karlsruhe (b) Europe

Figure 5: The number of vertices (horizontal) vs the number of arcs (vertical) in the search space of 1000 random vertices. The red hollow circles is KaHIP and the blue filled triangles is Metis.

Table 14: Elimination tree characteristics. Note that unlike in Table 13, these values are exact and not sampled over a random subset of vertices. We also report upper bounds on the treewidth of the (undirected) input graphs.

		# Children		Height		Treewidth
Instance	Order	avg.	max.	avg.	max.	(upper bound)
Karlsruhe	Metis	1	5	163.48	211	92
	KaHIP	1	5	142.19	201	72
Europe	Metis	1	8	1283.45	2017	876
	KaHIP	1	7	654.07	1232	479

(undirected) search space. As the ratio between the maximum and the average height is only about 2 we know that no special vertex exists that has a search space significantly differing from the the numbers shown in Table 14. The elimination tree has a relatively small height compared to the number of vertices in G (in particular, it is not just a path).

The treewidth of a graph is a measure widely used in theoretical computer science. Many NP-hard problems have been shown to be solvable in polynomial time on graphs of bounded treewidth. The notion of treewidth is deeply coupled with the notion of chordal super graphs and vertex separators. See [38] for details. The authors show in their Theorem 6 that the maximum upward degree $d_u(v)$ over all vertices v in $G_\pi^\wedge$ is an upper bound to the treewidth of a graph G. This theorem yields a straightforward algorithm that gives us the upper bounds presented in Table 14.

Interestingly these numbers correlate with our other findings: The difference between the bounds on the road graphs reflect that the KaHIP orders are better than Metis orders. The fact that the treewidth grows with the graph size reflects that the running times are not independent of the graph size. These numbers strongly suggest that road graphs are not part of a graph class of constant treewidth. However, fortunately, the treewidth grows sub-linearly. Our findings from Figure 4 suggest that assuming a $O(\sqrt[3]{n})$ treewidth for road graphs of n vertices might come close to reality. Further investigation into algorithms explicitly exploiting treewidth might be promising. The works of [39, 48] seem like a good start. Also, determining the precise treewidth could prove useful.

In Table 15 we evaluate the witness search performances for different metrics. It turns out that the distance metric is the most difficult one of the tested metrics. That the distance metric is more difficult than the travel time metric is well known. However it surprised us, that uniform and random metrics are easier than the distance metric. We suppose that the random metric contains

Table 15: Detailed analysis of the size of CHs. We evaluate uniform, random and distance weights on the Karlsruhe input graph. Random weights are sampled from $[0, 10000]$. The distance weight is the straight distance along a perfect Earth sphere's surface. All weights respect one-way streets of the input graph.

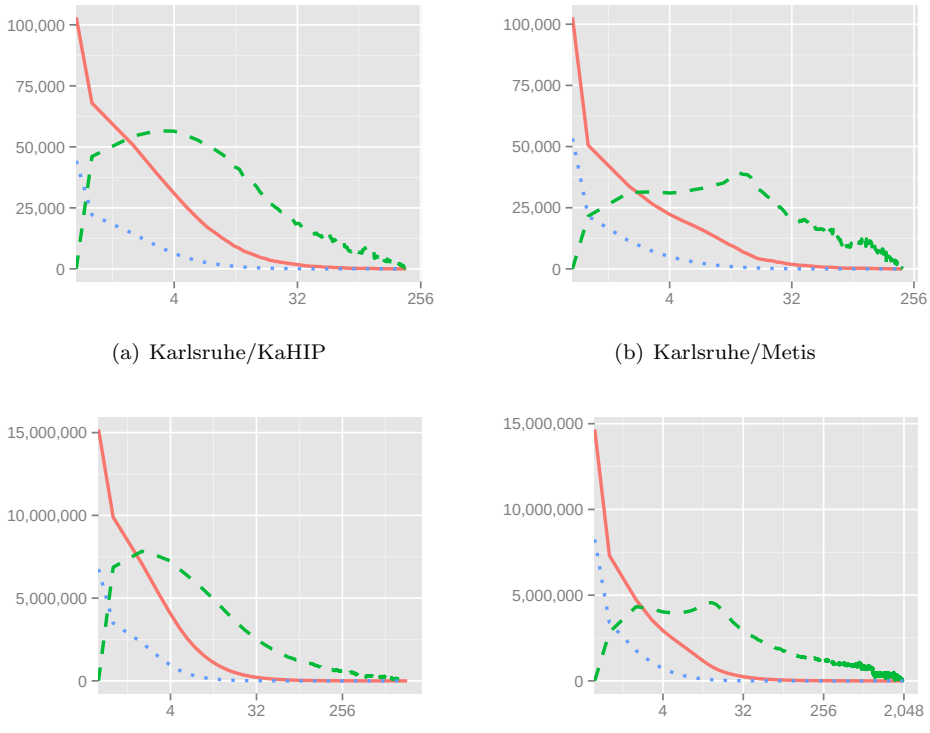
Instance	Metric	Order	Witness search	# Upward arcs	Avg. upward search space	
					# Vertices	# Arcs
Karlsruhe	Distance	Greedy	none	8 000 880	3 276	4 797 224
			heuristic	295 759	283	2 881
			perfect	295 684	281	2 873
		Metis	perfect	382 905	159	3 641
		KaHIP	perfect	441 998	141	2 983
	Uniform	Greedy	none	5 705 168	2 887	3 602 407
			heuristic	272 711	151	808
			perfect	272 711	151	808
		Metis	perfect	363 310	153	2 638
		KaHIP	perfect	426 145	136	2 041
	Random	Greedy	none	6 417 960	3 169	4 257 212
			heuristic	280 024	160	949
			perfect	276 742	160	948
		Metis	perfect	361 964	154	2 800
		KaHIP	perfect	424 999	138	2 093
Europe	Distance	Greedy	heuristic	39 886 688	4 661	133 151
		Metis	perfect	53 505 231	1 257	178 848
		KaHIP	perfect	60 692 639	644	62 014

a few very long arcs that are nearly never used. These could just as well be removed from the graph resulting in a thinner graph with nearly the same shortest path structure. The CH of a thinner graph with a similar shortest path structure naturally has a smaller size. To explain why the uniform metric behaves more similar to the travel time metric than to the distance metric we have to realize that highways do not have many degree 2 vertices in the input graph. (Note that for different data sources this assumption might not hold.) Highways are therefore also preferred by the uniform metric. We expect a more an instance with more degree 2 nodes on highways to behave differently. Interestingly the heuristic witness search is perfect for a uniform metric. We expect this effect to disappear on larger graphs.

Recall that a CH is a DAG, and in DAGs each vertex can be assigned a level. If a vertex can be placed in several levels we put it in the lowest level. Figure 6 illustrates the amount of vertices and arcs in each level of a CH. The many highly ranked extremely thin levels are a result of the top level separator clique: Inside a clique every vertex must be on its own level. A few big separators therefore significantly increase the level count.

4.1.4 Triangle Enumeration

We first evaluate the running time of the adjacency-array-based triangle enumeration algorithm. Figure 7 clearly shows that most time is spent enumerating the triangles of the lower levels. This justifies our suggestion to only precompute the triangles for the lower levels as these are the levels were the optimization is most effective. However, precomputing more levels does not hurt if enough memory is available. We propose to determine the threshold level up to which triangles are



(a) Karlsruhe/KaHIP

(b) Karlsruhe/Metis

(c) Europe/KaHIP

(d) Europe/Metis

Figure 6: The number of vertices per level (blue dotted line), arcs departing in each level (red solid line) and lower triangles in each level (green dashed line). Warning: In contrast to Figure 7 these figures have a logarithmic x-scale.

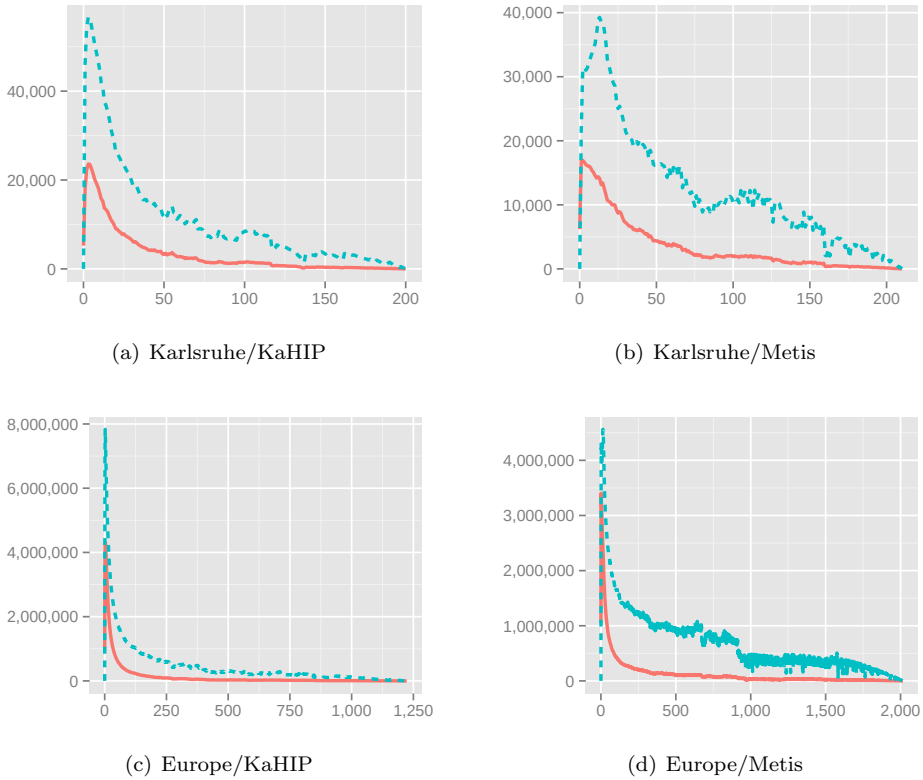
precomputed based on the size of the available unoccupied memory. On modern server machines such as our benchmarking machine there is enough memory to precompute all levels. The memory consumption is summarized in Table 16.

4.1.5 Customization

In Table 17 we report the times needed to compute a maximum metric given an initial one. A first observation is that on the road graphs the KaHIP order leads to a faster customization. Using all optimizations presented we customize Europe in below one second. When amortized[5], we even achieve 415 ms which is only slightly above the (non-amortized) 347 ms reported in [42] for CRP. (Note that their experiments were run on a different machine with a faster clock but 2×6 instead of 2×8 cores and use a turn-aware data structure making an exact comparison difficult.)

We report our partial update results in Table 18. The median, average and maximum running times significantly differ. There are a few arcs that trigger a lot of subsequent changes whereas for most arcs a weight change has nearly no effect. The explanation is that highway arcs and choke point arcs are part of many shortest paths and thus updating such an arc triggers significantly more changes. Interestingly in the worst observed case, using the KaHIP order triggers less changes on TheFrozenSea graph than using the Metis order but an update needs more time. The reason for

[5]We refer to a server scenario of multiple active users that require simultaneous customization, e. g., due to traffic updates.



(a) Karlsruhe/KaHIP

(b) Karlsruhe/Metis

(c) Europe/KaHIP

(d) Europe/Metis

Figure 7: The number of lower triangles per level (blue dashed line) and the time needed to enumerate all of them per level (red solid line). The time unit is 100 nanoseconds. If the time curve thus rises to 1 000 000 on the plot the algorithm needs 0.1 seconds. Warning: In contrast to Figure 6 these figures do not have a logarithmic x-scale.

this is that the KaHIP order results in significantly more triangles and thus the work per arc is higher than what is needed with the Metis order.

For completeness we report the running times of the perfect customizations in Table 19. Note that a perfect customization is not a necessary step of our proposed tool chain. Hence, optimizing this code path had a low priority.

4.1.6 Query Performance

We experimentally evaluated the running times of the queries algorithms. For this we ran 10^6 shortest path distance queries with the source and target vertices picked uniformly at random. (For Europe + Distance we only ran 10^4 queries.) The presented times are averaged running times on a single core without any SSE.

In Table 20 we compare the query running times of weighted CHs with Customizable CHs (CCHs). To construct the weighted CHs we used a (non-perfect) witness search whereas no witness search was used for the metric-independent CHs. We further reordered the vertices in the metric-independent CHs by ND-order. Preliminary experiments showed that this reordering results in better cache behavior and a speed-up of about 2 to 3 because much query time is spent on the topmost clique. We evaluate the basic query, the stall-on-demand optimization, and the elimination-tree based query. Note that the latter only works for metric-independent CHs (as the metric-independent search spaces of weighted CHs get huge).

Table 16: Precomputed triangles. The memory needed is proportional to $2t + m + 1$, where t is the triangle count and m the number of arcs in the CH. We use 4 byte integers. We report t and m for precomputing all levels (*full*) and all levels below a reasonable threshold level (*partial*). We further indicate how much percent of the total unaccelerated enumeration time is spent below the given threshold level. We chose the threshold level such that this factor is about 33 %.

		Karlsruhe		Europe	
		Metis	KaHIP	Metis	KaHIP
full	# Triangles [10^3]	2 590	2 207	1 409 250	578 247
	# CH arcs [10^3]	478	528	70 070	73 920
	Memory [MB]	22	19	11 019	4 694
partial	Threshold level	16	11	42	17
	# Triangles [10^3]	507	512	147 620	92 144
	# CH arcs [10^3]	367	393	58 259	59 282
	Memory [MB]	5	5	1 348	929
	Enum. time [%]	33	32	32	33

Table 17: Customization performance. We report the time needed to compute a maximum customized metric given an initial pair of upward and downward metrics. We show the impact of enabling SSE, precomputing triangles (Pre. trian.), multi-threading (# Thr.), and customizing several metric pairs at once.

