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: 36 months

D2.1 – New Prospects in Eco-friendly Vehicle Routing

Workpackage:	WP2 - Algorithms for Vehicle Routing
Due date of deliverable:	30 April 2012
Actual submission date:	30 April 2012
Responsible Partner:	CTI
Contributing Partners:	CTI, CERTH, ETHZ, KIT, TomTom, PTV

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: Route planning for private cars and fleets of vehicles, eco-aware quality measures, data uncertainty, robustness of routes, alternative routes, multi-criteria optimization, combinatorial optimization, heuristics.

The eCOMPASS project (www.ecompass-project.eu) is funded by the European Commission, Information Society and Media Directorate General, Unit G4-ICT for Transport, 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

Karlsruher Institut für Technologie (KIT), Germany

TOMTOM INTERNATIONAL BV (TOMTOM), Netherlands

PTV PLANUNG TRANSPORT VERKEHR AG. (PTV), Germany

Document history			
Version	Date	Status	Modifications made by
1.0	26.04.2012	First Draft	Spyros Kontogiannis, CTI
1.0	26.04.2012	Sent to internal reviewers	Spyros Kontogiannis, CTI
1.1	27.04.2012	Sent to PQB	Spyros Kontogiannis, CTI
1.2	28.04.2012	Reviewers comments incorporated (resent to PQB)	Spyros Kontogiannis, CTI
1.4	30.04.2012	PQBs comments incorporated	Spyros Kontogiannis, CTI
1.4	30.04.2012	Final (approved by PQB, sent to the Project Officer)	Spyros Kontogiannis, CTI Christos Zaroliagis, CTI

Deliverable manager

- Spyros Kontogiannis, CTI

List of Contributors

- Julian Dibbelt, KIT
- Georgios Grekas, CERTH
- Dimitris Gkortsilas, CTI
- Dionisis Kehagias, CERTH
- Felix König, TomTom
- Spyros Kontogiannis, CTI
- Georgia Mali, CTI
- Panagiotis Michail, CTI
- Sandro Montanari, ETHZ
- Sebastian Polzer, PTV
- Christos Zaroliagis, CTI

List of Evaluators

- Thomas Pajor, KIT
- Florian Krietsch, PTV

Summary
The purpose of the present deliverable is to present recent advances, their possible limitations, and new prospects for algorithmic techniques and methodologies concerning the problems related to single or multiple vehicle routing in urban areas. The focus is on *state-of-art approaches* that take into account the environmental impact, accommodate uncertainty of traffic data and their inherently temporal nature, and look for robust solutions, possibly with meaningful alternatives. We also investigate how to express the trade-offs between data precision, information content, and solution robustness, in order to be used in real time for meaningful and conscious online decisions based on updated traffic information.

Contents

1 Introduction

The present deliverable presents recent advances, their possible limitations, and new prospects for algorithmic techniques and methodologies concerning the problems related to WP2.

WP2 aims at providing novel algorithmic methods for optimizing vehicle routes in urban spaces with respect to their environmental impact. In particular, it aims at deriving novel algorithmic solutions to be applied on: (a) intelligent on-board navigator systems for private vehicles that take into account real traffic data (as well as statistical data about traffic at different hours during the day) and seamlessly provide "green" route recommendations; (b) eco-aware logistics and fleet management systems used in conjunction with on-board systems mounted on vehicles and used by drivers, aiming at minimal environmental footprint and fuel consumption.

The problems to be studied, concern the movement of private vehicles along with the movement of heavy vehicles engaged to human or goods transportation is the most common cause for heavy traffic conditions in cities. In particular, the route planning of multiple vehicles is important in logistics since the transportation of goods and the ensuing cost is a deciding factor for the success of any supply chain management system. Within this WP, we consider two problem classes. The first one concerns individual *private vehicles*, while the second one concerns *fleets of vehicles*.

The route planning of a single as well as multiple vehicles under various constraints is a central issue in any solution proposing smart urban mobility schemes. WP2 proposes route planning methods able to respond to dynamic events fast by adjusting vehicle routes on the fly. The real time information may be fed by the city traffic management system or vehicle drivers may notify the system through wireless communication about unexpected events happening on their way. All solutions proposed in WP2 will focus on minimizing the environmental footprint of vehicles by taking also into account traffic prediction, data reliability, route robustness, driving style, and load balancing issues. This focus differentiates this service from existing car navigation systems. Fuel consumption and hence CO_2 releases depend on many parameters such as the speed, variance of speed, type and load of vehicle. Hence, it is evident that there exist difficult trade–offs among a number of conflicting cost factors and therefore finding an efficient solution w.r.t. most cost measures represents a challenging endeavor. WP2 breaks down into five tasks:

- Task 2.1 – New prospects in eco-friendly vehicle routing: Alternative approaches, their limitations and promising new directions.
- Task 2.2 – Eco-friendly private vehicle routing algorithms: Optimize a private vehicles route with respect to the environmental footprint of its movement by investigating models that explicitly account for it, and also include traffic prediction. Exact and approximate solutions will be sought, an also dynamic and robust scenarios will be considered. New methodological approaches will be pursued to trade data precision, information content, and solution robustness.
- Task 2.3 – Eco-friendly routing algorithms for fleets of vehicles: Optimize routes of multiple vehicles in application scenarios that involve delivery/collection of goods by transportation/courier companies aiming at minimizing the environmental footprint associated with the vehicles' movement. Several applications will be considered (people transportation, parcel deliveries, etc). Exact and approximate solutions will be sought, and also dynamic and robust scenarios will be considered. New methodological approaches will be pursued to trade data precision, information content and solution robustness.
- Task 2.4 – Validation and empirical assessment: Validate the effectiveness of the derived algorithmic solutions in Tasks 2.2 and 2.3, as well as their suitability for online mobile applications upon a variety of realistic scenarios.
- Task 2.5 – Final assessment of eco-friendly vehicle routing algorithms: Assess the success of 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.

Deliverable D2.1 of eCOMPASS is the outcome of Task 2.1. The purpose of D2.1 is to present recent advances, their possible limitations, and new prospects for algorithmic techniques and methodologies concerning problems related to unimodal route planning in urban areas. The focus is on *state-of-art approaches* that take into account the environmental impact, accommodate uncertainty of traffic data and their inherently temporal nature, and look for robust solutions, possibly with meaningful alternatives. We also investigate how to express the trade-offs between data precision, information content and solution robustness, in order to be used in real time for meaningful and conscious online decisions based on updated traffic information.

We consider two distinct scenarios of eco-aware urban route planning: The "private vehicles" scenario concerns mobility problems of individual, selfishly motivated drivers, equipped with intelligent, mainly *unidirectional* on-board navigation systems, that receive updated information by a centralized traffic monitoring system. Therefore, a major assumption is that these on-board navigation systems have to take into account real traffic data and/or predictions of future traffic, as they are provided in real-time by the navigation system provider. The second scenario concerns the mobility of "fleets of vehicles", whose objective is to choose route plans that optimize the fleet's (rather than individual vehicle's) criteria. Toward this direction, eco-aware logistics and fleet management systems will be investigated, exploiting the *bidirectional collaboration* of the more powerful on-board navigation systems with the master control of the fleet, as well as the real traffic data and/or predictions of future traffic, provided by the navigation system provider.

The main issues to be addressed in both scenarios are the consideration of *data uncertainty*, the provision of *robustness* and *multiple-alternatives*, as well as the consideration of *eco-friendliness* in the proposed solutions.

In the remaining part of the deliverable we investigate the *state-of-art* approaches concerning algorithmic problems related to route planning of private vehicles and fleets of vehicles (Section 2). Section 3 concludes the deliverable with a matching of candidate techniques and methodologies with the main axes for future research on route planning in urban areas, within WP2.

2 Eco-Aware Route Planing: State-of-Art

In this section our focus is on state-of-art concerning problems related to route planning for private cars and fleets of vehicles, which move selfishly, based on their own objectives. We begin (cf. Subsection 2.1) with an overview of the main challenges when dealing with uncertain traffic and/or infrastructure data. The goal is to provide either good-in-expectation, or robust (i.e., good-in-worst-case) route plans, depending on the model of uncertainty that we consider. Consequently, in Subsection 2.2 we present the main cost models for urban traffic, that take into account the environmental footprint of the routes. Subsection 2.3 presents an overview of well known, generic techniques and speed-up heuristics for route planning, in the case of time-independent road networks. In Subsection 2.4 proceeds one step ahead by considering state-of-art techniques for planning routes in time-dependent networks. Subsection 2.5 presents the literature related to alternative and/or robust route plans. Robustness at this point is meant to deal with drivers' occasionally wrong decisions, rather than uncertainty of raw data. Subsection 2.6 overviews existing industrial solutions for educating drivers how to adapt their driving style and choose their own route plans, towards more eco-friendly choices. Finally, Subsection 2.7 studies load-balancing problems related to route plans for fleets of vehicles.

2.1 Route Planning Under Uncertainty

Every day, millions of people take their private cars to drive from home to work. However, the capacities of the roads in the cities are limited. If too many cars try to go through a road at the same time, there is an increased chance of traffic jams, accidents, and other disturbances. It may not be possible to predict such situations just by looking at the road map of a city. It is a major challenge to design navigation systems that deal with disturbances and produce attractive routes.

Route planning is a fundamental task for Computer Science. It is broadly studied, and it involves many different fields, ranging from robotics, to transportation planning, to VLSI design. From an algorithmic viewpoint, route planning is usually translated to the shortest path problem. In the most typical formulation, also called the single-source shortest path problem, we are given a graph $G = (V, E)$ where on every edge $e \in E$, a cost $c_v \in \mathbb{R}$ is assigned, and we are asked to find a path with minimum cost between two vertices $s, t \in V$. If the edge costs are positive, as we will assume in this work, this problem can be solved efficiently using Dijkstra's algorithm [10].

We focus on the issues arising in "Route Planning for Private Vehicles". Therefore, the graphs we will consider typically represent road networks, that are extremely well-studied, and many heuristics and optimization techniques are known that exploit their peculiar structure in order to improve the efficiency of computing a shortest path. However, one of the main issues in road networks, is that the exact time to follow a road is usually determined by possibly unknown factors, that we call *uncertainty*. At the time of writing, there exists no universal or standard technique to deal with uncertainty. An example of uncertainty in a road network is given by the presence of traffic delays: in the rush hours of the day it may not be the best choice to follow the usually fastest route, since it is probably congested. We provide a survey of the state-of-the-art techniques to deal with uncertainty in route planning tasks. We aim to point out pros and cons of the known solutions, and inspect what is open to improvement. Since the shortest path problem is at the core of route planning, we focus on this particular problem.

Classic techniques to deal with uncertainty usually assume that the uncertain data is not arbitrary, but follows a known model. The nature of this model typically falls into one of the two categories: *robustness*, and *stochastic optimization*. With robustness, the probability distribution of the uncertain data is not known, we do not know "how likely" it is to find a road congested. Nevertheless, we wish to be prepared for every possible situation that may happen. We want to be sure that we will not have to pay too much, in terms of travel time, even if the worst situation occurs. We refer to robustness as "worst-case optimization".

Stochastic optimization techniques assume that we *do* know how likely it is for a road to be congested. We refer to stochastic optimization as "optimization of expectations". We assume that the probability distribution over the uncertain data is known, and we can rely on this knowledge when designing a solution. In this situation, the cost of a path becomes a random variable, and we have to find a good trade-off between the expected cost of the path we pick, and the risk of deviating from that cost.

Within these two categories, it is possible to provide a further classification. This classification is based on *when* the "uncertain" data becomes "certain", and we have to face how good our chosen path is with respect to an actual shortest one. Then, we could allow to change the chosen decision, if we notice that another road produces a better result. For example, if we are listening to the news on the radio while driving the car, and we hear that there is a traffic jam ahead of us, we may want to change our route in order to avoid the congestion. We will define a shortest path problem under uncertainty to be *adaptive* if the uncertain information is gained piece by piece, and we are allowed to change our path in order to react to the new information. If, instead, the uncertain data is given completely after the path has been chosen, and the path cannot be changed, we will denote the problem as *non-adaptive*.

The remainder of this paper is organized as follows: In Subsection 2.1.1 we describe the techniques that fall into the category of worst-case optimization. In particular, we first present the adaptive solutions, and then the non-adaptive ones. The same distinction is made in Subsection 2.1.2, where we present the techniques related to optimization of expectations. Subsection 2.1.3 contains a brief summary of all the results presented.

2.1.1 Worst-Case Optimization

In worst-case optimization techniques, we assume that the distribution of the values of the uncertain data can be described using a model. No probabilistic assumptions are made in this model, it just provides a (sub)set of all the possible situations that we may encounter. We do not know how likely it is for a "bad" situation to appear, but we wish our routes to be acceptable even in this case. To formally define when a solution is acceptable, we can use one of these two criteria: the *absolute* criterion (also called min-max), and the *maximum deviation* criterion (or min-max regret).

The absolute criterion defines an optimal path, among all the feasible paths, as one whose maximum cost among all possible realizations of the uncertain data is minimized. It represents a pessimistic view, where we expect the worst situation to be the one that we will actually have to face, and we want to be sure that we do not pay much in this case. The maximum deviation criterion asks for a path that is more sensitive to the actual realization that we will face. It defines an optimal path as one that minimizes the maximum distance (or ratio) from an actual optimal one in every realization. It is more sensitive because it tends to produce paths that have, in general, lower costs with respect to the absolute criterion, but not in the worst situation, where we might indeed pay much more. In adaptive shortest path problems, these two criteria are called respectively, *Max/Max ratio*, and *competitive ratio*, using terminology from competitive analysis.

Adaptive Shortest Paths under Uncertainty.

In the classical shortest path problem, we are asked to find a shortest path from a source to a destination in a graph. The topology of the graph is known, as well as all the costs on the edges. If the edge costs are non negative, the problem can be solved efficiently using Dijkstra's algorithm.

When shortest paths are computed under uncertainty, some essential knowledge about the graph is missing, or is not complete. *Adaptive shortest path problems* are a branch of shortest path problems under uncertainty where the missing information is acquired piece by piece only after the actual trip begins. A typical example for such a problem is given by a driver in Canada during the winter. He knows which is the best path between two cities, but some roads may be blocked by snow, and he discovers this information only when he reaches one of the endpoints.

This problem was first introduced by Papadimitriou in [101], where it was referred to as the `Canadian Traveller Problem` (`CTP`). The goal is the same as the classical shortest path problem: in a graph $G = (V, E)$ with edge costs $c_e \in \mathbb{R}^+$ for every $e \in E$, we want to find a shortest path between two vertices $s, t \in V$, i.e., a path with minimum cost. The difference is that not all the edges in E can be used, because some roads may be blocked. Only a subset $E_{\text{free}} \subset E$ can be traversed, but this set E_{free} is not given. Whether or not an edge $e \in E$ is in E_{free} (i.e., is actually usable), or it is in $E_{\text{blocked}} = (E - E_{\text{free}})$ (i.e., is blocked and cannot be used), is discovered only when one of the adjacent vertices of the edge is reached. In case an edge on the path turns out to be blocked, we have to change the chosen path. Note that the set E_{free} is fixed and cannot change. If a vertex $v \in V$ is reached again in a subsequent step, the list of blocked edges adjacent to it that the algorithm receives as input will be the same it received the first time v was visited.

Given the adaptive nature of the problem, we ask for an *online algorithm* to compute the solution. The input of this algorithm in each step is the list of edges in E_{blocked} adjacent to the current vertex, and its output is the next edge to be followed. To compute the output, the algorithm can only use the underlying graph $G = (V, E)$, that is given as input at the beginning, and the information that it has acquired so far.

The quality of the path produced from this algorithm is evaluated using competitive analysis. For example, we may measure the *competitive ratio,* i.e, the cost of the computed path divided by the cost of an actual shortest path, using only edges in E_{free}. For an introduction to online algorithms and competitive analysis, we refer the reader to [30, 11]. In [101], it is shown that the problem of providing a path for the `CTP` with bounded competitive ratio is PSPACE-complete.

Bar-Noy and Schieber [14] studied two variants of `CTP`: k-`CTP`, and `Recoverable-CTP`. In k-`CTP`, a bound k on the overall number of edges that can be blocked is given as input at the beginning, i.e., we know that $|E_{\text{blocked}}| \leq k$. Their goal is to devise an algorithm with minimum *Max/Max ratio* [26, 30], that is the maximum cost of the path produced for any possible sequence of blocked edges, divided by the maximum cost of a shortest path for any sequence of the same size. Like the original `CTP`, this variant is also PSPACE-complete.

`Recoverable-CTP` differs from the original problem in three aspects: the overall number of edges that can be blocked is bounded by a fixed parameter k, the algorithm may reject a list that it receives as input by paying an additional cost, and blocked roads in $G = (V, E)$ can be reopened. In the original `CTP`, the first time a vertex $v \in V$ is reached, the list B of blocked edges adjacent to it is given as input to the online algorithm. The list cannot change if v is reached again in a later step. In `Recoverable-CTP`, *every time* a vertex v is reached, a *new* list is given, independently from the previous inputs of the algorithm. The online algorithm can reject a list that it has received as input, and require a new one for the current vertex. The penalty for taking such an action at vertex $v \in V$ is to pay a "waiting cost" $c_v \in \mathbb{R}^+$ in addition to the overall cost of the path, with the requirement that $c_v < c_e$ for every edge e adjacent to v. As a further assumption, the overall number of edges that can be blocked is bounded by a fixed parameter k. After k blocked edges have been discovered, it is assured that every list is empty for any $v \in V$, therefore the online algorithm can safely follow a shortest path in G from the current vertex to the destination. Note that even if a list is rejected, the number of blocked edges in it still counts towards k.

For `Recoverable-CTP`, Bar-Noy and Schieber [14] propose an efficient online algorithm. To model the rejection of a list, and the corresponding cost penalty, a self-loop edge that cannot be blocked with cost c_v is added to every vertex. When the algorithm decides to reject the current list B, it outputs the self-loop instead of a regular edge. The algorithm keeps a counter h that starts from k and decreases every time a blocked edge is encountered. To decide the best action to take at vertex v, two possible cases are distinguished, depending on the value of h:

- in case $h = 0$, no more edges can be blocked, B is empty for every $v \in V$. The output of the algorithm is the first edge of a shortest path from v to t in G;
- in case $h > 0$, more blocked edges can be encountered. Let B of size $0 \leq |B| \leq h$, be the input of the algorithm when reaching vertex v. Let $dist(i, v)$ be the worst case travel distance

from v to t, given that at most i edges can be blocked. It turns out [14] that this value can be computed efficiently. To get the next edge, the algorithm sorts the list L_v of edges $e_1, \ldots, e_d$ adjacent to v according to their values of $dist(h - |B|, y_j)$, where y_j is the vertex at the other end of e_j, for $j = 1, \ldots, d$. The output in this case is the first edge in L_v that is not in B.

The assumptions that the edges are recoverable and that the overall number of blocked edges is bounded, allow for efficient computation. In [14] it is shown that this algorithm has minimum Max/Max ratio among all possible online algorithms for this setting. Furthermore, it is shown that for a given j, the values $dist(j, v)$ and the corresponding L_v for every $v \in V$ can be computed in time $\mathcal{O}(jm + n \log n)$, if the previous values and lists for $i = 0, \ldots, j-1$ are known. Since we have to compute $dist(k, v)$ for every $v \in V$ and every $j = 0, \ldots, k$, the overall time complexity is $\mathcal{O}(k^2 m + kn \log n)$. Note that if k is no longer a fixed parameter but part of the input, this algorithm becomes pseudo-polynomial in k.

The assumptions made in `Recoverable-CTP` alter the nature of the original `CTP` significantly. Even though these assumptions allow an efficient computation of a solution, they might seem to be artificial. In particular, the assumption that the number of blocked edges in a list that has been rejected still counts towards the overall count of blocked edges is quite strong. With this assumption, a naive algorithm that follows the shortest path from the source to the destination, and always rejects the current list if the required edge is blocked could still perform quite well in many cases, even if the path that it produces is not optimum. We may wish to disallow this situation to happen, either by removing the bound k, or by imposing that the blocked edges in a rejected list do not count towards the overall count (but with the additional requirement that at every step at least one edge is usable). In this setting it may be worth accepting a list that otherwise would have been rejected just because it contains a large number of blocked edges, even though it may have us move away from the destination. Another feature that we might require is to allow blocked edges to become usable again after a fixed period. In the original version of `CTP`, whenever the status of an edge (whether it is usable or blocked) is discovered, it cannot change for the rest of the computation. On the other hand, in `Recoverable-CTP` every time a new vertex is reached, the status of every edge in the graph is reset. We can think of these two situations as extremes: a graph with infinite memory and a memory-less graph. In an intermediate situation, we could assume that the graph has limited memory, and that the status of an edge can change again only after a while, for example after a fixed number of steps of the online algorithm. It is unknown how efficiently we can compute solutions for these problems.

In [101], a problem called `Double Valued Graph Problem` is presented. We are given a *double-valued graph* $G = (V, E)$, where every edge $e \in E$ has two possible costs, $c_e^{(1)}$ and $c_e^{(2)}$. The actual cost is one of the two, and this information is discovered only when one of the endpoints of e is reached. The double-valued graph can also be used to study `CTP`, since we can model the absence of an edge with a suitably large edge cost. The authors show that providing a path from vertex s to vertex t with bounded competitive ratio in this setting is PSPACE-complete.

Non-adaptive Shortest Path under Uncertainty.

All the worst-case optimization techniques presented so far produce adaptive solutions. It is natural to ask for variants that allow us to devise non-adaptive solutions. For *non-adaptive shortest path problems under uncertainty* we assume that the initial knowledge about the underlying graph is not complete and the path we would pick at first might not be an actual shortest one. The difference is how and when the unknown information is obtained. In adaptive shortest path problems, missing knowledge is gained piece by piece in an online fashion, after reaching one of the endpoints of an edge. The online algorithm must therefore be able to react to this new knowledge by adapting the path to the destination. In non-adaptive problems, the whole missing information is given after the path to follow has been chosen, but the path cannot be changed. We require the chosen path has to be "good" (in a sense that will be specified later) with respect to every possible *realization* of

the missing information. Such a path is called a *robust path*. Without loss of generality, in every non-adaptive problem we assume only the cost of the edges in the graph to be uncertain, and not its topology. It is possible to emulate the absence of an edge in the graph by assigning a suitable large cost to it.

The uncertain information about the graph can be modeled in many different ways. Roughly, each model has a different way to describe the set of all possible realizations of the edge costs, which affects directly the feasibility of the computation of a robust shortest path.

In [128], two criteria are introduced to formally define a robust path:

- the *absolute* criterion aims to minimize the maximum path cost over all possible realizations of the graph. A path that optimizes this criterion is called ABSOLUTE ROBUST SHORTEST PATH;
- the *maximum deviation* criterion aims to minimize the maximum distance of the produced path cost from the optimal path cost over all the possible realizations. A path that optimizes this criterion is called a ROBUST DEVIATION SHORTEST PATH.

Intuitively, the absolute criterion represents a more pessimistic choice, where we expect the worst case to happen, and we wish to minimize the cost of the solution in this situation. On the other hand, the robust deviation criterion represents a choice where we wish to be more sensitive to the variation of the data, even though this may lead us to pay more in the worst case. For example, we may consider a trip from Zurich to Pisa on two different days, where the shortest routes take respectively five and seven hours. With the former criterion we prefer a route that takes nine hours in both days, while the latter criterion may lead us to a route that takes six hours in the first day, but ten hours in the second one, i.e., eight hours in average.

In order to make the notation easier to understand, in the rest of the section we will assume $V = \{1, \dots, n\}$, and an edge $e \in E$ with cost c_e between vertices $i, j \in V$ will also be denoted as (i, j), and its cost c_{ij}. Using this notation, we can formulate the classical shortest path problem using linear programming. Given two vertices $s, v \in V$, the following linear program yields a vector $\boldsymbol{x} = (x_{ij})$ that represents a path with minimum cost between s and t in G, if the edge costs are strictly positive and such a path exists.

$$\begin{aligned}
\text{Minimize} \quad & \sum_{(i,j)\in E} x_{ij} \cdot c_{ij} \\
\text{subject to} \quad & \forall i \in V : \sum_{j\in V} x_{ij} - \sum_{k\in V} x_{ki} = \begin{cases} 1 & \text{if } i = s \\ -1 & \text{if } i = t \\ 0 & \text{otherwise} \end{cases} \qquad (1) \\
& \forall i, j \in V : x_{ij} \in \{0, 1\} \qquad (2)
\end{aligned}$$

The indicator variable x_{ij} in the resulting vector is 1 if the corresponding edge (i, j) is part of the solution, otherwise is 0. The presence of constraint (1) requires any feasible solution to be a path from the source to the destination, although in this formulation it may also contain a cycle. Since we are minimizing the objective function, and the edge costs are strictly positive, every optimum solution is a simple path from s to t, i.e., it does not contain a cycle. We will modify this compact formulation of the shortest path problem to illustrate exactly how the models presented differ from each other, and from the classical shortest path problem.