				Karlsruhe		Europe	
SSE	Pre. trian.	# Thr.	# Metrics Pairs	Metis time [s]	KaHIP time [s]	Metis time [s]	KaHIP time [s]
no	none	1	1	0.0567	0.0468	21.90	10.88
yes	none	1	1	0.0513	0.0427	19.91	9.55
yes	all	1	1	0.0094	0.0091	7.32	3.22
yes	all	16	1	0.0034	0.0035	1.03	0.74
yes	all	16	2	0.0035	0.0033	1.34	1.05
yes	all	16	4	0.0040	0.0048	2.80	1.66

In comparison to the numbers reported in the original CH paper [45] our running times for weighted CHs tend to be slightly faster. However, our machine is faster which should explain most differences. The only exception is the Europe graph with the distance metric. Here, our measured running time of only 0.540 ms is disproportionately faster. We suppose that the reason is that our order is better as we do not use lazy update and thus have a higher preprocessing time. As already observed by the original authors we confirm that the stall-on-demand heuristic improves running times by a factor 2 to 5 compared to the basic algorithm on weighted CHs. When using ND-order the stalling query is however slower: The search spaces of weighted CHs are spare whereas in the metric-independent case they are dense. This significantly increases the number of arcs that must be tested in the stalling test and explains why stalling is not useful.

For the metric-independent CHs the basic query algorithm (i. e., bidirectional search with stopping criterion) visits large portions of the search space, as can be seen by comparing the search space sizes from Table 13 with the numbers reported in Table 20. For this reason, it pays off to use the elimination tree based query algorithm. It always visits the whole search space but as we see these are only slightly more vertices. However, it does not need a priority queue and therefore spends less time per vertex. Another advantage of the elimination tree based algorithm is that the

Table 18: Partial update performance. We report time required in milliseconds and number of arcs changed for partial metric updates. We report median, average and maximum over 10000 runs. In each run we change the upward and the downward weight of a single random arc in $G_\pi^\wedge$ (the arc is not necessarily in G) to random values in $[0, 10^5]$. The metric is reset to initial state between runs. Timings are sequential without SSE. No triangles were precomputed.

		Arcs removed from queue			Partial update time [ms]		
		med.	avg.	max.	med.	avg.	max.
Karlsruhe	Metis	1	4.1	442	0.001	0.004	0.9
	KaHIP	1	3.7	354	0.001	0.003	1.0
Europe	Metis	1	89.3	16997	0.003	1.0	219.3
	KaHIP	1	38.8	10666	0.003	0.2	87.2

Table 19: Perfect Customization. We report the time required to turn an initial metric into a perfect metric. Runtime is given in seconds, without use of SSE or triangle precomputation.

	Karlsruhe		Europe	
# Thr.	Metis	KaHIP	Metis	KaHIP
1	0.15	0.13	67.01	32.96
16	0.03	0.02	14.41	5.47

code paths do not depend on the metric. This means that query times are completely independent of the metric as can be seen by comparing the running times of the travel time metric to the distance metric. For the basic query algorithms the metric has a slight influence on the performance. A stalling query on the weighted CH with travel time is on Europe about a factor of 5 faster than the elimination tree based algorithm. However for the distance metric this is no longer the case. Here, the metric-independent elimination tree based approach is even faster by about 20% because of the lack of priority queue.

In Table 21, we give a more in-depth experimental analysis of the elimination tree query algorithm. We break the running times up into the time needed to compute the least common ancestor (LCA), the time needed to reset the tentative distances and the time needed to relax all arcs. We further report the total distance query time (which is in essence the sum of the former three) and the time needed to unpack the full path. Our experiments show that the arc-relaxation phase clearly dominates the running times. It is therefore not useful to further optimize the LCA computation or to accelerate tentative distance resetting using, e. g., timestamps. The path unpacking does not use precomputed lower triangles. Using them would result in a further speedup with a similar trade-off as already discussed for customization.

Fair query time comparisons with CRP [41] are difficult because they nearly only report turn-aware query running times, whereas the graphs we tested do not use turns. As far as we are aware, non-turn-aware query performance was only published in [40], but here queries were parallelized using two cores: The forward and backward searches are run in parallel. The authors report queries in 0.72 ms for travel time and 0.79 ms for distance metric on Europe. This is slower than our sequential query times of 0.41 ms and 0.43 ms, respectively. (Note that these experiments were run on a slightly different machine than ours.)

We have shown in Table 17 that ND-orders can be combined with perfect witness search to get CHs of smaller search spaces. This could be exploited to achieve (even) faster query times as the number of arcs decrease by a factor $\approx$2 on road and $\approx$4 on game maps. As the elimination-tree query spends nearly all of its time visiting arcs we expect its running time to go down by about the

Table 20: Contraction Hierarchies query performance. We report the query time *in microseconds* as well as the search space visited (we use visited to differentiate from the maximum reachable search space given in Table 13). For query algorithms that use stalling, we additionally report the number of vertices stalled after queue removal, as well as the number of arcs touched during the stalling test. Note that the search space figures do not contain such stalled vertices. All reported vertex and arc counts only refer to the forward search. We evaluate several algorithmic variants. Each variant is composed of an input graph, a contraction order, and whether a witness search is used. "+w" means that a (non-perfect) witness search is used, whereas "-w" means that no witness search is used. "greedy+w" corresponds to the original CHs. The metrics used for the non-greedy CHs are directed and maximum.

				Visited search space		Stalling		Time
Instance	Metric	Variant	Algorithm	# Vertices	# Arcs	# Vertices	# Arcs	[µs]
Karlsruhe	Travel-Time	Greedy+w	Basic	81	370	—	—	17
			Stalling	43	182	167	227	16
		Metis-w	Basic	138	5 594	—	—	62
			Stalling	104	4 027	32	4 278	67
			Tree	164	6 579	—	—	33
		KaHIP-w	Basic	120	4 024	—	—	48
			Stalling	93	3 051	26	3 244	55
			Tree	143	4 723	—	—	25
	Distance	Greedy+w	Basic	208	1978	—	—	57
			Stalling	69.5	559	46	759	35
		Metis-w	Basic	142	5 725	—	—	65
			Stalling	115	4 594	26	4 804	75
			Tree	164	6 579	—	—	33
		KaHIP-w	Basic	123	4 117	—	—	50
			Stalling	106	3 480	17	3 564	59
			Tree	143	4 723	—	—	26
Europe	Travel-Time	Greedy+w	Basic	546	3 623	—	—	283
			Stalling	113	668	75	911	107
		Metis-w	Basic	1 126	405 367	—	—	2 838
			Stalling	719	241 820	398	268 499	2 602
			Tree	1 291	464 956	—	—	1 496
		KaHIP-w	Basic	581	107 297	—	—	810
			Stalling	418	75 694	152	77 871	857
			Tree	652	117 406	—	—	413
	Distance	Greedy+w	Basic	3 653	104 548	—	—	2 662
			Stalling	286	7 124	426	11 500	540
		Metis-w	Basic	1 128	410 985	—	—	3 087
			Stalling	831	291 545	293	308 632	3 128
			Tree	1 291	464 956	—	—	1 520
		KaHIP-w	Basic	584	108 039	—	—	867
			Stalling	468	85 422	113	87 315	1 000
			Tree	652	117 406	—	—	426

Table 21: Detailed elimination tree performance. We report running time *in microseconds* for the elimination-tree-based query algorithms. We report the time needed to compute the LCA, the time needed to reset the tentative distances, the time needed to relax the arcs, the total time of a distance query, and the time needed for full path unpacking as well as the average number of vertices on such a path (which is metric-dependent).

			Distance query				Path	
			LCA [µs]	Reset [µs]	Arc relax [µs]	Total [µs]	Unpack [µs]	Length [vert.]
Karlsruhe	Travel-Time	Metis	0.6	0.8	31.3	33.0	20.5	189.6
		KaHIP	0.6	1.4	23.1	25.2	18.6	
	Distance	Metis	0.6	0.8	31.5	33.2	27.4	249.4
		KaHIP	0.6	1.4	23.5	25.7	24.7	
Europe	Travel-Time	Metis	4.6	19.0	1471.2	1496.3	323.9	1390.6
		KaHIP	3.4	9.9	399.4	413.3	252.7	
	Distance	Metis	4.7	19.0	1494.5	1519.9	608.8	3111.0
		KaHIP	3.6	10.0	411.6	425.8	524.1	

Table 22: Customization and query performance on the Europe road network instance using different metrics. Node order was obtained using KaHIP. Customization uses SSE, precomputed triangles, and 16 threads. Queries use the elimination tree-based algorithm and are single-threaded. We report average performance figures. Query source and destination are sampled uniformly at random; this implies expected long-distance paths. Performance of local queries, e. g., within a city, will be much faster.

			Query		
		Customization	Visited search space		Time
Network	Metric	Time [s]	# Vertices	# Arcs	[ms]
Europe	Travel-Time	0.744	652	117 406	0.413
Europe	Distance	0.736	652	117 406	0.426
Europe	Emissions	0.742	652	117 406	0.422

same factor. However, a perfect customization is slower by a factor of ≈3 (c. f. Table 19). In total, combining ND-orders and perfect witness search yields another Pareto-optimal trade-off between customization time and query time.

4.1.7 Optimizing Eco-friendliness

Our experiments so far show that the performance of our route planning approach is very robust with respect to the metric used. In the following, we will confirm this observation by applying our techniques on the computation of emission-optimal routes. We augmented our Europe road network with emission data by considering, as in Section 3.3.4, a 5-door Diesel Opel Astra with an average fuel-consumption of 4.2 l/100 km. We considered well-to-wheels greenhouse gas emissions calculated as CO_2 equivalents, for which we obtained a factor of 3.24 kg CO_2e/l from EN 16258:2012.

For the resulting emission metric, we reran our customization and query experiments. More specifically, we based customization on the nested dissection order (c. f. Deliverable D2.2.1, Section 4.4) computed by KaHIP and employed the fastest customization variant available, enabling precomputed triangles, SSE, and multi-core parallelization with 16 threads (c. f. D2.2.1, Sec-

tion 4.7). For the query algorithm we used our elimination tree query variant (c. f. D2.2.1, Section 4.8.3).

Table 22 summarizes the results. For comparison, we repeat figures for travel-time and distance metric from experiments above. Since both our customization algorithm and the elimination-tree-based query algorithm operate on the input graph in an order independent of the metric used, the results are not that surprising. One can see that timings for customization and queries for optimizing eco-friendly routes are almost identical to those of travel-time optimal or distance optimal routes, except for small measurement uncertainties. Again, customization is done in below one second, an average random query in below one millisecond. Indeed, the operations count for the elimination-tree-based query is identical. Clearly, we can gradually increase the complexity of the emissions model, e. g., by also taking driving speed and slopes into account, without degrading query performance results as long as the metric is *scalar* per edge after customization.

The proposed workflow in a production system would then be: When generating a new map release (e. g., every three months), run the metric-independent preprocessing of the node order. When a user logs into the service or changes her preferences, run the metric-dependent customization in below one second. To account for the current traffic situation, either re-run full customization in below one second or apply our partial update algorithm for only the changed road segments for an average runtime of below a millisecond (c. f. Table 18). After customization, even long-distance queries can be answered in below a millisecond on average.

Excursion on electric vehicles. These observations do not hold for functional metrics, e. g., when considering time-dependency (where edge travel-time is a function of the time-of-day) or energy-optimal routes for electric vehicles (where edge consumption is a function of the current state of charge). However, in eCOMPASS-TR-028, we have successfully applied comparable algorithmic techniques (i. e., a variant of CRP [41]) to the scenario of electric vehicles. Table 23 reports key results, showing that comparable query speeds can be achieved for energy-optimal electric vehicle route planning. Please see the TR for more details.

We also compared energy-optimal routes to those that optimize travel time and distance metric, respectively. We used the same 10 000 queries as in Table 23, but only evaluated those where the target was reachable (85 % for Europe PG-16, 86 % for Europe EV-85, and 100 % for Japan DH-∞). Table 24 shows results for all instances. For each metric, we report the percentage of queries that become unreachable when considering travel time and distance; the extra energy spent when using the quickest and shortest route (instead of the energy-optimal one); and the extra time and distance required when using the energy-optimal route. Note that travel times were not available for Japan DH-∞, therefore, we only evaluated the distance metric on this instance.

As the driving speed has a huge impact on the energy consumption, minimizing the travel time greatly reduces range. Consequently, more than half of the targets reachable on an energy-optimal route become unreachable when taking the quickest route. Even if the target is reachable in both cases, optimizing one criterion greatly increases the other. This effect becomes less significant when comparing energy consumption to distance. This indicates that there is a correlation between energy consumption and driving distance. However, since there are many other factors—such as road type and slope—that influence energy consumption, minimizing travel distance still fails to reach the target in more than 20 % of the queries.

5 Alternative Route Planning

5.1 Introduction

Route planning services – offered by web-based, hand-held, or in-car navigation systems – are heavily used by more and more people. Typically, such systems (as well as the vast majority of route planning algorithms) offer a best route from a source (origin) s to a target (destination) t,

Table 23: Evaluating our algorithms on both vehicle instances: A Peugeot iOn with a 16 kWh battery (Europe PG-16) and an artificial vehicle with a 85 kWh battery (Europe EV-85). We also report figures on an instance from [49]: It uses the geographical distance and height difference of the arcs to model consumption and assumes unlimited capacity. This table is reproduced from ECOMPASS-TR-028, where a more detailed description is available.