Scenario Based Description. A first model for non-adaptive problems under uncertainty was introduced in [128], following a natural approach to describe all the possible realizations of the graph. Here the realizations are given explicitly, as a finite number N of predetermined sets of edge costs, also called *scenarios*. We are given a graph $G = (V, E)$, and costs $c_{ij}^{(1)}, \dots, c_{ij}^{(N)} \in \mathbb{R}^+$ for

every edge $(i,j) \in E$. A scenario is specified by a value $r \in S = \{1, \ldots, N\}$, and in scenario r the cost of an edge $(i,j) \in E$ is $c_{ij}^{(r)}$.

We can extend the notation of absolute robust and the robust deviation shortest paths to this setting using the linear programming formulation of the classical shortest path problem. An absolute robust shortest path from s to t is a path that minimizes its maximum cost over all scenarios. By changing the objective function in the previous linear program, we can express the absolute robust shortest path in this context. Note, however, that by changing the objective function to a non-linear one, we cannot use classical algorithms for shortest paths. It turns out that this problem is NP-hard. An absolute robust shortest path is given by

$$\begin{aligned} \arg\min_{\boldsymbol{x}} \quad & \left\{ \max_{r \in S} \sum_{(i,j) \in E} x_{ij} \cdot c_{ij}^{(r)} \right\} \\ \text{subject to} \quad & (1) \text{ and } (2). \end{aligned}$$

A robust deviation shortest path is a path from s to t that minimizes the maximum distance of its cost from the optimum path cost over all possible scenarios in S. That is,

$$\begin{aligned} \arg\min_{\boldsymbol{x}} \quad & \left\{ \max_{r \in S} \sum_{(i,j) \in E} x_{ij} \cdot c_{ij}^{(r)} - sp_r \right\} \\ \text{subject to} \quad & (1) \text{ and } (2), \end{aligned}$$

where sp_r is the cost of a shortest path in scenario r.

In [128], the authors consider a special class of graphs called *layered graphs*. In a k-layered graph $G = (V, E)$, the vertex set can be partitioned into disjoint sets $V = V_0 \cup V_1 \cup \ldots V_k \cup V_{k+1}$, such that $V_0 = \{s\}$ and $V_{k+1} = \{t\}$, and edges exist only between sets V_l and V_{l+1}, for $l = 0, \ldots, k$. The authors show that both the problems of computing an absolute robust shortest path and a robust deviation shortest path are NP-complete, even for 2-layered graphs and only 2 scenarios. They also present an algorithm that computes the absolute shortest path and the robust deviation shortest path in general graphs with N scenarios with running time $\mathcal{O}(n^3 \cdot (lp_{\max})^N)$, where $lp_{\max}$ is the cost of the longest path from s to t among all scenarios. If N is bounded by a constant, the algorithm is pseudo-polynomial. In contrast, both problems are proven to be strongly NP-hard if N is not bounded.

Edge costs from Interval Data. A second model for non-adaptive problems under uncertainty is introduced in [127]. Here, the uncertainty about the edge costs is expressed in form of a lower and an upper bound on the possible costs of each edge. Given a graph $G = (V, E)$, every edge $e \in E$ has two values l_e and u_e associated with it, with $0 < l_e \leq u_e$. The cost of edge e in a realization r is $c_e^{(r)} \in [l_e, u_e]$.

This model allows for efficient computation of an absolute robust shortest path, since we can just consider the realization where the cost of each edge $e \in E$ is set to the upper bound of the interval $[l_e, u_e]$. It is easy to see that the maximum cost of every path is always found in this unique realization.

In [127], the authors derive a mixed integer programming formulation for the computation of a robust deviation shortest path, but give no complexity results. In [129], it is shown that computing a robust deviation shortest path in this setting is NP-hard, and it remains NP-hard even when the graph is restricted to be directed acyclic planar with maximum degree three.

In [89], a generalization of the problem is studied, introducing a way to model relations between edge costs. The edge costs are now drawn from a convex polytope $\mathcal{F} = \{\boldsymbol{y} \in \mathbb{R}^m \mid \boldsymbol{M}\boldsymbol{y} \leq \boldsymbol{b}\}$, described by a matrix $\boldsymbol{M} \in \mathbb{R}^{m \times m}$ and a vector $\boldsymbol{b} \in \mathbb{R}^m$. A realization is a point in this polytope, and the edge costs for a realization are represented by a vector $\bar{\boldsymbol{c}} = (\bar{c_{ij}}) \in \mathcal{F}$.

In this setting, an absolute robust shortest path is given by

$$\begin{array}{ll} \arg\min\limits_{\boldsymbol{x}} & \left\{ \max\limits_{\bar{\boldsymbol{c}} \in \mathcal{F}} \sum\limits_{(i,j)\in E} x_{ij} \cdot \bar{c_{ij}} \right\} \\ \text{subject to} & (1) \text{ and } (2). \end{array}$$

If we denote with $sp_{\bar{\boldsymbol{c}}}$ the cost of a shortest path when the costs in the graph are $\bar{\boldsymbol{c}}$, a robust deviation shortest path is given by

$$\begin{array}{ll} \arg\min\limits_{\boldsymbol{x}} & \left\{ \max\limits_{\bar{\boldsymbol{c}} \in \mathcal{F}} \sum\limits_{(i,j)\in E} x_{ij} \cdot \bar{c_{ij}} - sp_{\bar{\boldsymbol{c}}} \right\} \\ \text{subject to} & (1) \text{ and } (2). \end{array}$$

In [89], the authors show that both the problems of computing an absolute robust and a robust deviation shortest paths are NP-hard.

Robust Discrete Optimization. In [118], a model for general optimization problems with uncertainty is given. The authors consider many kinds of optimization problems, including linear, quadratic, discrete, and combinatorial optimization problems. They aim to provide a coherent framework, i.e., the idea behind every approach is the same, for formulating robust versions of problems of these kinds that are practically tractable.

On a general level, this framework takes an optimization problem as input and returns a robust formulation of the same problem that allows uncertainty on either the objective function or the constraints. In [29], it is shown how to apply this framework to *combinatorial optimization problems*, to which the shortest path problem belongs.

Formally, a combinatorial optimization problem is given by m indicator variables $x_1, \ldots, x_m$, an objective function $f(\boldsymbol{x}) = \sum_{i=1,\ldots,m} x_i \cdot c_i$, and a set $\mathcal{X} \subseteq \{0,1\}^m$ that constrains the values that the decision variables can assume. Our objective is to find a solution $\boldsymbol{x} \in \mathcal{X}$ that either minimizes or maximizes the objective function. This is usually expressed as (for minimization)

$$\begin{array}{ll} \text{Minimize} & \boldsymbol{c'x} \\ \text{subject to} & \boldsymbol{x} \in \mathcal{X}. \end{array}$$

The linear programming formulation of the shortest path problem shown before is a typical example of a combinatorial optimization problem expressed using this formulation. In this framework, we assume that the coefficients that are part of the objective function of the combinatorial optimization problem are uncertain, and could deviate from the nominal value c_i to any value within an interval $[c_i, c_i + d_i]$ for a given $d_i \geq 0$, and every $i = 1, \ldots, m$. Without loss of generality, we assume that the indices are ordered such that $d_1 \geq \cdots \geq d_m$, and we define $d_{m+1} = 0$. The new objective in the robust formulation of this problem is to obtain a solution that minimizes the maximum cost $\boldsymbol{c'x}$, when at most $\Gamma \geq 0$ coefficients are allowed to deviate. This solution, denoted with Z^*, is given by

$$\begin{array}{lll} Z^* = & \arg\min\limits_{\boldsymbol{x}} & \left\{ \boldsymbol{c'x} + \max\limits_{S\in\Phi} \sum\limits_{i\in S} x_i \cdot d_i \right\} \\ & \text{subject to} & \boldsymbol{x} \in \mathcal{X}, \end{array}$$

where $\Phi = \{S | \, S \subseteq \{1, \ldots, m\}, |S| \leq \Gamma\}$.

As previously stated, the focus of this framework is to give robust versions of optimization problems that are practically tractable. In [29] it is shown that, for combinatorial optimization problems, it is possible to compute the solution Z^* by solving $m+1$ simpler problems, one for each of the values $d_1, \ldots, d_{m+1}$. In particular, it is shown that Z^* is the solution with minimum cost among $Z^1, \ldots, Z^{m+1}$, where Z^l is given by:

$$Z^l = \begin{array}{ll} \arg\min\limits_{\boldsymbol{x}} & \left\{ \boldsymbol{c'x} + \Gamma d_l + \sum\limits_{j=1}^{l} x_j \cdot (d_j - d_l) \right\} \\ \text{subject to} & \boldsymbol{x} \in \mathcal{X} \end{array}$$

for $l = 1, \ldots, m+1$.

It is not hard to see that every objective function of the additional problems is linear, therefore Z^l is the solution of a standard combinatorial optimization problem of the same kind as the original one, where the coefficients of the objective function are a perturbation of those in $\boldsymbol{c}$. This implies that, if the original problem is NP-hard, then its robust counterpart is also NP-hard. On the other hand, if the original problem is polynomially solvable, so is its robust counterpart. Furthermore, it can be shown that, if $d_l = d_{l+1}$, then $Z^l = Z^{l+1}$. Therefore, if f is the number of distinct values among $d_1, \ldots, d_m$, only $f+1$ additional instances of the original problem need to be solved.

It is easy to see that the solution Z^* belongs to the category of absolute robust shortest paths. It remains an open question whether a formulation of a robust deviation shortest path can be given in this framework and, more importantly, whether the same complexity results hold for the computation of a robust deviation shortest path.

Recoverable Robustness. *Recoverable robustness* merges adaptive and non-adaptive ideas in order to benefit from both approaches. The model used in recoverable robustness works in two stages: in the first stage, we are given a graph where the edge costs represent a prediction of the situation that we will have to face. Once a path for the first stage is chosen, the edge costs are changed, and we are allowed to modify some parts of the chosen path in order to adapt to these new costs. The concept of recoverable robustness was introduced in [81].

Two possible applications of recoverable robustness to shortest path problems are presented in [31]. In k-`distRR` we are given a graph $G = (V, E)$, a source and a destination vertex $s, t \in V$, and a first-stage cost $c_e^{(1)} \in \mathbb{R}^+$ for every edge $e \in E$. In order to simplify the notation, we denote the set of all feasible paths with $\mathcal{X}$, and the first-stage cost of a path $\boldsymbol{x} \in \mathcal{X}$ as $c^{(1)}(\boldsymbol{x}) = \sum_{e \in \boldsymbol{x}} c_e^{(1)}$. Once a path $\boldsymbol{x}$ between s and t for the first-stage is chosen, the second-stage edge costs $c_e^{(2)}$ are revealed from a set $\mathcal{F} \subset \mathbb{R}_+^m$, and we are allowed to change the chosen path. However, we cannot choose any feasible path $\boldsymbol{x'}$ as the second-stage one, we require that $\boldsymbol{x}$ and $\boldsymbol{x'}$ differ by *at most* k edges. We denote this constraint as $d(\boldsymbol{x}, \boldsymbol{x'}) \leq k$. If the set of all possible second-stage edge costs $\mathcal{F}$ is known in advance we can require, following the absolute robust shortest path criterion, that the two chosen paths minimize their cost over all possible realizations of second-stage edge costs. That is, we look for the paths given by

$$\begin{array}{ll} \arg\min\limits_{\boldsymbol{x}, \boldsymbol{x'}} & \left\{ c^{(1)}(\boldsymbol{x}) + \max\limits_{c^{(2)} \in \mathcal{F}} c^{(2)}(\boldsymbol{x'}) \right\} \\ \text{subject to} & \boldsymbol{x}, \boldsymbol{x'} \in \mathcal{X} \\ & d(\boldsymbol{x}, \boldsymbol{x'}) \leq k. \end{array}$$

The complexity of this problem is determined by the value k and the description of the second-stage edge costs (the set $\mathcal{F}$). In [31], the following three options for $\mathcal{F}$ are considered:

1. $\mathcal{F}$ is a discrete finite set (like in the scenario based description),
2. the edge costs are taken from given intervals $[l_e, u_e]$ for every $e \in E$,
3. an interval $[l_e, u_e]$ is given for every $e \in E$, but at most a constant number of edges can take any value in the interval, the others have their cost set to l_e.

In all these three cases, the k-`distRR` problem is proven to be NP-hard , even if k is not part of the input, but a fixed parameter.