	Europe PG-16				**Europe EV-85**				**Japan DH-∞**			
	Customizing		Queries		Customizing		Queries		Customizing		Queries	
Algorithm	Space [B/n]	Time [s]	Vertex Scans	Time [ms]	Space [B/n]	Time [s]	Vertex Scans	Time [ms]	Space [B/n]	Time [s]	Vertex Scans	Time [ms]
Uni-MLD-PH	13.6	4.32	941	0.5	14.5	5.12	2 410	1.9	7.7	2.06	2 205	1.0
BPE-MLD-PH	13.6	4.32	929	0.3	14.5	5.12	2 266	1.4	7.7	2.06	2 198	0.8
BDB-MLD-PH	13.6	4.32	1 203	0.3	14.5	5.12	2 917	1.1	7.7	2.06	2 711	0.7

Table 24: Comparison of energy-optimal routes to routes that minimize either travel time or distance. This table is reproduced from ECOMPASS-TR-028, where a more detailed description is available.

	Travel Time			**Distance**		
Instance	Unr.	Extra Energy	Extra Time	Unr.	Extra Energy	Extra Dist.
Europe PG-16	54 %	41 %	46 %	21 %	11 %	5 %
Europe EV-85	60 %	62 %	63 %	25 %	15 %	4 %
Japan DH-∞	—	—	—	0 %	25 %	11 %

under a single criterion (usually distance or time). Quite often, however, computing only one such s-t route may not be sufficient, since humans would like to have choices and every human has also his/her own preferences. These preferences may well vary and depend on specialized knowledge or subjective criteria (like or dislike certain part of a road), which are not always practical or easy to obtain and/or estimate on a daily basis. Therefore, a route planning system offering a set of good/reasonable alternatives can hope that (at least) one of them can satisfy the user, and vice versa, the user can have them as back-up choices for altering his/her route in case of emergent traffic conditions. This can be particularly useful in several cases. For example, when the user has to choose the next optimal alternative path, because in the current one, adverse incidents are occurred, like traffic jams, accidents or permanent unavailability due to construction work.

The aggregation of alternative paths between a source s and a target t can be captured by the concept of the *Alternative Graph* (AG), a notion first introduced in [63]. Storing paths in an AG makes sense, because in general alternative paths may share common nodes (including s and t) and edges. Furthermore, their subpaths may be combined to form new alternative paths.

In general, there may be several alternative paths from s to t. Hence, there is a need for filtering and rating all alternatives, based on certain quality criteria. The study in [63] quantified the quality characteristics of an alternative graph (AG), captured by three criteria. These concern the non-overlappingness (*totalDistnace*) and the stretch (*averageDistnace*) of the routes, as well as the number of *decisionEdges* (sum of node out-degrees) in AG. For more details, see Deliverable D2.2. As it is shown in [63], all of them together are important in order to produce a high-quality AG. However, optimizing a simple objective function combining just any two of them is already an NP-hard problem [63]. Hence, one has to concentrate on heuristics. Four heuristic approaches were investigated in [63] with those based on Plateau [61], Penalty [64], and a combination of them to be the best.

In this deliverable, for the sake of completeness, we present our final improved methods for computing a set of alternative source-to-destination routes in road networks in the form of an alternative graph, which appear to be more suitable for practical navigation systems [62, 67]. These methods appeared in [69]. The resulting alternative graphs are characterized by minimum path overlap, small stretch factor, as well as low size and complexity. Our approach improves upon a previous one by introducing a new pruning stage preceding any other heuristic method and by introducing a new filtering and fine-tuning of two existing methods.

We extend the approach in [63] for building AGs in two directions. First, we introduce a pruning stage that precedes the execution (and it is independent) of any heuristic method, thus reducing the search space and hence detecting the nodes on shortest routes much faster. Second, we provide several improvements on both the Plateau and Penalty methods. In particular, we use a different approach for filtering plateaus in order to identify the best plateaus that will eventually produce the most qualitative alternative routes, in terms of minimum overlapping and stretch. We also introduce a practical and well-performed combination of the Plateau and Penalty methods with tighter lower-bounding based heuristics. This has the additional advantage that the lower bounds remain valid for use even when the edge costs are increased (without requiring new preprocessing), and hence are useful in dynamic environments where the travel time may be increased, for instance, due to traffic jams.

Finally, we conducted an experimental study for verifying our methods on several road networks of Western Europe. Our experiments showed that our methods can produce AGs of high quality pretty fast.

The rest of this section is organized as follows. In subsection 5.2, we provide the main background information, from Deliverable 2.2. In subsection 5.3, we present our proposed improvements for producing AGs of better quality. In subsection 5.4, we report a thorough experimental evaluation of our improved methods. In subsection 5.5, we demonstrate some of the visualized results we got with our alternative route planning implementation.

5.2 Preliminaries

A *road network* can be modeled as a *directed graph* $G = (V, E)$, where each node $v \in V$ represents intersection points along roads, and each edge $e \in E$ represents road segments between pairs of nodes. Let $|V| = n$ and $|E| = m$ and $d(u, v) \equiv d_G(u, v)$ be the shortest distance from u to v in graph G.

We consider the problem of tracing alternative paths from a source node s to a target node t in G, with edge weight or cost function $w : E \to \mathbb{R}^+$. The essential goal is to obtain sufficiently different paths with optimal or near optimal cost. We proceed with the definitions of an alternative graph and its quality indicators.

Alternative Graph. Formally, an *AG* $H = (V', E')$ [63] is a graph, with $V' \subseteq V$, and such that for all $e = (u, v) \in E'$, there is a P_{uv} path in G and a P_{st} path in H, so that $e \in P_{st}$ and $w(e) = w(P_{uv})$, where $w(P_{uv})$ denotes the weight or cost of path P_{uv}. Let $d_H(u, v)$ be the shortest distance from u to v in graph H.

Quality indicators. For filtering and rating the alternatives in an *AG*, we use the following indicators, as in [63]:

$$totalDistance = \sum_{e=(u,v)\in E'} \frac{w(e)}{d_H(s,u) + w(e) + d_H(v,t)} \qquad (overlapping)$$

$$averageDistance = \frac{\sum_{e\in E'} w(e)}{d_G(s,t) \cdot totalDistance} \qquad (stretch)$$

$$decisionEdges = \sum_{v\in V'\setminus\{t\}} (outdegree(v) - 1) \qquad (size\ of\ AG)$$

In the above definitions, the *totalDistance* measures the extend to which the paths in AG are non-overlapping. Its maximum value is *decisionEdges*+1. This is equal to the number of all s-t paths in AG, when these are disjoint, i.e. not sharing common edges.

The *averageDistance* measures the average cost of the alternatives compared with the shortest one (i.e. the stretch). Its minimum value is 1. This occurs, when every s-t path in AG has the minimum cost.

The *decisionEdges* measures the size complexity of AG. In particular, the number of the alternative paths in AG, depend on the "decision branches" are in AG. For this reason, as high the number o *decisionEdges*, the more confusion is created to a typical user, when he tries to decide his route. Therefore, it should be bounded.

Consequently, to compute a qualitative AG, one aims at high *totalDistance* and low *averageDistance*. Examples of the use of the above quality indicators can be found in Deliverable D2.2.

5.3 Our Improvements

In Deliverable 2.2, we reviewed the previous approaches for computing alternative graphs, and briefly highlighted our improved methods. In this deliverable, we present in detail these improved methods by extending the Plateau and Penalty approaches. Our improvements are twofold :

A) We introduce a pruning stage that precedes the Plateau and Penalty methods in order to a-priori reduce their search space without sacrificing the quality of the resulted alternative graphs.

B) We use a different approach for filtering plateaus in order to obtain the ones that generate the best alternative paths. In addition, we fine tune the penalty method, by carefully choosing the penalizing factors on the so far computed P_{st} paths, in order to trace the next best alternatives.

5.3.1 Pruning

We present two bidirectional Dijkstra-based *pruners*. The purpose of both of them, is to identify the nodes that are in P_{st} shortest paths. We refer to such nodes, as the useful search space, and the rest ones, as the useless search space. Our goal, through the use of search pruners, is to ensure: (a) a more quality-oriented build of the AG and (b) a reduced dependency of the time computation complexity from graph size. The latter is necessary, in order to acquire fast response on queries. We note that the benefits are notably for the Penalty method. This is because, the Penalty method needs to run iteratively several s-t shortest path queries. Thus, having put aside the useless nodes and focussing only on the useful ones, we can get faster processing. We also note that, over the P_{st} paths with the minimum cost, it may be desired as well to let in AG paths with near optimal cost, say $\tau \cdot d_s(t)$, which will be the maximum acceptable cost $w(P_{st})$. Indicatively, $1 \leqslant \tau \leqslant 1.4$. Obviously, nodes far away from both s and t, with $d_s(v) + d_t(v) > \tau \cdot d_s(t)$, belong to P_{st} paths with prohibitively high cost. In the following we provide the detailed description of both pruners, which are illustrated in Figures 8 and 9.

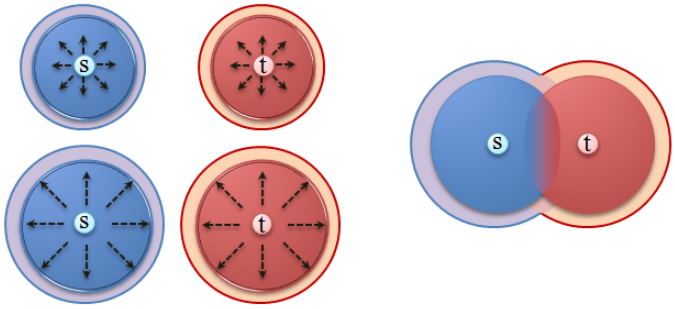


Figure 8: The forward and backward searches meet each other. In this phase the minimum distance $d_s(t)$ is traced.

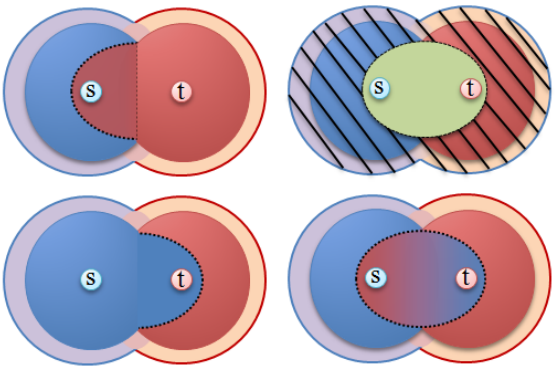


Figure 9: The forward and backward settles only the nodes in the shortest paths, taking account of the overall $d_s(v) + d_t(v)$.

Uninformed Bidirectional Pruner. In this pruner, there is no preprocessing stage. Instead, the used heuristics are obtained from the minimum distances of the nodes enqueued in Q_f and Q_b, i.e. $Q_f.minKey() = \min_{u \in Q_f}\{d_s(u)\}$ and $Q_b.minKey() = \min_{v \in Q_b}\{d_t(v)\}$.

We extend the regular bidirectional Dijkstra, by adding one extra phase. First, for computing the minimum distance $d_s(t)$, we let the expansion of forward and backward search until $Q_f.minKey() + Q_b.minKey() \geq d_s(t)$. At this step, the current forward T_f and backward T_b shortest path trees produced by the bidirectional algorithm will have crossed each other and so the minimum distance $d_s(t)$ will be determined. Second, at the new extra phase, we continue the expansion of T_f and T_b in order to include the remaining useful nodes, such that $d_s(v) + d_t(v) \leq \tau \cdot d_s(t)$, but with a different mode. This time, we do not allow the two searches to continue their exploration at nodes v that have $d_s(v) + h_t(v)$ or $h_s(v) + d_t(v)$ greater than $\tau \cdot d_s(t)$. We use the fact that Q_f and Q_b can provide lower-bound estimates for $h_s(v)$ and $h_t(v)$. Specifically, a node that is not settled or explored from backward search has as a lower bound to its distance to t, $h_t(v) = Q_b.minKey()$. This is because the backward search settles the nodes in increasing order of their distance to t, and if u has not been settled then it must have $d_t(u) \geq Q_b.minKey()$. Similarly, a node that is not settled or explored from forward search has a lower bound $h_s(v) = Q_f.minKey()$. Furthermore, when a search settles a node that is also settled from the other search we can calculate exactly the sum $d_s(u) + d_t(u)$. In this case, the higher the expansion of forward and backward search is, the more tight the lower bounds become. The pruning is ended, when Q_f and Q_b are empty.

Before the termination, we exclude the remaining useless nodes that both searches settled during the pruning, that is all nodes v with $d_s(v) + d_t(v) > \tau \cdot d_s(t)$.

Informed ALT bidirectional pruner. In the second pruner, our steps are similar, except that we use tighter lower bounds. We acquire them in a one-time preprocessing stage, using the *ALT* approach. In this case, the lower bounds that are yielded can guide faster and more accurately the pruning of the search space. We compute the shortest distances between the nodes in G and a small set of landmarks. For tracing the minimum distance $d_s(t)$, we use BLA as base algorithm, which achieves the lowest waste exploration, as experimental results showed in [66, 68]. During the pruning, we skip the nodes that have $d_s(v) + h_t(v)$ or $h_s(v) + d_t(v)$ greater than $\tau \cdot d_s(t)$.

The use of lower-bounding heuristics can be advantageous. In general, a heuristic stops being valid when a change in the weight of the edges occurs. But note that in the penalty method, we consider only increases on the edge weights and therefore this does not affect the lower bounds on the shortest distances. Therefore, the combination of the *ALT* speedup [68, 66] with Penalty is suitable. However, depending on the number and the magnitude of the increases the lower bounds can become less tight for the new shortest distances, leading to a reduced performance on computing the shortest paths.

5.3.2 Filtering and Fine-tuning

Over the standard processing operations of Penalty and Plateau, we introduce new ones for obtaining better results. In particular:

Plateau. We use a different approach on filtering plateaus. Specifically, over the cost of a plateau path we take into account also its non-overlapping with others. In this case, the difficulty is that the candidate paths may share common edges or subpaths, so the *totalDistance* is not fixed. Since at each step an insertion of the current best alternative path in AG may lead to a reduced *totalDistance* for the rest candidate alternatives, primarily we focus only on their unoccupied parts, i.e., those that are not in AG. We rank a x-y plateau $\overline{P}$ with $rank = totalDistance - averageDistance$, where $totalDistance = \frac{w(\overline{P})}{d_s(x)+w(\overline{P})+d_t(y)}$ is its definite non-overlapping degree, and $averageDistance = \frac{w(\overline{P})+d_s(t)}{(1+totalDistance)\cdot d_s(t)}$ is its stretch over the shortest s-t path in G. During the collection of plateaus, we insert the highest ranked of them via its node-connectors $v \in \overline{P}$ in T_f and T_b to a min heap with fixed size equal to *decisionEdges* plus an offset. The offset increases the number of the candidate plateaus, when there are available, and it is required only as a way out, in the case, where several P_{st} paths via the occupied plateaus in AG lead to low *totalDistance* for the rest P_{st} paths via the unoccupied plateaus.

Penalty. When we "penalize" the last computed P_{st} path, we adjust the increases on the weights of its outgoing and incoming edges, as follows:

$$w_{new}(e) = w(e) + (0.1 + r \cdot d_s(u)/d_s(t)) \cdot w_{old}(e), \qquad \forall e = (u, v) \in E : \ u \in P_{st}, \ v \notin P_{st}$$
$$w_{new}(e) = w(e) + (0.1 + r \cdot d_t(v)/d_t(s)) \cdot w_{old}(e), \qquad \forall e = (u, v) \in E : \ u \notin P_{st}, \ v \in P_{st}$$

The first adjustment puts heavier weights on those outgoing edges that are closer to the target t. The second adjustment puts heavier weights on those incoming edges that are closer to the source s. The purpose of both is to reduce the possibility of recomputing alternative paths that tend to rejoin directly with the previous one traced.

An additional care is given also for the nodes u in P_{st}, having $outdegree(u) > 1$. Note that their outgoing edges can form different branches. Since the edge-branches in G constitute generators for alternative paths, they are important. These edges are being inserted to AG with a greater magnitude of weight increase than the rest of the edges.

The insertion of the discovered alternative paths in G and the maintenance of the overall quality of AG should be controlled online. Therefore, we establish an online interaction with the AG's

quality indicators, described in subsection 5.2, for both Plateau and Penalty. This is also necessary because at each step an insertion of the current best alternative may lead to a reduced value of *totalDistance* for the next candidate alternative paths that share common edges with the already computed AG.

In order to get the best alternatives, we seek to maximize the $targetfunction = totalDistance - \alpha \cdot averageDistance$, where α is a balance factor that adjusts the stretch magnitude rather than the overlapping magnitude. Maximization of the target function leads to select the best set of low overlapping and shortest alternative paths.

Since the penalty method can work on any pre-computed AG, it can be combined with Plateau. In this way, we collect the best alternatives from Penalty and Plateau, so that the resulting set of alternatives maximizes the target function. In this matter, we can extend the number of decision edges and after the gathering of all alternatives, we end by performing thinout in AG. Moreover, in order to guide the Penalty method to the remaining alternatives, we set a penalty on the paths stored by Plateau in AG, by increasing their weights. We also use the same pruning stage to accommodate both of them.

5.4 Experimental Results

The experiments were conducted on an Intel(R) Xeon(R) Processor X3430 @ 2.40GHz, with a cache size of 8Mb and 32Gb of RAM. Our implementations were written in C++ and compiled by GCC version 4.6.3 with optimization level 3.

The data sets of the road networks in our experiments were acquired from OSM [59] and TomTom [60]. The weight function is the travel time along the edges. In the case of OSM, for each edge, we calculated the travel time based on the length and category of the roads (residential street, tertiary, secondary, primary road, trunk, motorway, etc). The data set of the Greater Berlin area was kindly provided by TomTom in the frame of the eCOMPASS project [62]. The size of the data sets are reported in Table 25.

In the rest of this section we first report on the performance of our algorithms and then on their eco-footprint.

map		n	m
B	Berlin	117,839	310,152
LU	Luxembourg	51,576	119,711
BE	Belgium	576,465	1,376,142
IT	Italy	2,425,667	5,551,700
GB	GreatBritain	3,233,096	7,151,300
FR	France	4,773,488	11,269,569
GE	Germany	7,782,773	18,983,043
WE	WesternEurope	26,498,732	62,348,328

Table 25: The size of road networks, where n denotes the number of nodes and m denotes the number of edges.

5.4.1 Performance

For our implementations, we used the packed-memory graph (PMG) structure [68]. This is a highly optimized graph structure, part of a larger algorithmic framework, specifically suited for very large scale networks. It provides dynamic memory management of the graph and thus the ability to control the storing scheme of nodes and edges in memory for optimization purposes. It supports almost optimal scanning of consecutive nodes and edges and can incorporate dynamic changes in the graph layout in a matter of μs. The ordering of the nodes and edges in memory is in such a

way that increases the locality of references, causing as few memory misses as possible and thus a reduced running time for the used algorithms.

We tested our implementations in the road network of the Greater Berlin area, the Western Europe (Austria, Belgium, Denmark, France, Germany, Italy, Luxembourg, Netherlands, Norway, Portugal, Spain, Sweden, Switzerland, and Great Britain), as well as in the network of each individual West European country. In the experiments, we considered 100 queries, where the source s and the destination t were selected uniformly at random among all nodes. For the case of the entire Western European road network, the only limitation is that the s-t queries are selected, such that their geographical distance is at most 300 kilometers. This was due to the fact that although modern car navigation systems may store the entire maps, they are mostly used for distances up to a few hundred kilometers.

For far apart source and destination, the search space of the alternative P_{st} paths gets too large. In such cases, it is more likely that many non-overlapping long (in number of edges) paths exist between s and t. Therefore, this has a major effect on the computation cost of the overall alternative route planning. In general, the number of non-overlapping shortest paths depends on the density of the road networks as well on the edge weights.

There is a trade-off between the quality of AG and the computation cost. Thus, we can sacrifice a bit of the overall quality to reduce the running time. Consequently, in order to deal with the high computation cost of the alternative route planning for far apart sources and destinations we can decrease the parameter τ (max stretch). A dynamic and online adjustment of τ based on the geographical distance between source and target can be used too. For instance, at distance larger than 200km, we can set a smaller value to τ, e.g. close to 1, to reduce the stretch and thereby the number of the alternatives. We adopted this arrangement on large networks (Germany, Western Europe). For all others, we set $\tau = 1.2$, which means that any traced path has cost at most 20% larger than the minimum one. To all road networks, we also set $averageDistance \leq 1.1$ to ensure that, in the filtering stage, the average cost of the collected paths is at most 10% larger than the minimum one.

In order to fulfill the ordinary human requirements and deliver an easily representable AG, we have bounded the *decisionEdges* to 10. In this way, the resulted AG has small size, $|V'| \ll |V|$ and $|E'| \ll |E|$, thus making it easy to store or process. Our experiments showed that the size of an AG is at most 3 to 4 times the size of a shortest s-t path, which we consider as a rather acceptable solution.

Our base *target function* [6] in Plateau and Penalty is $totalDistance - averageDistance + 1$. Regarding the pruning stage of Plateau and Penalty, we have used the ALT-based informed bidirectional pruner with at most 24 landmarks for Western Europe.

In Tables 26, 27, and 28, we report the results of our experiments on the various quality indicators: targetFunction (*TargFun*), totalDistance (*TotDist*), averageDistance (*AvgDist*) and decisionEdges (*DecEdges*). The values in parentheses in the header columns provide only the theoretically maximum or minimum values per quality indicator, which may be far away from the optimal values (that are based on the road network and the s-t queries).

In Tables 26, 27, and 28, we report the average value per indicator. The overall execution time for computing the entire AG is given in milliseconds. As we see, we can achieve a high-quality AG in less than a second even for continental size networks. The produced alternative paths in AG are directly-accessible for use (e.g., they are not stored in any compressed form).

Due to the limitation on the number of the decision edges in AG and the low upper bound in stretch, we have chosen in the Penalty method small penalty factors, $p = 0.1$ and $r = 0.1$. In addition, this serves in getting better low-stretch results, see Table 27. In contrast, the *averageDistance* in Plateau gets slightly closer to the 1.1 upper bound.

In our experiments, the Penalty method clearly outperforms Plateau on finding results of higher

[6] We have been very recently informed [65] that this is the same target function as the one used in [63] and not the erroneously stated $totalDistance - averageDistance$ in that paper.

quality. However it has higher computation cost. This is reasonable because it needs to perform around to 10 shortest *s-t* path queries. The combination of Penalty and Plateau is used to extract the best results of both of the methods. Therefore in this way the resulted *AG* has better quality than the one provided by any individual method. In Tables 26, 27, and 28, we also report on the *TargFun* quality indicator of the study in [63]. The experiments in that study were run only on the LU and WE networks, and on data provided by PTV, which concerned smaller (in size) networks and which may be somehow different from those we use here [59]. Nevertheless, we put the *TargFun* values in [63] as a kind of reference for comparison.

map	TargFun		TotDist	AvgDist	DecEdges	Time
	(max:11)	in [63]	(max:11)	(min:1)	(max:10)	(ms)
B	3.82	-	3.91	1.09	9.95	45.61
LU	4.44	3.05	4.49	1.05	9.73	37.05
BE	4.83	-	4.87	1.04	10.00	85.08
IT	4.10	-	4.14	1.04	9.92	114.29
GB	4.36	-	4.40	1.04	9.93	180.12
FR	4.22	-	4.26	1.04	9.97	159.93
GE	4.88	-	4.92	1.04	10.00	286.40
WE	4.35	3.08	4.37	1.02	9.88	717.57

Table 26: The average quality of the resulted *AG* via Plateau method.

map	TargFun		TotDist	AvgDist	DecEdges	Time
	(max:11)	in [63]	(max:11)	(min:1)	(max:10)	(ms)
B	4.16	-	4.23	1.07	9.92	49.34
LU	5.14	2.91	5.19	1.05	9.23	41.56
BE	5.29	-	5.33	1.04	9.54	159.71
IT	4.11	-	4.14	1.03	9.47	105.84
GB	4.38	-	4.41	1.03	9.87	210.94
FR	4.11	-	4.16	1.05	9.32	192.44
GE	5.42	-	5.46	1.04	9.91	388.97
WE	5.21	3.34	5.24	1.03	9.67	776.97

Table 27: The average quality of the resulted *AG* via Penalty method.

map	TargFun		TotDist	AvgDist	DecEdges	Time
	(max:11)	in [63]	(max:11)	(min:1)	(max:10)	(ms)
B	4.55	-	4.61	1.06	9.97	54.12
LU	5.25	3.29	5.30	1.05	9.81	43.69
BE	5.36	-	5.41	1.05	9.89	163.75
IT	4.37	-	4.41	1.04	9.79	178.08
GB	4.67	-	4.71	1.04	9.86	284.38
FR	4.56	-	4.60	1.04	9.86	217.30
GE	5.50	-	5.54	1.04	9.89	446.38
WE	5.49	3.70	5.52	1.03	9.94	987.42

Table 28: The average quality of the resulted *AG* via the combined Penalty and Plateau method.

We would like to note that if we allow a larger value of τ (up to 1.2) for large networks (e.g., WE)

and for s-t distances larger than 300km, then we can achieve higher quality indicators (intuitively, this happens due to the many more alternatives in such a case). Indicative values of quality indicators for WE are reported in Table 29, 30.

map WE	TargFun	TotDist	AvgDist	DecEdges	Time(ms)
Plateau	4.57	4.59	1.02	10.00	1564.28
Penalty	4.36	4.38	1.02	9.95	2588.31
Plateau & Penalty	6.29	6.31	1.02	9.97	2692.56

Table 29: Random alternative route queries in the road network of Western Europe, with geographical distance up to 400km.

map WE	TargFun	TotDist	AvgDist	DecEdges	Time(ms)
Plateau	4.71	4.73	1.02	10.00	2171.13
Penalty	4.78	4.80	1.02	9.97	3536.76
Plateau & Penalty	6.46	6.48	1.02	9.98	3806.92

Table 30: Alternative route queries in the road network of Western Europe, with geographical distance up to 500km.

5.4.2 Eco-Footprint Evaluation

The eco-footprint of the routes provided by the alternative graph are computed considering an average private car, in particular a 5-door Opel-Astra, which has an average fuel-consumption of 4.2 litres/ 100 km. Based on the information provided by the EN 16258:2012 – *Methodology for Calculation and Declaration of Energy Consumption and GHG Emissions of Transport Services (Freight and Passenger)*, published by CEN[7], the greenhouse gas (GHG) emissions were computed as CO_2 equivalents (CO_2e), and specifically the well-to-wheels value, measured as kg CO_2e / km (see Table 7).