The second recoverable robust formulation of the shortest path problem is called `Rent-RR`. Here, the first-stage costs represent a *rent* and, in the second-stage, *any* feasible path can be chosen. If we do reuse an edge that was bought in the first-stage we do not have to pay its cost, but for every new edge we have to pay a penalty. Formally, the first-stage cost of a path for edge costs $c \in \mathcal{F} \subset \mathbb{R}^m_+$ is defined as $c^{(1)}(\boldsymbol{x}) = \alpha \sum_{e \in \boldsymbol{x}} c_e$, for a given $\alpha \in (0,1)$. In the second-stage, we choose a path $\boldsymbol{x}'$, which does not pay anything for shared edges, but pays a penalty for the new edges. Formally, $c^{(2)}(\boldsymbol{x}') = \sum_{e \in \boldsymbol{x}'} c_e + \beta \sum_{e \in \boldsymbol{x}' \setminus \boldsymbol{x}} c_e - \alpha \sum_{e \in \boldsymbol{x} \cap \boldsymbol{x}'} c_e$, for a given $\beta \geq 0$. Note that the requirement that $\alpha \in (0,1)$ is necessary to make the problem non trivial. If $\alpha \in \{0,1\}$, it is easy to see that the first-stage and the second-stage path are the same. Following the absolute robust shortest path criterion, we ask for the two paths that minimize the sum of the first-stage and second-stage cost in every possible realization. That is, we look for the path given by

$$\begin{array}{ll} \arg\min\limits_{\boldsymbol{x},\boldsymbol{x}'} \max\limits_{c \in \mathcal{F}} & \left\{ c^{(1)}(\boldsymbol{x}) + c^{(2)}(\boldsymbol{x}') \right\} \\ \text{subject to} & \boldsymbol{x}, \boldsymbol{x}' \in \mathcal{X}. \end{array}$$

It is easy to see that, if $|\mathcal{F}| = 1$, then $\boldsymbol{x} = \boldsymbol{x}'$, and the problem can be efficiently solved. In [31], this problem is proven to be NP-hard for the three same cases presented above.

It is possible to give a different formulation of `Rent-RR`, that does not use the parameters α and β. We could assume that the first-stage costs are not subject to uncertainty. The cost of a path $\boldsymbol{x}$ at the first-stage is then given by $c^{(1)}(\boldsymbol{x}) = \sum_{e \in \boldsymbol{x}} c_e^{(1)}$ (without the parameter α), and the second-stage (uncertain) cost of $\boldsymbol{x}'$ in a realization $c^{(2)} \in \mathcal{F}$ is the cost of the newly added edges, i.e., $c^{(2)}(\boldsymbol{x}') = \sum_{e \in \boldsymbol{x}'} c_e^{(2)} - \sum_{e \in \boldsymbol{x} \cap \boldsymbol{x}'} c_e^{(2)}$. The goal is to find the two paths that minimize the sum of the first-stage and the second-stage costs. It is not known whether this problem can be solved efficiently. Note that, we could allow $\boldsymbol{x}$ not to be a feasible path, since a feasible path is chosen anyway in the second-stage, and at the first-stage we may be only interested in buying those edges that seem to be promising.

2.1.2 Optimization of Expectations

In the previous subsection, we looked at worst-case optimization of routes under uncertainty. We considered situations where a model describing all the possible realizations of the uncertain data was known, and we looked for solutions that were acceptable in the worst case. This model was our only knowledge about the uncertain data. In some situations, we might expect that something more about the uncertain data is known, like a probabilistic distribution of the values that it may assume. For example, we could assume that the edge costs are uncertain, and the probability for each edge cost to appear is known. Here, it may not be interesting to consider the worst case, if it appears with low probability. Instead, we are interested in *optimization of expectations*, for example minimize the expected cost of a path. We will in general refer to this kind of shortest path problems as *stochastic shortest path problems*. As before, we distinguish between non-adaptive and adaptive variants.

Adaptive Stochastic Shortest Path

In adaptive shortest path problems, it is possible to gain additional information about the uncertain data piece by piece after the trip is started. The chosen path can be changed in order to react to this newly discovered information. For example, in the `Canadian Traveller Problem`, we discover whether an edge is usable, or it is blocked, when one of its endpoints is reached. In stochastic adaptive shortest path problems, we are given, as additional information, a probability for each of these situations to appear. We can rely on these probabilities when computing a path from the source to the destination. The stochastic version of the `CTP` was introduced in [28].

In [101], we are given a graph $G = (V, E)$, where the cost of each edge is given by a discrete random variable with known probability distribution, a source vertex s and a destination t. When a vertex is reached, the actual cost of every edge incident to it is discovered. We aim to find an online algorithm that, in each step, is given as input the real costs of the edges adjacent to the current vertex, and outputs the next edge to follow along the path from s to t. Once the real cost of an edge is discovered, it cannot change again. Since we are working in a stochastic environment, our goal is to minimize the *expected competitive ratio* of the provided algorithm, i.e., the expected cost of the path produced by the algorithm on any possible realization of edge costs, over the cost of an actual shortest path for the same realization. The problem of providing a path with minimum expected competitive ratio for this setting was introduced in [101], where it was proven to be #P-hard. Since giving an online algorithm with minimum expected competitive ratio is too hard, we may decide to settle for a path with minimum *expected cost* (not ratio). It is shown in [101, 106] that also this problem is #P-hard.

In worst-case optimization, we can usually emulate the absence of an edge in a realization by assigning a suitably large cost to it. This means that classical complexity results carry over to the case where edges can fail. In optimization of expectations, it is not clear that this is the case. If we assign a large cost to an edge with a certain probability, we alter its expected cost and the expected cost of any path using it. This can result in a different optimum.

Note that probability distributions are generally independent. Since stochastic dependency may give us some kind of ability to "look-ahead", there are cases where a solution can be computed efficiently. In [106], one of these cases is shown. We are given, together with the graph $G = (V, E)$, a set $S = \{1, \ldots, N\}$ of *scenarios*. A scenario $r \in S$, determines edge costs $c_e^{(r)} \in \mathbb{R}^+$ for every $e \in E$. At the beginning, a probability $p_r \geq 0$ for a scenario $r \in S$ to be the actual one is known. Upon reaching a vertex, the actual costs of the edges incident to it are discovered, and once an edge has been discovered, its cost does not change in the future. We might be able to eliminate elements from the set S, if we see that a scenario is incompatible with the edge costs that we have seen so far, and change the current path for a more promising one. We aim to find an online algorithm that produces a path between $s, t \in V$ with minimum expected cost. In the general case, where the number of scenarios N can be arbitrarily large, it is NP-hard to find such a path. If, instead, N is a fixed parameter, it is possible to find a polynomial-time algorithm to compute the path with minimum expected cost [106].

To devise such an algorithm, denote as $\mathbb{E}[v, I]$ the minimum expected cost of a path from a vertex $v \in V$ to the destination t, when $I \subseteq S$ is the set of scenarios compatible with the current situation. If $\mathbb{E}[v, I]$ is known for every vertex $v \in V$, and every possible subset I, it is easy to compute the corresponding path. At any step the online algorithm only has to update the set of compatible scenarios, and output the edge that leads to the neighbor with minimum expected cost in this situation. For a subset I of S, define $\mathcal{I}(v, I)$ as a partition of I such that a set $I' \in \mathcal{I}(v, I)$ contains all the scenarios that are compatible from v, for a realization of costs of the edges incident to v. Let $\mathbb{P}[I'|v, I]$ be the conditional probability that the actual scenario belongs to the set I', given the current set I and the current vertex v. Note that $\mathbb{P}[I'|v, I] = 0$ if and only if $I' \notin \mathcal{I}(v, I)$, and $\sum_{I' \in \mathcal{I}(v,I)} \mathbb{P}[I'|v, I] = 1$. We can therefore define the expected cost from v when the current set is I as

$$\mathbb{E}[v, I] = \sum_{I' \in \mathcal{I}(v,I)} \mathbb{E}[v, I'] \cdot \mathbb{P}[I'|v, I].$$

It is easy to see how to compute the partitions $\mathcal{I}(v, I)$ and the corresponding probabilities for every vertex and every possible set $I \subseteq S$ in time $\mathcal{O}(nN2^N)$ by enumerating all possible subsets of S. Once the partitions $\mathcal{I}(v, I)$ are known, we can compute the values $\mathbb{E}[v, I]$ for every vertex v and every possible subset $I \subseteq S$. If $|I| = 1$, then the $\mathbb{E}[v, I]$ can be determined using Djkstra's algorithm. If $\mathbb{E}[v, I']$ is known for every v and every $I' \subset S$ of size $|I'| = k$, we can compute $\mathbb{E}[v, I]$ for sets of size $|I| = k + 1$ by using the previous formula, if $I \notin \mathcal{I}(v, I)$ (i.e., upon reaching v we can reduce the set of compatible scenarios at least by one). Otherwise, the costs of the edges incident

to v are uniquely determined by I, and we can set $\mathbb{E}[v, I]$ as the minimum between the sum of the expected costs from the neighbors of v with the cost of the edge to reach that neighbor. In order to handle these two cases properly, we have to consider the vertices following the order generated by Dijkstra's algorithm from t using the currently computed costs. The overall time to find a path with minimum expected cost is $\mathcal{O}(2^N(nN + n^2))$, which is polynomial if N is a constant.

In [14], a stochastic variant of the `Recoverable-CTP` is formulated. As in the original problem, every time a vertex is reached, a new list of blocked edges adjacent to it is given as input to the algorithm, even if this vertex was already reached in a previous step. For each edge $e = (u, v) \in E$, a probability p_e of being blocked is known, and these probabilities are independent. The algorithm may decide to reject the list at the current vertex $v \in V$, and ask for another one, by paying a penalty of c_v as a "waiting cost" in addition to the cost of the path. We require that $c_v < c_e$ for every edge e adjacent to v. The rejection of a list can be modeled by adding a self-loop edge to every vertex $v \in V$ with cost c_v and $p_{(v,v)} = 0$.

In [14], the authors show an online algorithm that leads from the source s to a destination t with minimum expected cost. They start from the assumption that, for every vertex $v \in V$, there is a subset of neighbors to which it is worth to travel to, if the corresponding edge is found non-blocked. We will show later how this subset can be computed efficiently. Denote the minimum expected cost from a vertex v to the destination as $\mathbb{E}[v]$. To compute this value, we assume that the expected costs from the neighbors of v are known, as well as the subset of them to which it is worth to travel. We can sort the neighbors of v according to the sum of their expected cost with the cost of the edge that connects them to v. Let L_v denote this *priority list* with size $|L_v| > 0$, and $\mathsf{P}_i = \prod_{j=1}^{i} p_{(v,v_j)}$ denote the probability to find all the first i edges in the priority list blocked (for sake of notation, we define $\mathsf{P}_0 = 1$). In the case that at least one of the edges in L_v is not blocked, we can compute the expected cost we would have to pay in this case from v to t as

$$\alpha = \frac{\sum_{i=1}^{|L_v|} \mathsf{P}_{i-1}(1 - p_{(v,v_i)})(c_{(v,v_i)} + \mathbb{E}[v_i])}{1 - \mathsf{P}_{|L_v|}}.$$

The probability that all the edges in the priority list are blocked is $\mathsf{P}_{|L_v|}$. In this case, we assume that the best action is to reject the current list. The cost that we would get if we have to reject the list exactly one time would therefore be α, plus the cost of the self-loop c_v. The same reasoning applies if we have to reject the list exactly k times. The expected cost from v is given by the sum of these costs for every $k = 0, 1, \ldots$. It is easy to see that this series converges, and the overall expected cost from the current vertex v can be written as

$$\begin{aligned}
\mathbb{E}[v] =& (1 - \mathsf{P}_{|L_v|})\alpha + \mathsf{P}_{|L_v|}(1 - \mathsf{P}_{|L_v|})(c_v + \alpha) + \mathsf{P}^2_{|L_v|}(1 - \mathsf{P}_{|L_v|})(2c_v + \alpha) + \ldots \\
=& (1 - \mathsf{P}_{|L_v|})(\alpha + c_v(\sum_{k=1}^{\infty} k\mathsf{P}^k_{|L_v|}) + \alpha(\sum_{k=1}^{\infty} \mathsf{P}^k_{|L_v|})) \\
=& \frac{\sum_{i=1}^{|L_v|} \mathsf{P}_{i-1}(1 - p_{v,v_i})(c_{v,v_i} + \mathbb{E}[v_i]) + c_v\mathsf{P}_{|L_v|}}{1 - \mathsf{P}_{|L_v|}}.
\end{aligned}$$

In [14], the authors define an optimal priority list for a vertex v as the one that minimizes the value $\mathbb{E}[v]$. They also prove that the graph induced by the edges that are in some optimal priority list is acyclic.