Since the fuel-consumption of the vehicle is considered to be an average value, the amount of the resulting CO_2 emissions depends solely on the length of the computed route. As a result, we adopt the GHG emissions of the shortest path with respect to the distance (rather than travel-time) metric, from a source s to a destination t, as the baseline in our experiments.

For each alternative graph we compute the average eco-footprint for all routes that they are included in it. The corresponding optimal distance-based st-route is of course expected to provide the minimum eco-footprint (free flow mode), but on the other hand it is typically a suboptimal route with respect to travel-times. The average deviation of the eco-footprint for an alternative graph (from s to t) is compared with respect to this baseline GHG emission, and it is provided for all the alternative path routing algorithms of this section that we experimentally tested. In all our experiments we consider Diesel to be the type of fuel used by the vehicle, which means that the well-to-wheel value is $gw = 3.24CO_2e$. In each case, the computation of the total CO_2e emissions of a path p is computed by the formula:

$$CO_2e(p) = \text{total fuel consumption} \cdot gw = distance(p) \cdot \text{fuel consumption per } km \cdot gw$$

Table 31 demonstrates the comparison on the eco-footprint quality of the resulted altenative graphs. We observe that Penalty achieves better average eco-footpint. This is mainly due to the fact that it prevents averageDistance to get high. On the other hand, Plateau produces longer paths and therefore its resulted eco-footpint gets worse.

[7] http://www.transport2020.org/newsitem/cen-publishes-european-standard-for-calculation-of-ghg-emissions

map	Plateau	Penalty	Plateau & Penalty
B	23.83 / 9.90%	18.51 / 7,69%	20.81 / 8,65%
BE	13.69 / 5.03%	10.53 / 3,87%	11.21 / 4,12%
IT	21.85 / 6.31%	14.68 / 4,22%	16.24 / 4,79%
GB	19.95 / 5.58%	12.11 / 3,39%	14.15 / 3,86%
FR	20.81 / 5.82%	13.81 / 3,86%	15.94 / 4,46%
GE	20.27 / 4.36%	15.20 / 3,27%	16.34 / 3,51%
WE	22.10 / 5.64%	19.11 / 4,88%	20.45 / 5,22%

Table 31: Average Eco-footprint (CO_2e emissions) and Deviation(%) from optimal Eco-footprint using Penalty, Plateau and their combination.

5.5 Visualization of Alternative Graphs

In Figures 10, 11, 12 and 13, we demonstrate some of the visualized results [8] we got with our alternative route planning implementation.

Figure 10: Improved Penalty method. Shape of AG in Italy.

[8] The images produced by Google Maps © mapping service.

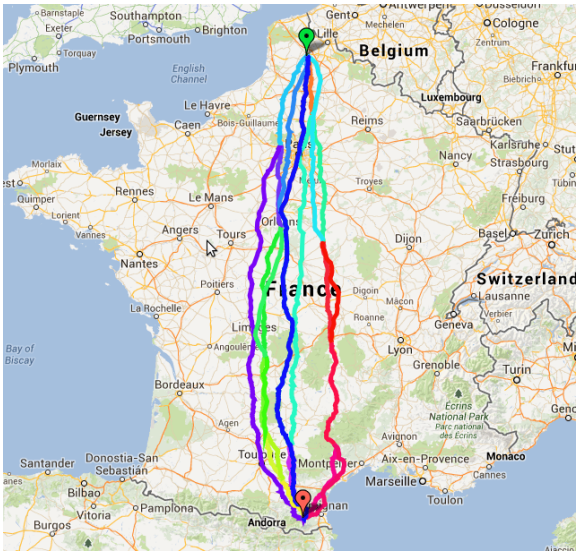


Figure 11: Improved combination of Penalty and Plateau methods. Shape of AG in France.

Figure 12: Improved Plateau method. Shape of AG in Spain.

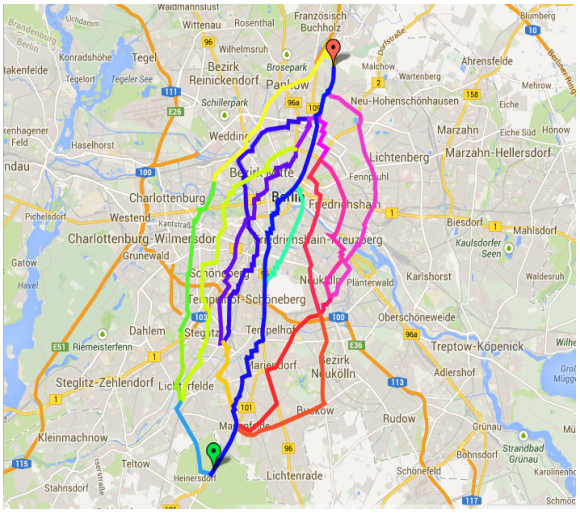


Figure 13: Improved combination of Penalty and Plateau methods. Shape of AG in Berlin.

6 Robust Route Planning

6.1 Introduction

Given two places in a road network, the standard goal in route planning is to compute a quickest route between them. This task can be modeled as the well-known *shortest path* problem: the road network is represented by a graph with vertices corresponding to crossings, edges corresponding to roads connecting the crossings, and the goal is to find a shortest path with respect to edge weights that typically correspond to travel time estimates. However, when a computed route is traveled in reality, the travel time is influenced by various factors such as the weather, the traffic situation, the amount of road work along the route, and so on. Some of these factors can be taken into account by replacing static edge weights with time-dependent ones. The problem becomes then to find a time-dependent shortest path, usually referred to as the *quickest path* problem. Unfortunately, not everything can be modeled easily using time-dependency. A typical example is given by factors that appear often but not regularly, like traffic congestions. In the presence of such factors, one often seeks *robust* routes instead of just fast ones. In loose terms, the quality of a robust route is given by both the average and the variance of its travel time in the typical traffic situations. A slower road through the countryside that hardly sees a car per day might in this sense be considered more robust than a fast highway that is often congested.

Within the project, we follow the approach proposed by Buhmann *et al.* [51] for finding robust solutions of general optimization problems. Applied to the quickest path problem, the method works as follows.

Let $G = (V, E)$ be a directed graph with edge weights $w : E \times T \to \mathbb{N}$ defined for a given time horizon T. A *path* P is a sequence $\langle v_1, ..., v_k \rangle$ of vertices $v_i \in V$, $1 \leq i \leq k$, where $(v_i, v_{i+1}) \in E$ for $i = 1, ..., k-1$, and P is called a *simple path* iff $v_i \neq v_j$ for each $i \neq j$. We overload the weight function w to express the travel time of a path $P = \langle v_1, ..., v_k \rangle$ departing at time $\tau \in T$ as

$$w(P, \tau) = \begin{cases} 0 & \text{if } k = 1 \\ w((v_1, v_2), \tau) & \text{if } k = 2 \\ \tau' + w((v_{k-1}, v_k), \tau + \tau') & \text{otherwise,} \end{cases}$$

where $\tau' = w(\langle v_1, ..., v_{k-1} \rangle, \tau)$ is the travel time of P without the last hop. Note that in the above definition we do not allow waiting at vertices even though it could be beneficial if the weight of an

edge decreases dramatically over time. However, in road networks this is not the case.

The approach by Buhmann *et al.* [51] assumes that an unknown *problem generator* $\mathfrak{PG}$ generates related instances that differ due to noise. Nothing is known about the noise or $\mathfrak{PG}$ itself, and all we are given are two instances I_1 and I_2 (i.e., concrete travel times for a given time period) generated by $\mathfrak{PG}$. For $i \in \{1, 2\}$, the travel times of I_i are given by a weight function $w_i : E \times T \to \mathbb{N}$. The goal is to compute a robust solution that is likely to be good for a future (yet unknown) instance I_3 from $\mathfrak{PG}$. This model fits quite naturally with the quickest path problem under uncertainty where instances represent the traffic situation on different days. For example, we could be given the travel times for last Monday and Monday two weeks ago, and our wish is to plan a robust route for next Monday.

Since nothing is known about the underlying noise, it is a natural choice to consider only paths that are good for both I_1 and I_2. From the set of all s-t paths $\mathcal{P}$ we compute the *approximation sets* $A_\rho(I_1)$ and $A_\rho(I_2)$ where, for $i \in \{1, 2\}$, departure time $\tau \in T$ and a suitable value $\rho \geq 1$ (we explain the meaning of "suitable" later on),

$$\begin{aligned} A_\rho(I) &:= \{P \in \mathcal{P} \mid w_i(P, \tau) \leq \rho \cdot OPT_I\}, \\ OPT_I &:= \min_{P \in \mathcal{P}} w_i(P, \tau). \end{aligned}$$

We then pick a path at random from the intersection $A_\rho(I_1) \cap A_\rho(I_2)$ of the two approximation sets. As a "suitable" ρ, Buhmann *et al.* propose to choose the value that maximizes

$$\frac{|A_\rho(I_1) \cap A_\rho(I_2)|}{|A_\rho(I_1)| \cdot |A_\rho(I_2)|}. \tag{2}$$

During an extensive evaluation of this approach [52], we observed that the value of ρ maximizing (2) most often corresponds, at least on the data provided by TomTom for the project eCOMPASS, to the first value for which the intersection $A_\rho(I_1) \cap A_\rho(I_2)$ is not empty, the so-called *first intersection*. In light of this observation, the idea of heuristically approximating a robust path with a path belonging to the first intersection of two given instances comes naturally.

In this section of the deliverable, we report on an experimental evaluation of a series of algorithms for the computation of a path in the first intersection [55]. We evaluate the runtime of these algorithms in order to assess their suitability for real-world applications.

6.2 Computing the Pareto front

An interesting property of the first intersection is that it is tightly related to the Pareto front of all s-t paths in the graph $G = (V, E)$ with time-dependent bi-criteria edge weights

$$w(e, \tau) = \begin{pmatrix} w_1(e, \tau) \\ w_2(e, \tau) \end{pmatrix}. \tag{3}$$

In particular, a path in the first intersection can always be found among those in the above Pareto front.

In a theoretical study within the scope of eCOMPASS [55], we considered a known algorithm for the computation of Pareto fronts in graphs with edge weights as in eq. 3 and show different ways to apply a speed-up technique known as *bidirectional search* to it. In the following, we briefly recall the resulting algorithms in order to experimentally assess their feasibility from a practical point of view.

Unidirectional search. A basic algorithm for the computation of Pareto fronts in graphs with static bi-criteria edge weights was introduced by Hansen [53] and extended by Martins [54] for weights with more than two criteria.

Martins' algorithm allows an almost straightforward generalization to time-dependent weights. However, the time-dependent variant is only correct if both criteria in the edge weights satisfy a property known as *FIFO property*. In settings where this property is not satisfied, different and in general less efficient algorithms must be used. Since in our data this property is (almost always) satisfied, we restrict our attention to this algorithm.

Bidirectional search. One of the most well-known technique for speeding-up routing algorithms is *bidirectional search*. It is well-known how to apply this technique efficiently for the computation of static shortest path, quickest paths, and Pareto fronts in graphs with static bi-criteria edge weights.

We used this technique to design a three phases bidirectional algorithm for the computation of Pareto fronts in graphs with time-dependent bi-criteria edge weights [55]. This algorithm employs, in the backward direction, two independent searches working in parallel, one for each criterion. We also showed how to parametrize this algorithm for a value $K \geq 1$ in order to compute a K-*approximation* of a Pareto front.

6.3 Computational results

We now present an experimental evaluation of the unidirectional and bidirectional searches introduced above. The experiments were performed on the high-performance cluster of ETH Zurich, Brutus [58]. Each experiment was run on a single core of a computation node of the cluster. The results shown in the following refer to computation nodes with AMD Opteron 8380 processors clocked at 2.5 GHz and 32 GB main memory. The code was written in C++ and compiled using GNU C++ compiler version 4.8.2 and optimization level 3.

Input. The algorithms were tested on a road network of the area around Berlin and Brandenburg with ∼0.5 million vertices and ∼1 million edges. Each edge of the network is paired with a table providing speed values on the corresponding road. These speeds are in the form of estimations at discrete intervals every 5 minutes for each day of the week. The estimations are computed by TomTom with a proprietary algorithm taking as input measured travel times on each road collected for a period of roughly two years. We interpolated linearly the speeds provided by the above tables to obtain values for times within the 5 minutes windows.

Additionally, we were provided live measurements (also called *probes*) that indicate, for a fixed road and a fixed absolute point in time, the actual speed on that road at that time. These live measurements cover a period of two weeks, from March 18th 2012 to March 31st 2012. However, since they are very expensive to collect, we were given only a limited quantity of them. In a 30 minutes window inside those two weeks, typically less than 22% of the edges of the network receive at least one probe. We decided for our implementation that, when available, the speed indicated by a probe replaces the one provided by the time-dependent function and that the new speed remains valid for 5 minutes. Note that, in general, a probe may break the FIFO property of the underlying time-dependent function; this indeed happens for some cases in our data. However, the number of probes is so limited and the difference in speed with the underlying function is so small that we decided to ignore this discrepancy and treat the edge weights as if they still satisfy the FIFO property.

Setup. To obtain two different instances (edge weight functions) as required, we consider departure times in two different days. Each s-t path in I_1 departs on Tuesday, March 20th 2012 at 17:00 (CET), and in I_2 on Wednesday, March 21st 2012 at 17:00 (CET). The following tables show results for 10,000 s-t pairs taken uniformly at random from the set of all vertices.

We assess the robustness of the routes computed with our method against a competitor denoted as AVG. AVG computes a quickest path in a graph with single-criteria time-dependent edge weights,

	Average	Variance	Max
AVG	1.015	$7.366 \cdot 10^{-4}$	1.562
SIM	1.012	$5.678 \cdot 10^{-4}$	1.590

Table 32: Ratio over the quickest path in I_3

	Runtime (avg)	Runtime (max)	Labels (avg)	Labels (max)
Unidirectional	7.823	166.95	1,256,751	24,129,100
Bidirectional	1.668	44.25	408,289	7,648,531
Bid. $K = 1.2$	1.183	49.06	288,568	7,569,067
Bid. $K = 1.4$	0.856	27.63	240,332	5,696,411
Bid. $K = 1.6$	0.803	22.74	236,136	5,696,411
Bid. $K = 1.8$	0.887	28.78	235,458	5,696,411
Bid. $K = 2.0$	0.879	24.93	235,119	5,696,411

Table 33: Runtime in seconds and number of scanned labels

where the weight w of an edge e at time τ is

$$w(e, \tau) = \frac{w_1(e, \tau) + w_2(e, \tau)}{2}.$$

That is, AVG computes quickest paths in the graph where the edge weights are averaged between the weights of I_1 and I_2. A simple calculation shows that, if w_1 and w_2 satisfy the FIFO property, also their average does.

6.3.1 Results

The results shown in the following aim to assess the quality of the routes computed, and the time required to compute them.

Quality. To define the quality of a robust route we consider its weight in a third instance I_3, corresponding to departure time Thursday, March 22nd 2012 at 17:00. The quality of a path is the ratio of its weight in I_3 over the weight of the quickest path in I_3.

Table 32 shows the average ratio, its variance, and the maximum ratio obtained by each of the two competitors among all 10,000 *s-t* pairs. The method SIM is the one that returns a path in the first intersection of I_1 and I_2. It appears evident from Table 32 that the paths returned by SIM are more robust than those returned by AVG, both in terms of average and variance. However, the maximum ratio of AVG is smaller than that of SIM. It might be interesting to inspect further this fact to better understand the difference between the routes returned by the two methods.

Runtime. The time required to compute a robust route is measured in CPU time in seconds. Furthermore, a machine-independent measure is also provided, that is the number of labels scanned by the algorithm. The counter of labels scanned for the bidirectional algorithm is increased for each iteration of the forward search, and for every two iterations of each of the backward searches.

Table 33 shows the runtime and the number of labels scanned of the unidirectional and bidirectional algorithms presented in the previous section. It can be seen that the improvement given by the bidirectional search is quite substantial. In particular, the speed-up of bidirectional over unidirectional is a factor of more than 4.

Furthermore, the runtime of the bidirectional algorithm steadily decreases if we allow approximation factors K greater than 1. It appears however that there is a limit to the speed-up that can

	AVG	SIM
Emissions	1.041	1.088

Table 34: Emissions of robust routes over optimum in I_3

be obtained via approximation. The reason for this is that values of K greater than 1 only reduce the time spent by the algorithm in phase 2, since the termination condition is met earlier in time (see [55] for more details). Therefore, there exists some value of K, say K^*, for which the time spent in phase 2 is null. All the values of K greater than K^* will result in the same runtime and number of scanned labels. Looking at Table 33, it appears that, for our data, this value is around 1.4.

Remark: It should be observed that the presented results are in some sense biased by the application we are considering, that is, robust routing. A peculiarity of this application as we modeled it is that each criterion in an edge weight corresponds to a travel time for a different day of the week. If the two days considered are in some sense related like, for example, two working days as opposed to a working day and a Sunday, we can expect this correlation to somehow appear in the edge weights values as well. That is, the weights of a path in the two instances are quite likely similar. This implies that our data will have some special features that are not usually found in more general applications of time-dependent bi-criteria routing. One of these features is that the average size (i.e., number of paths) of a Pareto front is quite small; for our setting it is around 5.5. In general, small Pareto fronts are not likely, and the typical size of a Pareto front is much more than 5.5. Furthermore, the bigger the sizes of the Pareto front are, the slower Martins' algorithm gets. It is an interesting open question to assess the efficiency of the proposed algorithms in more typical applications of time-dependent bi-criteria routing problems.

6.3.2 Eco-footprint of robust routes

For the project eCompass, the ecological impact of a route on the environment is as important as its travel time. An estimation of the eco-footprint of a given route is usually provided as a measurement of the CO_2 emissions of cars driving on it. In the calculation of the CO_2 emissions several factors can be considered, like, for example, speed, acceleration, deceleration, driving patterns, etc. Unfortunately, models that consider all possible factors are quite complex and require expert knowledge even for the simple task of applying them. For the purpose of a preliminary investigation, however, simpler models might already be sufficiently accurate and easier to apply. In the following, we adopted on of the simple models for the estimation of the CO_2 emissions of cars driving on the robust routes computed by the above algorithms.

For certain specific cars, it is possible to obtain charts and tables that specify the fuel consumption for different speeds and gears. From the fuel consumption, rough estimations of the CO_2 emissions can be obtained using standard methods [57].

Table 34 shows these estimations for a Volkswagen Golf [56] (a common 1400cc car) in terms of the average ratio of the emissions when driving on a robust route over the emissions when driving along the quickest path in I_3 for all the 10,000 s-t pairs considered. As it can be seen from the table, it appears that the "price of robustness" for the environment is not high. Eco-aware drivers that have a need for robustness can drive along these routes with a clean conscience. Furthermore, the use of more accurate models for the estimation of CO_2 might show an additional benefit for the environment that is not captured by our current model, that is the avoidance of traffic congestions.

6.4 Further improvements

The computation of robust routes using the approach by Buhmann *et al.* [51] presents some peculiarities that do not usually arise in standard routing applications. One of these features is the

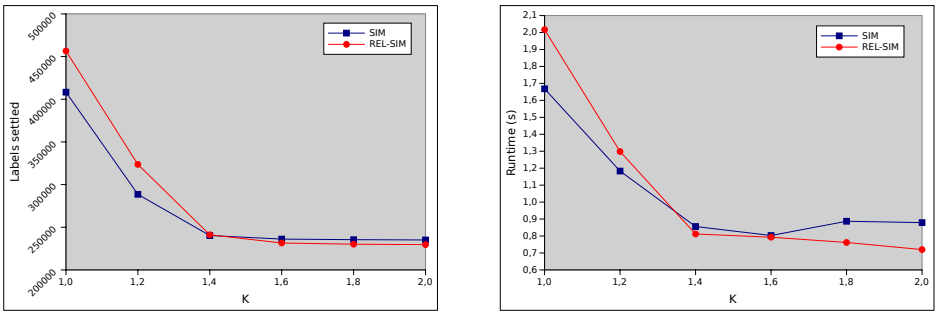


Figure 14: Comparison of bidirectional algorithms

fact that both criteria of the time-dependent edge weights of the graph, as defined in (3), represent travel times in different days of the week. If the days considered are somehow related like, for example, two consecutive Mondays, we can expect that the travel times on same roads to not differ much.

In light of this observation, we designed a modified variant of the bidirectional algorithm presented above exploiting the correlation of related days [55]. The modified algorithm works in a similar manner as the original one, but it replaces the two independent backward searches on each criterion with a single backward search that uses rougher bounds that are however faster to compute. The efficiency of this algorithm is a tradeoff between how rough these bounds are and the increased simplicity of the backward search. We expect that, in some applications, one overcomes the other while in other applications the opposite happens. Also the modified algorithm can be parametrized by a value $K \geq 1$ to compute approximate Pareto fronts.

6.4.1 Experimental results

Let SIM denote the original bidirectional algorithm and REL-SIM its modified variant. We present in the following an experimental evaluation of REL-SIM under the same conditions considered for the previous experiments and for the same 10,000 s-t pairs.

Figure 14 shows a plot of the average runtime and the number of labels settled by the two algorithms for different values K of approximation. It can be seen that, for $K = 1.0$, SIM is faster and settles less labels than REL-SIM. However, the difference between the two decreases for increasing values of K until $K = 1.4$, where the latter takes over the former.

The reason for this can probably be explained as follows. As we observed already, REL-SIM compromises between using rougher bounds and a simplified backward search. However, the penalization might not be too bad if the simpler backward search is much faster than in the original algorithm. Additionally, larger values of K correspond to an earlier termination of phase 2 and of the backward search. Starting with $K = 1.4$, it seems that the sum of these two factors gets relevant enough to appear in the overall computation time.

7 Fleet-of-Vechiles Route Planning

7.1 Vehicle Routing Problem Data

In this Section, we examine the differences between real world data and synthetic data. The approach followed by eCOMPASS was to first test our algorithm with synthetic data to verify that it behaves as expected. The second step was to test the algorithm with real life datasets, that were provided by PTV.

7.2 Laboratory test data compared to real life data

In both cases, laboratory and real world, the data model is the same but the scope addresses two different worlds. The laboratory test data is used most often to test the function of modules and the performance of the solution. Real world data in contrast to laboratory data has to deal in many cases with complex data sets which describe specific problems. Even the problem analysis stage is not trivial in practice. In practical scenarios, the problems are not clear or well defined. A precise classification is not so easy, as many real problems include characteristics and features of more than just one model. Thus it is a real challenge to provide solution procedures that match real world practitioner needs and expectations.

7.3 Richness of real world problems in VRP

In an operative setting, real world VRP problems do not come with an unlimited homogeneous fleet. Instead we have to deal most often with different types of vehicles and limited availabilities. It goes without saying that this probably imposes new constraints and new aspects on the original problem.

For an acceptance in practical settings it is very important to have a reasonable network model that allows the calculation of reliable distances and driving times. There may exist different routes for different types of vehicles, for different loads and cargo or for different times of the day.

Further aspects of richness are the presence of multiple customer time windows with different kinds of service. In real world problems we distinguish between start of service intervals and full service intervals. The correct handling of working hours regulations increases the degree of complexity considerably.

Often real world problems do not focus on one problem but deal with multiple objectives, such as service level, social criteria, robustness, ecological criteria and visual attractiveness.

7.4 Operative setting of real world problems

In an operative setting, planning is a process. Dispatcher works systematically towards certain objectives. In addition to the algorithm that supports planning, he performs manual operations: Insertions, relocations of customers, assigning a certain vehicle to a tour or vice versa assigning tours to a vehicle.

In operations, he has to deal with modifications or cancellations of orders. Tours may have different states, e.g. special states can limit the degrees of freedom for modifications; e.g. if the loading for a tour has already started it might be desired that this tour shall keep its vehicle. Thus data does not remain static but behaves dynamically.

In professional settings typically the planning is carried out in a multi-user mode. Multiple planners are involved with dedicated tasks and rights. The planner can relax constraints to allow the actions. Of course the planner can overrule each decision of the system. Furthermore, an appropriate IT-infrastructure is required to match all the requirements. The logic model must ensure that it is possible to partition and share the planning data correctly, according to defined rules and concepts. The IT-infrastructure has to physically support and implement the logic model, e.g. it has to be decided whether the model shall support concurrent-competitive or cooperative work modes.

7.5 Synthetic Laboratory Test Data

Regarding laboratory test data the question to ask is what is needed to perform a meaningful test. Often only a function or a working hypothesis has to be proven. In this case simplistic data without high complexity may be sufficient to perform the verification. Of course some functionality tests,

especially regarding performance, may require more complex test data to achieve a meaningful result.

Besides data availability, the main reason to use laboratory data is the possibility to generate data which fulfils all test requirements without introducing additional complexity. In essence laboratory can manipulated to reflect real world problems. For this manipulation, synthetic test data adapts data from real world problems and applies the restrictions and constraints to the synthetic data set.

A further advantage of laboratory tests is, to verify algorithmic functions in a controlled environment without uncontrollable influences.

7.6 eCOMPASS Approach Regarding Fleets of Vehicles

One of the most challenging tasks for eCOMPASS regarding fleets of vehicles, is to develop an algorithm that takes into account the ecological impact of the tours of fleets of vehicles. For this reason, a new three phase approach was developed that tries to group customers together in order to be served by a vehicle. The main idea of the eCOMPASS approach is the following:

- Phase I tries to group together customers regarding their time windows. A graph $G = (V, E)$ is constructed where each customer is represented by a vertex $u \in V$, and there is an edge $e_{u,v}$ connecting nodes (customers) u, v if their time windows are compatible. This means that if a vehicle serves customer u it can also serve customer v without violating any time constraints. At the end of Phase I customers are grouped together into clusters.

- Phase II tries to group together customers taking into account their geographical location. A geographic partition is performed and customers are grouped into cells. All customers that belong to a cell C are close together regarding the real distance among them. At the end of Phase II customers are grouped together in cells.

- Phase III is a refinement phase. It tries to split or merge clusters and cells created from the previous phases. The main idea is that if some customers that are close (regarding real distance) and have compatible time windows are merged together into a final group. On the other hand, if a cell contains customers that are close but have incompatible time windows this cell must be split into two groups.

The ecological aspect is taken into account implicitly. The final clusters that are created have the property that all their customers are close together and have compatible time windows. Thus, a vehicle can serve them without wasting time going back and forth to the depot or travelling with low load. More details regarding eCOMPASS approach can be found in D2.2.