To compute the optimal priority lists, we start from a situation where an empty tentative list is assigned to every vertex $v \in V - \{t\}$, and $\mathbb{E}[v]$ is set to infinity. We label a vertex as "done" if its current tentative list is optimal. At the beginning, the only vertex marked as "done" is t, and its expected cost is 0. At any step, we take every edge connecting an unlabeled vertex with a labeled one, and we insert it in the priority list of the unlabeled vertex, if doing so would decrease its expected cost. Then the unlabeled vertex with smallest expected cost is labeled "done", and we

repeat from the previous step. We continue until every vertex in the graph is labeled. The authors show that the tentative lists at the end of the computation are optimal. They also prove that the cost of the algorithm when starting from the source, $\mathbb{E}[s]$, is minimum. Using the previous idea, it is possible to compute the optimal priority lists in time $\mathcal{O}(m \log n)$. Note that the previous analysis does not constitute a competitive analysis of the online algorithm. Its expected competitive ratio is unknown.

Non-adaptive Stochastic Shortest Path

It is usually not trivial to define an appropriate objective function for a stochastic variant of an optimization problem. This is because two parameters can be taken into account, both being important from the user's point of view: the *expected cost,* and the *standard deviation*, of a solution. It is crucial to define properly what we wish to optimize.

In a non-adaptive stochastic context, we are given a graph $G = (V, E)$, where a random cost Y_{ij} is assigned to each edge $(i, j) \in E$. The cost of a path p is therefore a random variable Y_p itself, with $\mathsf{Y}_p = \sum_{(i,j)\in p} \mathsf{Y}_{ij}$. In order to establish when a path is considered optimum, we have to define a proper *objective function* over the random variable Y_p.

The most natural criterion for when a path from s to t is optimum, is its expected cost [83]. If the mean values μ_{ij} of the edges in the graph are known, then the expected cost of a path p is the sum of the expected values of the edges that are part of it, i.e., $\mathbb{E}[\mathsf{Y}_p] = \sum_{(i,j)\in p} \mu_{ij}$. This objective function is linear, therefore the optimum can be computed using classical shortest path algorithms in the graph where the cost of every edge $(i, j) \in E$ is set to μ_{ij}.

As we pointed out previously, minimizing only the expected cost of a path is not always the best choice, because it does not take into account any measure of risk aversion. For example, we might prefer a path that has slightly higher expected cost, but it is more reliable in terms of standard deviation. In case of independent random variables, a naive solution could be to minimize a bi-criteria function of the mean and variance of the produced path. Unfortunately, this may not always be tractable, since the number of Pareto optimal solutions for bi-criteria shortest path problems may be exponential in the worst case [104].

Following a common approach in economics to model users' risk aversion, in [97], the author proposes to find a path that minimizes a *convex combination* of mean and standard deviation. We are given a graph $G = (V, E)$, where for each edge $(i, j) \in E$, the mean value μ_{ij} and the variance τ_{ij} of the corresponding independent random cost are known, as well as a value $\alpha \in [0, 1]$ that represents the user's risk aversion. We are asked to find the path given by

$$\begin{aligned} \arg\min_{\boldsymbol{x}} \quad & \left\{ \alpha \sum_{(i,j)\in E} \mu_{ij} \cdot x_{ij} + (1-\alpha) \sqrt{\sum_{(i,j)\in E} \tau_{ij} \cdot x_{ij}} \right\} \\ \text{subject to} \quad & (1) \text{ and } (2). \end{aligned}$$

The constraints (1) and (2) specify the set of all feasible paths, denoted as $\mathcal{X}$. This set can be seen as a subset of the unit hyper-cube in $|E| = m$ dimensions, and all the feasible paths can be represented as elements $\boldsymbol{x} \in \mathcal{X}$. If the edge $(i, j) \in E$ is part of a path $\boldsymbol{x}$, we will write $(i, j) \in \boldsymbol{x}$. We can consider a relaxation of this set, over the continuous convex hull generated by the elements of $\mathcal{X}$, denoted as $\mathrm{Conv}(\mathcal{X})$. For any choice of $\alpha \in [0, 1]$, the above objective function is concave in $\mathrm{Conv}(\mathcal{X})$, and it is known [27] that it is minimized at an extreme point of this convex hull. The corresponding solution is also feasible and optimum for the discrete problem. In [98], the author shows that it is not necessary to check the whole boundary of $\mathrm{Conv}(\mathcal{X})$, because an optimal solution lies in a smaller restricted space. In particular, the author shows that a solution lies on an extreme point of projection of the convex hull on the linear span generated by the vectors $\boldsymbol{\mu} = (\mu_{ij})$ and $\boldsymbol{\tau} = (\tau_{ij})$. However the number of such extreme points can be $n^{1+\log n}$ in the worst-case. In [97] it is shown that it is possible to identify all such points, and that the algorithm that enumerates all

of them, looking for the optimal solution, has good practical performance on randomly generated grid graphs.

In [96], this model is further generalized. The intuition behind this generalization comes from the consideration that, usually, the cost of an edge corresponds to the time it takes to traverse it. If, for example, a deadline $t_0 \in \mathbb{R}$ is known, we can use values smaller than t_0 to denote early arrivals, and values greater than t_0 for late arrivals.. We can now devise a function $\mathcal{C} : \mathbb{R} \to \mathbb{R}$ that penalizes early and/or late arrivals to the destination by assigning them a larger value than the one that we would get if we are on time. For example, we could take $\mathcal{C}$ as a quadratic function that is minimized at t_0, if we wish to penalize early and late arrivals in the same way. In general, we could use any, even non-convex, function as our penalty $\mathcal{C}$, as long as it attains a minimum at t_0. The nature of the penalty function, determines the complexity to compute a solution. In the stochastic scenario, where edge costs are represented by random variables, we look for a path that minimizes

$$\begin{array}{ll} \arg\min\limits_{\boldsymbol{x}} & \mathbb{E}[\mathcal{C}(\mathsf{Y}_{\boldsymbol{x}})] \\ \text{subject to} & \boldsymbol{x} \in \mathcal{X}, \end{array}$$

Alternatively, we can imagine $\mathcal{C}$ as a utility function, instead of a penalty function [93, 83]. The goal then is to maximize the expected utility. In [96], it is shown that it is NP-complete to find a path that minimizes the above objective if $\mathcal{C}$ is a general function with global minimum and independent random edge costs, and it is hard even to approximate within a constant factor for simple paths. The authors also present an algorithm that finds a non-simple path for the above problem if $\mathcal{C}$ allows to express the objective function in a simple closed form. This algorithm is pseudo-polynomial in the maximum mean value of the random edge costs.

In [96], this same problem is also looked at from a slightly different perspective. Referring to the intuition that the cost of an edge is the time it takes to traverse it, we can argue that in the above problem we are assuming to be leaving the source vertex at a fixed time $\tilde{t} = 0$. We can study a related problem where choosing the right starting time is part of the optimization, along with the path. That is, we allow to change the beginning of the trip, while looking for a path *and* a leaving time $\tilde{t}$ that assure minimum penalty. Formally, we look for the path $\boldsymbol{x}^*$, and the time t^* given by

$$\begin{array}{ll} \arg\min\limits_{\boldsymbol{x},\tilde{t}} & \mathbb{E}[\mathcal{C}(\tilde{t} + \mathsf{Y}_{\boldsymbol{x}})] \\ \text{subject to} & \boldsymbol{x} \in \mathcal{X}, \tilde{t} \in \mathbb{R}. \end{array}$$

In some cases, it turns out that this is a much simpler problem than the original one. In [96], the case where the penalty function is quadratic $\mathcal{C}(y) = y^2$ is considered. The corresponding objective function can be expressed in closed form, i.e., $\mathbb{E}[\mathcal{C}(t + \mathsf{Y}_{\boldsymbol{x}})] = (t + \mu_{\boldsymbol{x}})^2 + \tau_{\boldsymbol{x}}$, where $\mu_{\boldsymbol{x}}$ and $\tau_{\boldsymbol{x}}$ are the mean value and the variance of path $\boldsymbol{x}$. It is easy to see that the starting time that minimizes its expected penalty is $t^* = -\mu_{\boldsymbol{x}}$ (therefore the quadratic term in the objective is 0 for any path), and that a path $\boldsymbol{x}^*$ with minimum variance is therefore optimum (since it minimizes the term $\tau_{\boldsymbol{x}}$ in the objective). Such a path can be found efficiently using classical shortest path algorithms.

In [52], it is proposed to look for the path that *maximizes* the probability to not exceed a cost c. For example, we want to be as confident as possible that we will not exceed a given deadline. Namely, we want to find the path given by

$$\begin{array}{ll} \arg\max\limits_{\boldsymbol{x}} & \mathbb{P}\left[\mathsf{Y}_{\boldsymbol{x}} \leq c\right] \\ \text{subject to} & \boldsymbol{x} \in \mathcal{X}, \end{array}$$

where c is a value provided as input. The authors characterize the optimal solution for some special cases, but they do not give complexity results.

For this problem, the authors in [99] consider a setting where the random edge costs are independent and normally distributed. In this case, the objective function can be written as

$$\mathbb{P}\left[\mathsf{Y}_{\boldsymbol{x}} \leq c\right] = \mathbb{P}\left[\frac{\mathsf{Y}_{\boldsymbol{x}} - \mu_{\boldsymbol{x}}}{\sqrt{\tau_{\boldsymbol{x}}}} \leq \frac{c - \mu_{\boldsymbol{x}}}{\sqrt{\tau_{\boldsymbol{x}}}}\right] = \Phi\left(\frac{c - \mu_{\boldsymbol{x}}}{\sqrt{\tau_{\boldsymbol{x}}}}\right),$$

where Φ is the cumulative distribution function of the standard normal random variable, with mean value 0 and variance 1.

Since the function Φ is monotone increasing, a solution that maximizes it also maximizes its argument. The problem becomes to find the path given by

$$\begin{array}{ll} \arg\max_{\boldsymbol{x}} & \dfrac{c - \mu_{\boldsymbol{x}}}{\sqrt{\tau_{\boldsymbol{x}}}} \\ \text{subject to} & \boldsymbol{x} \in \mathcal{X}. \end{array}$$

No efficient algorithm to minimize this objective function, that is not separable and not convex, is known. In [99], the authors show that this problem has a peculiar structure that forces an optimal solution to lie on the boundary of the feasible set. In particular, they show that it lies in a subset of the extreme points of the feasible set of size at most $n^{\Theta(\log n)}$. The algorithm that finds an optimal solution by enumerating all the elements of this set therefore has sub-exponential worst-case running time.

2.1.3 Summary

We have shown some known techniques to handle shortest path problems under uncertainty. We classified these techniques by their similarities and differences. From an abstract level, we have distinguished two main classes into which these techniques fall: *worst-case optimization*, and *optimization of expectations*. All techniques rely on the assumption that a model describing the possible realizations of the uncertain data are known.

In worst-case optimization, the probability distributions of the realizations are unknown. Our best chance is to pick a path that has an acceptable cost in every realization that may occur. Two criteria are usually considered in worst-case optimization: an *absolute* criterion, and a *maximum deviation* criterion. With the former criterion, we look for a path that minimizes its maximum cost over all the realizations. The latter criterion yields a path that minimizes its maximum cost distance (or ratio) compared to the optimal cost in each realization.

In optimization of expectations, we assume that the probability distribution governing the input is known, and that we can exploit this knowledge when designing paths. The cost of a path is expressed as a random variable and it is not clear how to optimize it. We might be interested in keeping both the expected cost and the standard deviation of an optimal solution as low as possible. This leads us into the domain of multi-criteria optimization, where even devising a proper objective function is not a trivial task.

We provide a further categorization, based on *when* the uncertain data becomes known. In *adaptive* settings, the data is discovered piece by piece after the actual trip is started. We are allowed to modify the solution according to new information. The evaluation of paths for adaptive settings comes from competitive analysis. We usually compare how well our adaptive path performs with respect to the optimal one. In *non-adaptive* settings, we are not allowed to modify the path once we fix it. We need to be sure that the chosen path is good with respect to *all "likely"* realizations.

What follows is a brief summary of the known problems and results according to the above categorization.

Worst-case Optimization

Adaptive:
In worst-case optimization of adaptive problems, we are asked to find a path in a graph where

edges may change over time, and we are allowed to modify the decision when new information is discovered. The following table summarizes results for this setting:

	Maximum Deviation	*Absolute*
`CTP`	PSPACE-complete	?
k-`CTP`	?	PSPACE-complete
`Recoverable-CTP`	?	$\mathcal{O}(k^2m + kn \log n)$
`2-Valued Graph`	PSPACE-complete	?

`Canadian Traveller Problem` Edges can be blocked.

k-`CTP` The number of blocked edges is bounded by k.

`Recoverable-CTP` Blocked edges can recover, the number of times edges can be blocked is bounded by k, and we may wait at vertices.