7.7 Experimental Study and Data Sets

The main benefit of the eCOMPASS approach of balanced and compact trips is to provide trip structures are stable during the execution phase in case of unplanned events (e.g., unplanned multiple stops, additional stops). As the available existing solutions do not cover this target in a meaningful way, a direct comparison between the eCOMPASS approach against existing optimized (for a set of different criteria) VRP solutions is not the focus of our experimental study. Consequently, the experimental study focuses on two aspects: 1) to prove the functionality of the eCOMPASS algorithm at a generic level 2) to achieve an understanding of the tradeoff between compact and balanced eCOMPASS solutions in comparison to baseline solutions of existing state of the art VRP approaches. The experiments therefore compares eCOMPASS solutions of the Munich data sets against baseline instances of PTV which focused on different optimization aspects.

To achieve the above goals, we conducted our experimental study on three real-world data sets provided by PTV. The first, is a data set in the city of Milan (Italy). The other 2 data sets

include customers located in the city of Munich. Specifically, one regards a parcel delivery and the other a furniture delivery. All Munich data sets are in urban areas. All data sets provide the following information: total number of customers, a unique customer id, a location of each customer (longitude,latitude), one (or more) time window(s) of each customer, the weight of each customer (a number representing the amount of goods that have to be delivered) and a distance matrix with the real distance among all customers.

The quality measures that are reported are: total driving distance (in km), number of vehicles used for each scenario, number of tours and number of tour stops. For the Milan dataset, a comparison was made between the routes computed with the real distance against the routes computed with the Euclidean distance.

7.7.1 Milan Dataset

The Milan dataset consists of 1000 customers and is the largest dataset on which we conducted experiments. Due to lack of quality measures of other approaches we did not perform a comparison with the eCOMPASS approach. However, we report on the ratio between the total distance travelled using the real distance and the total distance travelled using the Euclidean distance. Our experiments showed that this ratio is 1.75, a number that is acceptable because in urban areas the distance between two points is typically Manhattan, i.e. at least greater than 1.41 times bigger than the Euclidean distance.

7.7.2 Munich Dataset - Parcel Delivery

The tour planning results for the parcel courier express service providers are listed in Table 35. Without traffic information a total tour length of 163.32 km for serving 32 customer orders was calculated. For the process of delivery one vehicle is needed for generated tour. In Figure 15, all 32 customers are shown on the map. Customers are grouped together creating clusters.

	Previous Approach	eCOMPASS Approach
Total km driven	163.32	114.01
Total driving time	4h 12 min	4h 32min
CO_2 emissions	62.45kg	41.33kg
Total vehicles used	1	1
Number of tours	1	1
Tour stops	34	34

Table 35: Munich Dataset: Performance indicators for the parcel delivery scenario. The vehicle type chosen for CO_2 emissions calculation was truck (7,5t).

In Table 35, the eCOMPASS approach achieves a further improvement in total kilometres driven. The generated tour takes 48 minutes longer and part of the tour uses the motorway.

Figure 15: Munich Dataset: Groups created for the parcel delivery scenario. Each customer is represented by a marker. In this case, all customers form one group and are served by one vehicle.

7.7.3 Munich Dataset - Furniture Delivery

The second scenario to be considered is the delivery of furniture, in particular kitchen furniture from a furniture store to various customers in the city centre of Munich. The furniture store with its warehouse is located outside of Munich in the district of Taufkirchen. For this scenario it is assumed that the furniture can be ordered directly in the furniture store by the customer and every piece of furniture is available from the stock. Thus, a suitably short period of time between the point of order and delivery will be accepted. For simplicity the furniture store's warehouse is operating all the time.

After the customers chose the pieces of furniture they wish to receive, the warehouse processes their orders and the delivery will be planned. As furniture is often bulky, the delivery process of the furniture is modelled as mid-size truck operations. We modelled the distribution process with two trucks and assumed 5 tons payload. Furthermore we assumed service time of 15 minutes for a drop per truck stop for unloading the pieces of furniture. As a consequence, a vehicle is not immediately ready for use again after the point of delivery. After finishing the tours the trucks return to the furniture store/warehouse. The vehicle fleet we modelled consists of two mid-size lorries with 5.000 kg payload and an overall weight of about 12.000 kg per lorry.

For the case of furniture delivery the calculations are based on a data set with 150 entries for a time period of about two weeks. The handled information are real, but made anonymous. For the calculation and tour planning two trucks with 5 tons payload were used with an availability of 24/7. The only restrictions for tour planning are the weight of the transported pieces of furniture and a service time per tour stop of 15 minutes to guarantee the unloading process. For simplicity, the delivery time windows, in which customers can receive their furniture were standardised from 08:00 to 18:00 o'clock and Monday to Friday. As mentioned already in the other scenarios the order specifications on each of the both Mondays are identical making them comparable in the case of traffic information. The database contains 31 orders for each Monday. For the initial tour planning solution the results are shown in Table 36.

Based on the given information without any traffic the following tours for the furniture delivery on Monday were generated. There are three tours operated by two vehicles to serve all 31 customers. The visualization of the furniture delivery scenario is shown in Figure 16.

In Table 36, the eCOMPASS approach achieves a further improvement both for total kilometres driven and total driving time. The tours generated do not use the motorway.

	Previous Approach	eCOMPASS Approach
Total km driven	204.36	103.15
Total driving time	4h 29min	4h 06min
CO_2 emissions	115.46kg	57.61kg
Total vehicles used	2	2
Number of tours	3	3
Tour stops	37	37

Table 36: Munich Dataset: Performance indicators for the furniture delivery scenario. The vehicle type chosen for CO_2 emissions calculation was truck (7,5t).

Figure 16: Munich Dataset: Groups created for the furniture delivery scenario. Each customer is represented by a marker. In this case, customers are divided into 3 groups, served by two vehicles that perform three tours.

7.8 Experiments with Large Datasets

7.8.1 Partitioning Methods.

The next step was to consider larger datasets and compare all three geographic partition methods. There are three datasets: Malta, Hamburg and Germany. All datasets consist of 1000 customers. Malta as a small geographical area, Hamburg as a middle scale geographical area and Germany as a large geographical area. The aim of this experiment is to focus on the behavior of the partition techniques on these geographical areas of varying size. We consider that every customer is available for delivery the whole day. Hence, their time windows are 24 hours long. The results are reported in Table 37.

	Malta		Hamburg		Germany		
	km	CO_2	km	CO_2	km	CO_2	Areas
Quad Trees	556	311.36	6917	3873.52	23852	13357.12	16
KaHIP	536	300.16	6703	3753.68	24072	13480.32	10
Natural cuts	479	268.24	6341	3550.86	23158	12968.00	10

Table 37: Performance indicators for large datasets. The numbers report total km driven and CO_2 emissions in kg for every partition method and areas of partition.

A first observation is that the natural cuts technique is more suitable for road networks. KaHIP

technique performs better than quad trees which were outperformed by the other two techniques. Another interesting observation is that for bigger geographical areas such as Hamburg and Germany the improvement in total distance is higher when the natural cuts technique is used. Since time windows are constant for each customer, the difference in the results lies in the method used for geographic partition.

7.8.2 Time Windows and Dynamic Scenarios.

In this case we take into account time windows. The time windows range is: 08:00 - 12:00, 12:00 - 16:00, 16:00 - 20:00 and 20:00 - 23:00. Each customer is assigned a time window at random. Then, we compute the new tours using our approach. The results are reported in Table 38. In order to make a comparison, it is customary (in the logistics sector) to solve the problem without any constraint (time window, partition method, number of vehicles available). This will yield an ideal solution that will not be reached, as we add constraints such as time windows and number of available vehicles. This procedure is called free planning. Hence, we compare our approach against free planning to determine how far away we are from the free planning solution.

	Malta		Hamburg		Germany	
	km	Difference (%)	km	Difference (%)	km	Difference (%)
Quad Trees	587	24.6	7077	9.1	25264	9.0
KaHIP	575	22.0	7059	8.9	24933	8.5
Natural cuts	493	4.6	6866	5.9	25039	9.0
Free Planning	471	0.0	6481	0.0	22969	0.0

Table 38: Performance indicators for large datasets against free planning. The numbers report total distance driven in km. Difference shows how far away our approach is from the free planning case.

As expected, our solutions are inferior than those of free planning but they are away by a small margin. The presence of time windows makes the tours longer since they pose an additional constraint. Another observation is that the natural cuts method performs better for Hamburg and Malta datasets while for Germany the best method is KaHIP.

One of the advantages of the suggested approach is that it can be used in an online environment. In real life, one or more customers may cancel their order unexpectedly. On the contrary, some new customers may appear that need to be served as soon as possible. The way many planners deal with such cases is to run a planning algorithm from scratch. In our case, since we have formed clusters we can easily check in which cluster to assign a new customer. The cancellation of an order is treated easily. We just remove the customer that canceled his order from the route. This will not affect the whole tour since all customers that belong to this route have compatible time windows.

For the experiments we used the datasets of Malta, Hamburg and Germany and we created three dynamic scenarios. The Incremental Scenario adds 20 additional customers with random coordinates to the initial dataset. The Decremental Scenario removes 10 customers from the initial dataset. In the Fully Dynamic Scenario there is a sequence of 20 insertions and 5 deletions of customers. The number of customers that were inserted/deleted was deliberately kept small since in real life cases a large number of new unexpected orders or cancellations is unlikely to happen. The results are shown in Table 39. We report total distance in km. As expected, the Incremental scenario computes routes that cover more total distance than the initial scenario. On the other hand, the Decremental scenario computes routes that cover less total distance than the initial scenario. The Fully Dynamic scenario computes routes that cover more total distance than the initial scenario since there are more insertions to the tour than deletions.

	Initial	Incremental	Decremental	Fully Dynamic
Malta	479	492	459	487
Hamburg	6341	6367	6312	6358
Germany	23158	23204	23129	23179

Table 39: Initial datasets and dynamic scenarios are presented. The numbers report total distance driven in km.

8 Conlusions

In deliverable D2.4, we have assessed via thorough experimental evaluations the success of the algorithmic solutions developed within WP2 traffic prediction, route planning for private cars, and routing fleets of vehicles. We have discussed necessary modifications with respect to Deliverable D2.1 an D2.2.1, identified the most applicable and technically most robust algorithmic solutions. The proposed algorithmic solutions have considerably advanced the state of the art, and their success was also experimentally verified. In particular, our techniques allow for faster, more robust, precise routes which also demonstrate eco-friendliness of the proposed solutions in our experiments. Some of the proposed solutions were also integrated by WP5 partners and successfully piloted in Berlin within the scope of WP6.

For our traffic-prediction mechanisms, we experimentally tested our parametric technique, an improvement of the previously developed Lag-STARIMA technique, and evaluated it against several benchmark methods. We also experimentally tested our non-parametric approach, namely Speed Dynamic Short-Term (SDST) forecasting technique, and compared it against a set of techniques selected from the literature.

For the time-dependent travel-time metric, we experimentally evaluated our novel distance oracles on the real-data of Berlin. In particular, we conducted an extensive experimental study of the proposed oracles for six different landmark sets, achieving remarkable speedups over `TDD`. The speedups that we observed for our query algorithms over the average time of a time-dependent Dijkstra run, may be up to 723, with average relative error less than 1.634%. Analogous speedups are observed if our quality measure is not the computational time, but the (machine-independent) number of settled vertices of the query algorithms. The best possible observed relative error is indeed much better than the theoretical bounds provided by the analysis of the query algorithms. In particular, it can be as small as 0.298% (for 2000 KaHIP landmarks). The corresponding speedup is then 118.

If we focus on the absolute response times, we manage to provide responses (via `FCA`) to arbitrary queries, in times less than $0.4ms$ for all landmark sets that we used, with relative error no more than 2.201%. For relative error at most 0.701%, we can provide answers in no more than $1.345ms$ using $\texttt{FCA}^+$, for all the considered landmark sets.

As for the preprocessed data, we create and succinctly store roughly $300K$ approximate travel-time summaries from a given landmark, in average sequential time less than $40sec$. That is, the amortized sequential time per approximate travel-time summary is no more than $0.134ms$.

Our future plans concern the parallelization of both the preprocessing phase and the query algorithms, since no attempt has been made so far to exploit the inherent potential of parallelization in them, which would significantly speed-up the execution times. In particular, parallelizing the preprocessing phase is straightforward and would significantly improve the adaptivity of our oracle to live-traffic reports of unforeseen disruptions (e.g., temporal congestion, or even blockage of a particular road segment). Moreover, there are still many possibilities of improving the required preprocessing space, by exploiting the one-to-all flavour of the constructed travel-time summaries. All these issues are part of our ongoing research towards truly efficient time-dependent distance oracles.

We have also extended Contraction Hierarchies to a three-phase speedup approach. While com-

mon two-phase speedup techniques achieve fast route planning queries at the expense of substantial and slow preprocessing, we split preprocessing into metric-independent preprocessing of the road topology and a lightweight preprocessing of a given metric (*customization*), while still offering fast query times. We demonstrated that our Customizable Contraction Hierarchies (CCH) approach is practicable and efficient on real world road graphs, achieving customization speeds of below one second and query below one millisecond on the standard benchmark instance of DIMACS Europe. Furthermore, we have performed an extensive experimental analysis of its performance that hopefully sheds some light onto the inner workings of Contraction Hierarchies. Our experiments clearly show that the running times are completely independent of the metric used (if it can be expressed as a scalar value after customization). This means that the technique is well-suited to work with even highly detailed user-centric metrics that aim at offering a favorable trade-off between vehicle restrictions (e. g., width, height), user preferences (e. g., avoiding highways, preferred maximal driving speeds, disliked areas) and environmental-friendliness. At the same time, our technique supports fast, responsive query times and a light-weight preprocessing that easily enables consideration of current traffic conditions. The overall CCH workflow in a server-based production system would be as follows: Run the metric-independent preprocessing of the node order when generating a new map release (e. g., every three months). When a user logs into the service or changes her preferences, run the metric-dependent customization in below one second. To account for the current traffic situation, either re-run full customization in below one second or apply our partial update algorithm on only the changed road segments for even faster performance. After customization, queries can be answered in below a millisecond on average (on a single core, leaving resources to parallelize over multiple users' queries). For future work outside the time-frame of eCOMPASS, we will further consider functional metrics (where edge weights dependend on the current time of day or the current state of charge of the vehicle's battery). For that, we will investigate combining CCH with techniques from Section 3 and ECOMPASS-TR-028.

We also experimented on the Penalty and Plateau based methods [63] as well as their combination, and we extended them in several ways. Now a large number of qualitative alternatives can be computed in time less than one second on continental size networks along with their eco-footprints. Future work includes the optimization of these algorithms and the development of even stronger heuristic approaches.

References

[1] Corrado De Fabritiis, Roberto Ragona, and Gaetano Valenti. Traffic estimation and prediction based on real time floating car data. In *Intelligent Transportation Systems, 2008. ITSC 2008. 11th International IEEE Conference on*, pages 197–203. IEEE, 2008.

[2] Themistoklis Diamantopoulos, Dionysios Kehagias, Felix G König, and Dimitrios Tzovaras. Investigating the effect of global metrics in travel time forecasting. In *Proceedings of 16th International IEEE Conference on Intelligent Transportation Systems (ITSC)*, 2013.

[3] Benjamin Hamner. Predicting travel times with context-dependent random forests by modeling local and aggregate traffic flow. In *Data Mining Workshops (ICDMW), 2010 IEEE International Conference on*, pages 1357–1359. IEEE, 2010.

[4] Yiannis Kamarianakis and Poulicos Prastacos. Forecasting traffic flow conditions in an urban network: comparison of multivariate and univariate approaches. *Transportation Research Record: Journal of the Transportation Research Board*, 1857(1):74–84, 2003.

[5] METIS – serial graph partitioning and fill-reducing matrix ordering, 2013. Stable version: 5.1.0.

[6] KaHIP – Karlsruhe High Quality Partitioning, May 2014.

[7] Rachit Agarwal and Philip Godfrey. Distance oracles for stretch less than 2. In *Proceedings of the 24th Annual ACM–SIAM Symposium on Discrete Algorithms (SODA'13)*, pages 526–538. ACM-SIAM, 2013.

[8] Hannah Bast, Daniel Delling, Andrew V. Goldberg, Mattias Müller-Hannemann, Thomas Pajor, Peter Sanders, Dorothea Wagner, and Renato Werneck. Route planning in transportation networks. Technical Report MSR-TR-2014-4, Microsoft Research, January 2014.

[9] Gernot Veit Batz, Daniel Delling, Peter Sanders, and Christian Vetter. Time-Dependent Contraction Hierarchies. In *Proceedings of the 11th Workshop on Algorithm Engineering and Experiments (ALENEX'09)*, pages 97–105. SIAM, April 2009.

[10] Gernot Veit Batz, Robert Geisberger, Peter Sanders, and Christian Vetter. Minimum time-dependent travel times with contraction hierarchies. *ACM Journal of Experimental Algorithmics*, 18, 2013.

[11] Hans L. Bodlaender and Arie M.C.A. Koster. Treewidth computations I. Upper bounds. *Information and Computation*, 208(3):259–275, 2010.

[12] K. Cooke and E. Halsey. The shortest route through a network with time-dependent intermodal transit times. *Journal of Mathematical Analysis and Applications*, 14(3):493–498, 1966.

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

[14] Brian C. Dean. Algorithms for minimum-cost paths in time-dependent networks with waiting policies. *Networks*, 44(1):41–46, 2004.

[15] Brian C. Dean. Shortest paths in fifo time-dependent networks: Theory and algorithms. Technical report, MIT, 2004.

[16] Frank Dehne, Omran T. Masoud, and Jörg-Rüdiger Sack. Shortest paths in time-dependent FIFO networks. *ALGORITHMICA*, 62(1-2):416–435, 2012.

[17] Daniel Delling. Time-Dependent SHARC-Routing. *Algorithmica*, 60(1):60–94, May 2011. Special Issue: European Symposium on Algorithms 2008.

[18] Daniel Delling, Andrew V. Goldberg, Thomas Pajor, and Renato F. Werneck. Customizable Route Planning. In Panos M. Pardalos and Steffen Rebennack, editors, *Proceedings of the 10th International Symposium on Experimental Algorithms (SEA'11)*, volume 6630 of *Lecture Notes in Computer Science*, pages 376–387. Springer, 2011.

[19] Camil Demetrescu, Andrew V. Goldberg, and David S. Johnson, editors. *The Shortest Path Problem: Ninth DIMACS Implementation Challenge*, volume 74 of *DIMACS Book*. American Mathematical Society, 2009.

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

[21] eCOMPASS Project (2011-2014). `http://www.ecompass-project.eu`.

[22] Luca Foschini, John Hershberger, and Subhash Suri. On the complexity of time-dependent shortest paths. *Algorithmica*, 68(4):1075–1097, 2014. Preliminary version in ACM-SIAM SODA 2011.

[23] Spyros Kontogiannis, Dorothea Wagner, and Christos Zaroliagis. Hierarchical distance oracles for time-dependent networks. Technical report, eCOMPASS Project, December 2014.

[24] Spyros Kontogiannis and Christos Zaroliagis. Distance oracles for time-dependent networks. In *J. Esparza et al. (eds.), ICALP 2014, Part I*, volume 8572 of *LNCS*, pages 713–725. Springer-Verlag Berlin Heidelberg, 2014. Full version as eCOMPASS Technical Report (eCOMPASS-TR-025) / ArXiv Report (arXiv.org>cs>arXiv:1309.4973), September 2013.

[25] Georgia Mali, Panagiotis Michail, Andreas Paraskevopoulos, and Christos Zaroliagis. A new dynamic graph structure for large-scale transportation networks. In *Algorithms and Complexity*, volume 7878 of *Lecture Notes in Computer Science (LNCS)*, pages 312–323. Springer, 2013.

[26] Giacomo Nannicini, Daniel Delling, Dominik Schultes, and Leo Liberti. Bidirectional A* Search on Time-Dependent Road Networks. *Networks*, 59:240–251, 2012. Journal version of WEA'08.

[27] Ariel Orda and Raphael Rom. Shortest-path and minimum delay algorithms in networks with time-dependent edge-length. *Journal of the ACM*, 37(3):607–625, 1990.

[28] Mihai Patrascu and Liam Roditty. Distance oracles beyond the Thorup–Zwick bound. In *Proc. of 51th IEEE Symp. on Found. of Comp. Sci. (FOCS '10)*, pages 815–823, 2010.

[29] Ely Porat and Liam Roditty. Preprocess, set, query! In *Proc. of 19th Eur. Symp. on Alg. (ESA '11)*, LNCS 6942, pages 603–614. Springer, 2011.

[30] Hanif D. Sherali, Kaan Ozbay, and Sairam Subramanian. The time-dependent shortest pair of disjoint paths problem: Complexity, models, and algorithms. *Networks*, 31(4):259–272, 1998.

[31] Christian Sommer. Shortest-path queries in static networks. *ACM Computing Surveys*, 46, 2014.

[32] Christian Sommer, Elad Verbin, and Wei Yu. Distance oracles for sparse graphs. In *Proc. of 50th IEEE Symp. on Found. of Comp. Sci. (FOCS '09)*, pages 703–712, 2009.

[33] Mikkel Thorup and Uri Zwick. Approximate distance oracles. *J. of ACM*, 52:1–24, 2005.

[34] C. Wulff-Nilsen. Approximate distance oracles with improved preprocessing time. In *Proc. of 23rd ACM-SIAM Symp. on Discr. Alg. (SODA '12)*, 2012.

[35] C. Wulff-Nilsen. Approximate distance oracles with improved query time. arXiv abs/1202.2336., 2012.

[36] Gernot Veit Batz, Daniel Delling, Peter Sanders, and Christian Vetter. Time-dependent contraction hierarchies. In *Proceedings of the 11th Workshop on Algorithm Engineering and Experiments (ALENEX'09)*, pages 97–105. SIAM, April 2009.

[37] Moritz Baum, Julian Dibbelt, Thomas Pajor, and Dorothea Wagner. Energy-optimal routes for electric vehicles. In *Proceedings of the 21st ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, pages 54–63. ACM Press, 2013.

[38] Hans L. Bodlaender and Arie M.C.A. Koster. Treewidth computations i. upper bounds. *Information and Computation*, 208(3):259–275, 2010.

[39] Soma Chaudhuri and Christos Zaroliagis. Shortest paths in digraphs of small treewidth. part i: Sequential algorithms. *Algorithmica*, 2000.

[40] 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.

[41] Daniel Delling, Andrew V. Goldberg, Thomas Pajor, and Renato F. Werneck. Customizable route planning in road networks. *Transportation Science*, 2014. accepted for publication.

[42] 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.

[43] Camil Demetrescu, Andrew V. Goldberg, and David S. Johnson, editors. *The Shortest Path Problem: Ninth DIMACS Implementation Challenge*, volume 74 of *DIMACS Book*. American Mathematical Society, 2009.

[44] Julian Dibbelt, Ben Strasser, and Dorothea Wagner. Customizable contraction hierarchies. In *Proceedings of the 13th International Symposium on Experimental Algorithms (SEA'14)*, volume 8504 of *Lecture Notes in Computer Science*, pages 271–282. Springer, 2014.

[45] 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.

[46] Giacomo Nannicini, Daniel Delling, Leo Liberti, and Dominik Schultes. Bidirectional A* search on time-dependent road networks. *Networks*, 59:240–251, 2012. Best Paper Award.

[47] Ariel Orda and Raphael Rom. Shortest-path and minimum delay algorithms in networks with time-dependent edge-length. *Journal of the ACM*, 37(3):607–625, 1990.

[48] Léon Planken, Mathijs de Weerdt, and Roman van Krogt. Computing all-pairs shortest paths by leveraging low treewidth. *Journal of Artificial Intelligence Research*, 2012.

[49] Sabine Storandt. *Algorithms for vehicle navigation*. PhD thesis, Universität Stuttgart, February 2013.

[50] Tim Zeitz. Weak contraction hierarchies work! Bachelor thesis, Karlsruhe Institute of Technology, 2013.

[51] Joachim M. Buhmann, Matúš Mihalák, Rastislav Srámek, and Peter Widmayer. Robust optimization in the presence of uncertainty. In *ITCS*, pages 505–514, 2013.

[52] eCOMPASS. D2.3 – 2 validation and empirical assessment of algorithms for eco-friendly vehicle routing. Technical report, The eCOMPASS Consortium, 2013.

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

[54] Ernesto Queirós Vieira Martins. On a multicriteria shortest path problem. *European Journal of Operational Research*, 16(2):236 – 245, 1984.

[55] Matús Mihalák, Sandro Montanari, and Peter Widmayer. Bidirectional algorithms for bicriteria quickest paths in road networks. Technical report, The eCOMPASS Consortium, 2014.

[56] Motor-talk.de. G6 90 kw tsi verbrauchskurven. Accessed on 29.10.2014.

[57] Martin Schmied and Wolfram Knörr. Calculating GHG emissions for freight forwarding and logistics services. European Association for Forwarding, Transport, Logistics and Customs Services (CLECAT), 2012.

[58] Wikipedia. Brutus cluster. Accessed on 23.10.2014.

[59] Openstreetmap. http://www.openstreetmap.org.

[60] Tomtom. http://www.tomtom.com.

[61] Camvit: Choice routing, 2009. http://www.camvit.com.

[62] eCOMPASS project, 2011-2014. http://www.ecompass-project.eu.

[63] Roland Bader, Jonathan Dees, Robert Geisberger, and Peter Sanders. Alternative route graphs in road networks. In *Theory and Practice of Algorithms in (Computer) Systems*, pages 21–32. Springer, 2011.

[64] Yanyan Chen, Michael GH Bell, and Klaus Bogenberger. Reliable pretrip multipath planning and dynamic adaptation for a centralized road navigation system. *Intelligent Transportation Systems, IEEE Transactions on*, 8(1):14–20, 2007.

[65] Daniel Delling and Moritz Kobitzsch. Personal commnication, July 2013.

[66] Andrew V Goldberg and Chris Harrelson. Computing the shortest path: A* search meets graph theory. In *Proc. 16th ACM-SIAM symposium on Discrete algorithms*, pages 156–165. Society for Industrial and Applied Mathematics, 2005.

[67] Felix Koenig. Future challenges in real-life routing. In *Workshop on New Prospects in Car Navigation*. February 2012. TU Berlin.

[68] Georgia Mali, Panagiotis Michail, Andreas Paraskevopoulos, and Christos Zaroliagis. A new dynamic graph structure for large-scale transportation networks. In *Algorithms and Complexity*, volume 7878 of *LNCS*, pages 312–323. Springer, 2013.

[69] Andreas Paraskevopoulos and Christos Zaroliagis. Improved alternative route planning. In *13th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems*, volume 33 of *OASICS*, pages 108–122, 2013. Also eCOMPASS Project, Technical Report TR-024, July 2013.