`2-Valued Graph` Edges have two costs, the actual cost is discovered only when one of its endpoints is reached.

Non-adaptive:
For worst-case optimization of non-adaptive problems, we are asked to find an optimum path in a situation where the costs of the edges are uncertain, but we are not allowed to change the initial path:

	Maximum Deviation	*Absolute*
`Scenario Based`	NP-complete	NP-complete
`Interval Edge Costs`	NP-hard	$\mathcal{O}(m + n \log n)$
`Costs from Polytope`	NP-hard	NP-hard
`Robust Opt Framew`	?	$\mathcal{O}(m^2 + mn \log n)$
k-`distRR`	?	NP-hard
`Rent-RR`	?	NP-hard

`Scenario Based` The combinations of edge costs are taken from a scenario set.

`Interval Edge Costs` Edges have a cost interval associated with them, and the actual costs are taken from the corresponding intervals.

`Edge Costs from Polytope` The combination of edge costs is taken from a convex polytope.

`Robust Optimization Framework` Edges have a cost interval associated with them, the cost of at most a constant number of edges can differ from the lower end of the interval.

k-`distRR` We are asked to find *two* paths that minimize the sum of first-stage and second-stage edge costs, and that differ by at most k edges. Only the second-stage costs are uncertain.

`Rent-RR` We are asked to find *two* paths that minimize the sum of the first-stage and second-stage edge costs, and we have to pay a penalty for every additional edge used in the second stage. Both first- and second-stage costs are uncertain.

Optimization of Expectations

Adaptive:
In adaptive optimization of expectations, we are asked to find an optimum path in a graph, where edges may change over time according to known probability distributions.

	Competitive ratio	*Expected cost*
`Stochastic CTP`	?	?
`Stochastic Valued Graph`	#P-hard	#P-hard
`Stochastic Scenario (General)`	?	NP-hard
`Stochastic Scenario (Fixed scenarios)`	?	$\mathcal{O}(2^N(nN+n^2))$
`Stohastic Recoverable-CTP`	?	$\mathcal{O}(m \log n)$

`Stochastic Candian Traveler Problem` Edges can be blocked with a given probability.

`Stochastic Valued Graph` Edges have a discrete probability distributions governing their costs. The actual cost is discovered when the edge is reached.

`Stochastic Scenario Based` The combinations of edge costs are drawn from a scenario set according to known probabilities.

`Stochastic Recoverable-CTP` Edges can be blocked with known probability and can recover over time.

Non-Adaptive:
In non-adaptive optimization of expectations, we are asked to find an optimum path in a graph with random edge weights with respect to different objective functions.

Objective Function	
`Minimum Expected Cost`	$\mathcal{O}(m + n \log n)$
`Convex combination` μ, τ	$n^{\Theta(\log n)}$
`Penalty Function (easy)`	$\mathcal{O}(n \log n)$
`Penalty Function (hard)`	NP-hard
`Prob not to exceed cost`	$n^{\mathcal{O}(\log n)}$

`Expected Cost` An optimum path minimizes expected cost.

`Convex combination of` μ, τ An optimum path minimizes a convex combination of expected cost and variance.

`Penalty Function` An optimum path minimizes a function over expected cost.

`Probability to not exceed a given cost` An optimum path maximizes the probability not to exceed a given cost.

2.2 Eco-Aware Cost Models for Routing

When computing an optimized route suggestion for a given pair of origin and destination, the first question is which objective will be used in the optimization. Textbooks usually talk about *shortest paths* when addressing route planing, implying that routes are optimized for minimum distance. In road networks, however, the differences in road class, e.g. a freeway versus a gravel road, render the measure of pure distance useless for practical applications. Hence, the predominant criterion used in practical vehicle route planning is *travel time*, i.e. routes are mainly optimized for earliest time of arrival (ETA), with different providers of optimized routes tweaking their objective functions a bit to make routes even better in their customers' view.

One of the goals of eCOMPASS is to compute *eco-friendly* routes, and the first imminent question is how to define a cost model which yields an eco-friendly routes result when computing an optimized route for given origin and destination. In this section, we will look at the current

status of such features from a market perspective, conduct a brief review on scientific literature on the topic, and finally point out challenges and opportunities for eCOMPASS.

In this section we present cost models that attempt to take into account the environmental footprint of the proposed route plans. In particular, Section 2.2.1 is an overview of currently used techniques in the vehicle navigation market. Section 2.2.2 presents related literature for eco-aware cost models. Finally, Section 2.2.3 presents a work that considers eco-aware cost models in time-dependent road networks.

2.2.1 Eco-Routing in the Navigation Market

In recent years, many major navigation providers have started to add features for eco-routing to their products. **TomTom** offers an "Eco Routes" route planning mode which computes routes that use less fuel, based on advanced road attributes in their map data. **Navigon** offers a self-learning feature called "MyRoutes", which records the speed the user traveled at certain times of day on streets of different road classes. These recorded speeds are used to adapt route planning to reflect the user's individual driving style. By this, Navigon claims, a better trade-off between travel time and distance can be realized, aiding in saving fuel. Finally, **Garmin** offers a quite elaborate set of eco features, namely "ecoRoute", "ecoRoute HD", and "ecoChallenge". The first feature is a route planning option which optimizes the computed route for fuel consumption based on map data like changes in speed limit, anticipated number of stops, plus the vehicle's fuel economy as entered by the user. The latter two features provide driver feedback and will be described in more detail in Section 2.6.

The German professional navigation journal NAVIconnect compared three top-of-the line devices by TomTom, Navigon, and Garmin with respect to fuel consumption in their March 2009 issue [121]. Three testers drove the routes suggested by the three devices in three identical cars, switching cars frequently to even out differences due to the drivers. The routes visited six destinations, covered 283 (Garmin), 320 (Navigon), respectively 332 (TomTom) kilometers, and included inner city as well as cross-country driving. On the Garmin device, the ecoRoute option was used for route planning, and on the Navigon, the corresponding MyRoutes feature. Quite interestingly, the TomTom device was used with the regular "Fastest Route" routing option, not "Eco Routes", because a similar TomTom device without the Eco Routes feature had exhibited the best fuel economy in an earlier test. All three devices included a traffic information service: The Garmin and the Navigon were equipped with a TMC antenna, receiving traffic updates over radio channels, while the TomTom device used the companies HD Traffic live feed received via GPRS. Moreover, it uses TomTom's IQ Routes feature, a database of historical speeds on the entire road network for each time of each weekday, yielding more realistic travel times at any time.

In the comparison, the Garmin device led to the lowest total fuel consumption of 17.7 liters total. It also had the shortest route, 283 kilometers, but it took by far the most time, 376 minutes. The car with the TomTom device used 20.2 liters fuel total on a total distance of 332 kilometers, but it had the lowest total travel time of 286 minutes, ninety minutes, or 24%, less than the Garmin. Moreover, its route yielded the most efficient drive, using only 6.1 liters per 100 kilometers on average, as compared to 6.3 with the Garmin, and even 7.1 liters with the Navigon. Finally, the Navigon device led to a total fuel consumption of 22.8 liters over 320 kilometers, with a total travel time of 302 minutes. It is also notable that the Garmin and the Navigon underestimated travel time by seventy and 51 minutes, respectively, while the TomTom overestimated it by just fourteen minutes.

In the conclusion, the author consequently states the following: While ecoRoute on the Garmin does apparently yield a route with very low fuel consumption, it is a feature that will not be widely accepted by users, because a 24% increase in travel time, paired with the discomfort of travelling mainly on small roads, constitutes an unacceptable user experience for most. The TomTom, on the other hand, manages to yield a route using less fuel than the Navigon by a larger margin, while delivering a superb user experience with its route avoiding traffic and slow turns efficiently by use

of HD Traffic and IQ Routes, which is also indicated by its much more accurate ETA estimate. We will draw conclusions from these insights in Section 3.2 below.

Another trend which seems to accelerate the quest for consumption-based cost models in routing is the current introduction of purely electric vehicles (EVs) to the mass market. A common phenomenon frequently mentioned in relation to EVs is users' range anxiety: Since the range of EVs is limited and they can only be refueled at certain charging stations, users demand regular reassurance that they can reach their destination as planned, i.e. without running out of power. Also, if needed, stops at charging stations need to be incorporated on longer routes by EV navigation systems.

Obviously, the remaining range of an EV can only be estimated when a somewhat accurate energy consumption model is available for routing. A first system using such model has recently been introduced to the market by TomTom for Renault, the Renault Carminat TomTom Z.E. LIVE, which will be a standard feature in Renault's Z.E. series of electric vehicles. This system is based on input from several different sources: It uses a self-learning, car model specific consumption model which permanently adapts to the user's driving style; elaborate online input from the car itself, like battery status, current consumption of auxiliary systems like heater and headlights; and finally, it relies heavily on advanced road attributes in the routing map, like elevation data, stop probabilities at intersections, etc.

2.2.2 Literature on Fuel Consumption Models

The diploma thesis *Planung energieeffizienter Routen in Straßennetzwerken* (Energy-Efficient Route Planning in Road Networks) done by S. Neubauer and supervised by R. Geisberger et al. [95] investigates the application of eco-aware quality measures to route planning. They evaluate several variants of two main fuel consumption models, one simplified physical and one based on version 2.1 of the *Handbook Of Emission Factors* (HBEFA) [9], developed on behalf of the Environmental Protection Agencies of Germany, Switzerland, Austria, Sweden, Norway and France as well as the JRC (European Research Center of the European Commission). From these models the authors construct a fuel consumption metric for the routing graph, i. e., they compute the consumption on each road segment. With this metric (and a subsequent application of Dijkstra's Algorithm) they can evaluate the influence of the model variants on the course of the resulting route.

The metric based on HBEFA takes into consideration base consumption for three street categories (highway, overland, inner city) of a EURO4-type car. In order to account for the slope of a street they apply a correction factor from the same handbook (defined for slopes between -6% – $+6\%$). The result is multiplied by the length of the street segment to get the final consumption estimate. It is noteworthy, that HBEFA contains far more detailed data such as more fine-grained street categories. Even more importantly the authors do not take into account traffic conditions but average over the respective entries in the handbook to get their base consumption numbers.

The metric based on physical modeling takes into account resistances (slope, aerodynamics, rolling but *not* inertia), combustion and transmission efficiency (very roughly parameterized as more detailed input was not available to the authors). The consumption is evaluated on basis of the average velocity of a street segment, typically based on the speed-limit. The authors do not account for different traffic conditions neither do they account for acceleration/deceleration on or between road segments.

During their evaluation, the authors observe that purely optimizing fuel consumption leads to very undesirable routes that spend a lot of extra travel-time in order save just a few more cents in fuel. Therefore they apply a mixed goal function as a trade-off of both the price of the fuel spent and a given hourly wage of the driver. They can show that for some ratio between hourly wage and fuel optimization the resulting routes are almost as fast as a respective route optimized for travel-time while at the same time being more friendly towards the environment, typically by avoiding highways and regions of steep ascent.

On the other hand, the authors observe that highway routes are less fuel consuming if the travel speed on respective segments is reduced (remember that they only apply *one* velocity on each

segment). It turns out that highway routes with reduced speed are often even more environmentally friendly than overland routes at maximum speed. The authors do not reflect this insight in their algorithm design, which would require them to look at the consumption of an edge for *several* different speeds.

As traffic conditions have a huge impact on fuel consumption and emission the negligence of traffic condition as input to parameterization of the consumption metric seriously hampers the authors' ability to be conclusive about the impact of different routes on consumption.

After the authors realize how poor their assumption of constant speed per road segment really is, they turn in the last part of the thesis towards inner-city traffic. They describe the effect that turns, right-of-way and traffic lights have on the fuel consumption of combustion engines. Instead of taking these considerations into account within their graph metric, the authors turn towards small-scale simulation of tiny subnetworks of a city, e. g. a single four-way corner. The results of this simulation are at least questionable.

2.2.3 Fuel-optimal Paths in Time-Dependent Networks

A first approach to fuel-optimal paths in time-dependent networks has been considered in [76] which, to the best of our knowledge, is the only one currently known such approach. The problem consists in finding paths that use the least fuel (fuel-optimal paths) in time-dependent networks[1]. This means that a graph $G = (V, E)$ is considered where V is the set of nodes and E is the set of edges. For every edge e, there is a piecewise linear function $f_e(t)$ which denotes the time needed to traverse the edge.

Fuel Consumption Function. In order to calculate fuel-optimal paths, a fuel consumption function must be defined. It is proven that fuel consumption is directly linked with CO_2 emissions. The minimization of fuel consumption leads to the minimization of CO_2 emissions. An important observation is that a vehicle that keeps moving through a road segment has to overcome a sum of resistances. According to various studies, the fuel consumption function B for a given time interval $[0, T]$ is defined as:

$$B = \int_0^T \max\left\{0, \frac{(F_{climb}(t) + F_{roll}(t) + F_{aero}(t) + F_{inert}(t) + F_{brake}(t))\upsilon(t) + P_0(t)}{H_l \cdot \eta_e(t) \cdot \eta_t(t)}\right\} \mathrm{d}t$$

where $F_{climb}(t)$ is the climbing resistance, $F_{roll}(t)$ is the rolling resistance, $F_{aero}(t)$ is the aerodynamic resistance, $F_{inert}(t)$ is the inertial resistance and $F_{brake}(t)$ is the breaking resistance introduced to model the influence of the braking system of the vehicle. The velocity of the vehicle is denoted with $\upsilon(t)$. Auxiliary energy consumers like radio, AC, light and other are denoted as $P_0(t)$. Since no fuel can be generated in a combustion-engined vehicle the fuel consumption is 0 when the driver leaves the throttle. The terms in the denominator are the lower heating value H_l of the utilized fuel, the engine efficiency $\eta_e(t)$ and the transmission efficiency $\eta_t(t)$.

Speed Distributions in Road Networks and Historical Traffic Data. Let us consider a road segment of length L on which the free flow velocity plots result in a normal distribution of the measured vehicle speeds. The assumption that traffic follows the normal distribution is true for highway traffic and does not hold for urban traffic (only in free flow the assumption is true). There are four events E_i, $i = \{1, 2, 3, 4\}$ that are used to capture the urban traffic:

- E_1 : A car drives freely throughout the road segment

[1] Route plans in time-dependent road networks are covered more thoroughly in Section 2.4. Here we mainly focus on the fuel-optimal aspect.

- E_2 : A car accelerates from the speed 0 to some speed V_0^+, and continues driving freely until the end of the road segment
- E_3 : A car drives freely until it is forced to decelerate from some speed V_0^-, to the speed 0 and then is forced to stop for time T_h
- E_4 : In road segment $[0, L/2]$ a car accelerates from 0 to V_0^+, drives freely until it is forced to brake from V_0^- to 0 in road segment $[L/2, L]$, and then is forced to stop for time T_h

Although the model is simple, it captures the presence of junctions in urban traffic. The assumption made is that V_0^+, V_0^-, T_h are independent random variables. The variables V_0^+, V_0^- follow the normal distribution $N(\mu_0, \sigma_u^2)$ and variable T_h follows the uniform distribution $U([0, T])$ where T denotes time and can be interpreted as the duration of the red light in a junction.

One fundamental issue is the update of historical traffic data. Digital maps were used in order to get average speeds for every segment of the road network. Every event E_i, $i = \{1, 2, 3, 4\}$ is associated with a probability p_t: $P\{E_1\} = (1-p_t)^2, P\{E_2\} = P\{E_3\} = p_t(1-p_t)$ and $P\{E_4\} = p_t^2$. Then a probabilistic analysis for each event was made and probability density functions that model traffic were computed.

The Road Network as a Time-Dependent Network. A time-dependent network is a quintuple $G = (V, E, \tau, \beta, \delta)$ where:

- V is the set of nodes
- E is the set of directed edges
- τ is the travel time function
- β is the travel cost function
- δ is the waiting cost function

The time dependent network introduces the sense of time so the complexity of the problem increases. For every time point t_i, a static version of the shortest path problem (in this case fuel optimal path problem) has to be solved. The problem is NP-Hard in the general case [76]. In order to achieve solutions in polynomial time the network has to respect the FIFO or non passing property. This means that for two vehicles a, b starting at times t_1, t_2 with $t_1 < t_2$ from a node v_1 to node v_2, vehicle a will arrive earlier at node v_2 than vehicle b. In other words, vehicle b can not overtake vehicle a in time.

Since in a time-dependent network the travel time function τ and the cost functions β, δ depend on the time variable, it is convenient to define the state space $X = V \times R$. In particular, each state $\chi \in X$ is a pair (v, t), consisting of a node $v \in V$ and a time $t \in R$. A state transition (also called a control action) is defined as a pair $u = (\Delta t, e)$ consisting of a waiting time and an edge $e \in E$. The application of a control action u in the state χ is called *control to state mapping:*

$$\varphi((v, t), (\Delta t, e)) = (\omega(e), t + \Delta t + \tau(e, t + \Delta t)).$$

The second argument of τ denotes the departure time on the edge determined by the first argument. The notation $\omega(e)$ shows the target node of edge e. The meaning of function φ is that if one is at state χ at time t and applies a control action u requiring time Δt along an edge e, he is led to the node $\omega(e)$ in which he arrives at time $t + \Delta t$ plus the time τ shown from the travel cost function.

Some additional definitions must be presented. A control action $u = (v, t)$ is *admissible* for a given state $\chi = (v, t)$ if $u \in \Delta T$ and $\varphi(\chi, u) \in X$, where ΔT is the set of admissible waiting times. A sequence of control actions $u = (u_k)_{k=1,2,\ldots}$ is called admissible for a given state $\chi_k \in X$, if u_k is admissible for $\chi_{k-1}, k = 1, 2, \ldots$ where:

$$\chi_k = \varphi(\chi_{k-1}, u_k), k = 1, 2, \dots$$

A path p is a sequence of states $p = (\chi_0, \chi_1, \dots)$ such that there exists a sequence of control actions $u = (u_k)_{k=1,2,\dots} \in U(\chi_0)$, which satisfies the above definition, where $U(\chi_0)$ is the set of all finite action control sequences admissible for χ_0.

The path cost function is then defined as:

$$b(\chi_0, u) = \sum_{k=1}^{|u|} [\delta(\upsilon_{k-1}, t_{k-1}, \Delta t_k) + \beta(e_k, t_{k-1} + \Delta t_k)]$$

Given an initial state $\chi_0 \in X$ and a goal node $\upsilon' \in V$ the fuel optimal path p^* can be found if there exists a finite control sequence $u^* \in U(\chi_0), \omega(u^*) = \upsilon'$ and $b^*(\chi_0, u) = \inf\{b(\chi_0, u) : u \in U(\chi_0), \omega(u) = \upsilon'\}$, where $U(\chi_0)$ is the set of all finite control sequences admissible for χ_0.

The idea behind the previous definitions is that a path can be expressed as a sequence of states $(\chi_0, \chi_1, \dots)$ where a sequence of control actions $u = (u_k)_{k=1,2,\dots} \in U(\chi_0)$ is applied. In order to find the optimal path p^*, only admissible control actions must be considered.

DOT* and TD-APX Algorithms. The algorithms proposed to solve the time dependent fuel optimal path problem use the method of dynamic programming. This method is used to solve complex problems by breaking them in simpler subproblems. Then the solutions of the subproblems are combined to get the solution for the initial problem. In order to apply the method of dynamic programming a plethora of theorems and lemmas that regard the form of travel time function τ, travel cost function β, and waiting cost function δ are proven. The main properties of these functions are piecewise linearity and continuity, for more details see [76, Chapter 5]. Let $\upsilon' \in V$ be a goal node. The dynamic programming equations are:

$$b^*(\upsilon', t) = 0, \forall t \in T(\upsilon')$$

$$b^*(\upsilon, t) = \min_{u \in U(\upsilon,t), u=(\Delta t, e)} [b^*(\varphi(\upsilon, t), u)) + \delta(\upsilon, t, \Delta t) + \beta(e, t + \Delta t)], \forall \upsilon \in V \setminus \{\upsilon'\}, t \in T(\upsilon)$$

where $T(\upsilon)$ is the set of admissible points in time at $v \in V$.

The DOT* algorithm is an extension of the DOT algorithm standing for decreasing order of time. The main idea of the DOT algorithm is the following: compute the optimal value function b^* and the corresponding optimal paths by scanning backwards in time. By using a different interpretation of chronological scan algorithms, i.e., by extending Dijkstra's idea of sorting cost values to sorting cost functions, the concept of DOT methods is generalized to a heuristic search algorithm. The difference of the DOT* algorithm is that it uses a new decision rule that determines a node $\hat{v}$ and a time interval $\hat{I}$ for which the optimal value function b^* can be determined in one iteration of a chronological scan algorithm. This decision rule is a generalization of Dijkstra's shortest path decision rule. In a backwards implementation of Dijkstra's shortest path algorithm, in each iteration, the open node with the minimum cost value is identified and declared as closed. Then, each of its non-closed predecessors is declared as open and its cost value is updated according to the dynamic programming equation. With the same idea in mind, in discrete-time time-dependent networks the computation of one cost value in each iteration is sufficient. In this case, a node $\hat{v}$ and a time interval $\hat{I}$ are computed in order to minimize the computational overhead.

The TD-APX is an algorithm which approximates the optimal value function and the corresponding optimal paths. Assuming that the network satisfies the FIFO-property, the main idea is the generation of two initial solutions in polynomial time, which are then used to iteratively approximate the optimal solution. The approximation algorithm is introduced since the computational complexity of determining an exact solution may still be infeasible in many practical applications.

Experiments. The experimental evaluation of the algorithms proposed was made using the road network of the German city of Ingolstadt. The collection of data was made as follows: the speed values which form the basis for the evaluation of the algorithms are derived from taxi car data. Based on all data which has been received for one particular road segment e during a time interval $[t_k - 15min, t_k], t_k = k \cdot 5min, k = 0, ..., 287$, the current average speed $\bar{v}_k(e)$ on this road segment is determined.

The time-dependent network of Ingolstadt (the graph (V, E)) was provided by PTV AG. It contains 4447 nodes, 11808 (directed) edges. The hardware used for the experiments is: an 2quad-core Xeon E3750 with each processor clocked at 2.33 GHz and provided with 8 MB of L2 cache. The machine has 16 GB of RAM, the operating system is Ubuntu Linux 2.6.32-25-generic, which was compiled with GCC 4.4.3. The implementation of algorithms is written in Java and compiled with Java version 1.6.0 18.

Four scenarios are considered to evaluate the proposed algorithms. In the first scenario, the benefits of using precise lower bounds in the DOT* algorithm is evaluated using a set of 10 test cases using the time frame [7:58 h, 8:02 h] taken as basis. The utilization of precise lower bounds has a drastic impact on the performance of the DOT* algorithm. The average query time varies from 33 sec, 38 sec and 41 sec (best cases) to 2301 sec, 993 sec and 813 sec (worst cases). The behaviour of DOT* depends on a parameter $s \in [0, 1)$ which yields different results. In the second scenario, the best four cases of scenario 1 (fastest cases) are taken into consideration for DOT* using a different time frame [7:50 h, 8:00 h] as a basis, then are compared with TD-APX algorithm. The best average completion time of DOT* is 834 sec, while for TD-APX is 0.3 seconds. So, the average runtime of theTD-APX algorithm outvalues the average runtime of the DOT* algorithm by 3 orders of magnitude. Another observation is that the runtime of the DOT* algorithm is highly sensitive to the input data.

In the third scenario the TD-APX algorithm is evaluated in more detail. The time frames used are: [7:00 h, 7:30 h],[7:00 h, 8:00 h],[7:00 h, 8:30 h],[7:00 h, 9:00 h],[7:00 h, 10:00 h],[7:00 h, 11:00 h] and [7:00 h, 12:00 h]. The average computation times vary from 2 sec (for the shortest time interval ([7:00 h, 7:30 h]) to 30 seconds for the longest time interval [7:00 h, 12:00 h]. In the fourth scenario, the influence of the time of day and the influence of the day of the week is evaluated regarding the solutions of the time-dependent optimal path problem. The time frames used are [1:00 h, 2:00 h] and [7:30 h, 8:30 h] for days Tuesday-Thursday (working week) and Sunday. The solutions of the optimal path problems are analysed in terms of the average energy consumption. In the considered test cases, the average energy assumption associated with energy-optimal paths is about 10% lower than the average energy consumption associated with the control sequence which allows the latest departure time at the source node.

2.3 Time-Independent Route Planning

Finding shortest paths is a well-studied problem in algorithmic research: given a weighted graph $G = (V, E)$ with nodes V, edges (or arcs) E, and a weight (or length) function $\text{len} : E \to \mathbb{R}$, and given a source node s and a target node t, the goal is to compute a path $p = s\, e_1\, u_1\, e_2\, \ldots\, u_k\, e_k\, t$ (an alternating sequence of nodes and edges from s to t) with the least sum of edge weights of all paths from s to t. The Shortest Path Problem (SPP) can be solved by Bellman-Ford's algorithm [25] in the absence of negative cycles, or by Dijkstra's algorithm [47] if edge weights are also positive.

While Dijkstra's algorithm is fastest in general, algorithms with lower asymptotic complexity or better average performance have been devised for special graph classes. We review relevant literature in theoretical computer science on distance oracles in sparse, planar or bounded-genus graphs in Section 2.3.1. We then proceed with an overview of approaches specifically targeting (static) road networks in the field of algorithm engineering in Section 2.3.2; this experimentally inclined research has rapidly improved route planning performance on road networks, enabling todays fast web-based route planning services. In Section 2.3.3, we discuss extensions of these approaches to settings where multiple criteria (such as travel-time and costs) should be considered

during optimization. Finally, in Section 2.3.4 we give an introduction to the general Vehicle Routing Problem (VRP).

2.3.1 Distance Oracles for Sparse Graphs

Aside from the results portrayed in Section 2.3.2, extensive research in theoretical computer science has been spent on the fast computation of shortest paths in certain graph classes such as planar graphs. We review some of the results of that field of research in the following.

Overview of Distance Oracles

A *distance oracle* is a data structure, created from a given $n-$node graph with arc lengths and no negative cycles, that aims at answering arbitrary origin-destination queries as fast as possible. The trivial solution would be to compute the entire $n \times n$ distance matrix of the graph by solving the corresponding *all-pairs shortest paths* (APSP) problem, which would require polynomial time (for preprocessing) and $O(n^2)$ space to store the data structure itself, and would assure responses to arbitrary queries in constant time. Of course in many realistic cases, such as continental-size road networks, the occupation of such a huge amount of space is actually prohibitive. Therefore, rather than computing and storing an entire $n \times n$ distance matrix, we wish to *precompute* alternative data structures (distance oracles) that: (i) require strictly sub-quadratic (actually, close to linear) space, (ii) are polynomial-time computable, and (iii) allow for *efficient* (typically sublinear, poly-logarithmic, or even constant, if possible) computations of shortest-path distances (aka *queries*) for arbitrary pairs of nodes. Depending on the nature of the application, we may be interested in either exact or approximate answers to shortest-path distance queries, focus on directed or undirected graphs, or in special classes of graphs such as sparse, planar, or bounded-genus graphs, etc.. In case of approximations, the provided estimation should be an *upper bound* on the actual distance between the required nodes in the graph. The data structure should also allow for the reconstruction of a path with the given guarantee on its length, in time that is proportional to the number of arcs comprising it.

When considering approximate distance oracles, two notions of approximation are usually considered: The first, the *(multiplicative) stretch* $\alpha \geq 1$, is the worst-case ratio (over all possible pairs of nodes) of the query result over the corresponding shortest-path distance. The second, the *additive-stretch* $b \geq 0$, demands that the approximate distance of any source-destination pair is at most the actual distance *plus* the given additive term b. When we wish to refer to both notions of approximation, we typically refer to the $(a, b)-$*stretch factor* of a distance oracle, implying a multiplicative stretch $\alpha \geq 1$ and an additive stretch $b \geq 0$.

Exploiting Graph Decompositions

Our focus in the project is on large-scale road networks, which are sparse directed graphs. For such graphs one may use a *graph decomposition* that will assist the shrinking of the instance before applying well-known algorithms for solving a problem (e.g., Dijkstra for shortest path computations).

The main idea for providing distance oracles is the following: The original graph is recursively partitioned into arc-induced subgraphs (usually called *pieces*), where each piece is connected to the rest of the graph via a small subset of *boundary nodes*. Various data structures attempt to store, compute, or even efficiently cover (in case of approximations) distances among boundary nodes in the graph, or between a boundary vertex and an internal vertex.

Graph Separators. A set S of nodes in a graph G is called a *separator* if its removal from G results in G being split into components. We typically wish to find *balanced separators* which assure that the size of each component is at most a constant fraction of G's original size, where the size of a graph is considered to be the number of its nodes. Additionally, we would like S itself to have

size significantly smaller than the size of G. Observe that for any two nodes $u, v \in V(G)$ belonging to different components of G w.r.t. a given separator S, it holds that any $uv-$path has to cross a node of S. Therefore, assuming that we already knew the graph distances in G of u, v from/to every node $w \in S$, then a trivial minimization operation among the nodes of S would return the shortest path distance $d_G(u, v)$:

$$d_G(u, v) = \min_{w \in S} \{d_G(u, w) + d_G(w, v)\}$$

Lipton and Tarjan [82] proved that, for any rooted, undirected spanning tree T in a triangulated planar graph G, there exists a non-tree arc e such that the corresponding fundamental cycle in $T \cup \{e\}$ is a balanced separator of G, in which every component has size at most $2/3$. In particular, the nodes of this separator comprise two root-paths in T from the endpoints of the chosen non-tree arc. Thorup [123] proved later that this construction can be extended to selecting three root-paths, whose node set determines a balanced separator each component of whose has at most half the size of G. We call these spanning-tree-based separators, *separating paths*. An alternative separator, based on separating cycles (rather than separating paths) has been provided by Miller [87].

The generic idea behind the distance oracles is the following: If for every node $v \in V(G)$ we create a set $S(v)$ of separating paths / cycles, then for any pair of nodes $u, v \in V(G)$, it will certainly hold that every $uv-$path has to cross via nodes from $S(u) \cap S(v)$. If we have all the distances of a node v from/to its separating paths set $S(v)$, then a shortest $uv-$path can be restricted to paths that intersect $S(u) \cap S(v)$, whose partial distances are already known. A distance oracle typically creates not just one, but a collection of separating paths w.r.t. any particular node $v \in V(G)$. These sets are constructed recursively, producing consecutive refinements of existing decompositions of the graph, by separating the existing pieces. For every pair of nodes $u, v \in V(G)$, either these two nodes belong to the same part of the finest decomposition, in which case a direct shortest path calculation is possible due to the small size of the piece containing both of them, or they belong to different pieces and indeed any shortest $uv-$path must cross a small number (typically, logarithmic) of separators for moving from the origin to the destination. But the distance of such a path can be done exhaustively exploiting the precomputed distances of nodes belonging to these separators.

Recursive Divisions of Planar Graphs. We now review two main tools in providing distance oracles for planar graphs, namely the *recursive decomposition* and the *$r-$decomposition* of a planar graph $G = (V, E)$ with $|V| = n$ nodes. Every arc-induced subgraph $P = (V_P, E_P)$ by the subset $E_P \subseteq E$ is called a *piece* of G. The size $|P|$ of P is the number of its nodes. The nodes of V_P that are incident to nodes of $V \setminus V_P$ are called *boundary nodes* for P, and the entire set of boundary nodes for P is denoted by ∂P. In every piece P, that is also a planar graph, a face f that was *not* a face in G, is called a *hole* of P.

In order to construct a (binary) *recursive decomposition* of G, we consider G as a single piece (at level 0 of the recursive decomposition). At each level, using any mechanism for producing balanced separators, every piece P is split into two pieces P_1, P_2, each with at most $2|P|/3$ nodes, and at most $2|\partial P|/3 + O(\sqrt{r})$ boundary nodes. It is required that boundary nodes of P are also boundary nodes of the sub-pieces containing them. The recursive decomposition ends with pieces containing only one arc each. For any integer $r \in [n]$, that may depend on n, an *$r-$decomposition*, or *$r-$division* of G is a decomposition of G into $O(n/r)$ arc-disjoint pieces, each containing $O(r)$ nodes and $O(\sqrt{r})$ boundary nodes.

Frederickson [55] proved that it is possible to create an $r-$decomposition of a planar graph in time $O(n \log(n))$ using space $O(n)$ by recursively applying Lipton and Tarjan's algorithm for finding separating paths. An alternative would be to use Miller's algorithm for separating cycles, that allows for decompositions with a small number of holes per piece. For example, Fakharoenphol and Rao [51] demonstrated the construction of a recursive decomposition of G into pieces, such that every piece is connected, and has a constant number of holes, in time $O(n \log(n))$ using space $O(n \log(n))$. A similar approach was adopted in other approaches (e.g., [32]).

Having constructed a decomposition of the graph that assures a relatively small number of boundary nodes, a typical approach for an *exact* distance oracle is then to consider a *dense distance graph*, consisting of all the boundary nodes in the decomposition of G. The subset ∂P of boundary nodes for a particular piece P induces a complete subgraph in DDG with arc lengths equal to the actual distances of the nodes in the subgraph of G induced by P. A *multiple-source shortest paths* (MSSSP) algorithm [75] is then exploited for the computation of the actual distances among boundary nodes in G. If a recursive decomposition is considered, then every elementary piece contains a single edge. Therefore, the distances in DDG equal their distances in the original graph. The crucial observation at this point, is that in order to compute distances in DDG, it is not necessary to check the entire graph. It actually suffices to consider only edges corresponding to shortest paths between boundary nodes in a quite small (at most logarithmic) number of pieces. It is those pieces containing the two nodes under consideration, as well as their siblings in the tree of the recursive decomposition. There are only $O(\log(n))$ such pieces, with a total of $O(\sqrt{n})$ boundary nodes. As an example, exploiting this observation [51] managed to implement Dijkstra's algorithm that runs over a small subgraph of DDG, in time $O(\sqrt{n}\log^2(n))$ and space $O(n\log(n))$, with preprocessing time $O(n\log^2(n))$.

As for approximate distance oracles, rather than having the DDG that preserves the actual graph distances, the data structure now maintains a small set of connections (per separating set) that represent connecting paths from arbitrary nodes to nodes of a separating set, in such a way that for every shortest path that crosses a separating set there exists a good approximation that is the concatenation of connecting paths maintained in the data structure. For example, Thorup [123] proposed an algorithm that, given a planar graph and a shortest path p (which is actually a separating path), computes a set C of connecting paths that provides $(1+\epsilon)-$approximate paths for all the shortest paths in G crossing it, and has only $O(1/\epsilon)$ connections per node.

State-of-Art Review of Distance Oracles

This subsection provides a brief overview of the line of research concerning either exact or approximate distance oracles, mainly for sparse directed graphs that are of interest in road networks. Most of the actually appealing results concern planar graphs, but many of them can be almost directly transformed into equivalent results for bounded-genus graphs. Table 1 sums up the mentioned state-of-art results.

Exact Distance Oracles. Frederickson [55] was the first to provide an exact distance oracle that is efficient for *planar graphs* $G=(V,E)$ with $n=|V|$ nodes. In particular, he provided a data structure of linear size, $S\in O(n)$, preprocessing time $O(n\log(n))$ and linear query time $O(n)$. Later, Henzinger et al. [66] provided an $O(n)$-time algorithm for the direct computation of a shortest path tree rooted at an arbitrary node of a planar graph, yielding the preprocessing time actually obsolete, and also reducing the required space to $O(n)$, if the query time is allowed to be linear in the graph size.

The first attempt to provide a trade-off between space (and preprocessing time) and query time, was by Djidjev [48]. In particular, he proposed three different data structures, for various ranges of the required space: For $S\in[n,n^{3/2}]$ he proposed a data structure of size $O(S)$ with preprocessing time $O(n\sqrt{S})$ and query time $O(n^2/S)$. For $S\in[n^{3/2},n2]$ he proposed a data structure of size $O(S)$ with preprocessing time $O(S)$ and query time $O(n^2/S)$. Finally, for $S\in[n^{4/3},n^{3/2}]$ he proposed a data structure of size $O(S)$ with preprocessing time $O(n\sqrt{S})$ and query time $O(n\log(n/\sqrt{S})/\sqrt{S})$.

Fakharoenphol and Rao [51] provided a data structure requiring $S\in O(n\log(n))$ space and $O(n\log^3(n))$ preprocessing time, that supports query time $O(\sqrt{n}\log^2(n))$. Klein [75] later improved the preprocessing time of this oracle to $O(n\log^2(n))$. By means of space vs. query time trade-off, this is actually the best result to date.

Cabello [32] provided a data structure that requires $O(S)$ space and preprocessing time $O(S)$, which answers in time $O(n\log^{3/2}(n)/\sqrt{S})$ an arbitrary query, for any $S\in[n^{4/3}\log^{1/3}(n),n^2]$.

Compared to Djidjev's oracle, Cabello's oracle has slower query time (by a root-logarithmic factor) but a larger range for the allowable space S. It is worth mentioning that the structure is designed so that it answers, not just one, but k queries, in time $O(n^{4/3}\log^{1/3}(n) + k^{2/3}n^{2/3}\log(n))$. If these k queries are known in advance, then the required space can be reduced down to $O(n+k)$.

Moses and Sommer [90] recently proposed an exact oracle of size $O(S)$ in preprocessing time $O(S\log^3(n)/\log\log(n))$ that supports arbitrary queries in time $O(n\log^2(n)\log^{3/2}(\log(n))/\sqrt{S})$, for any $S \in [n\log\log(n), n^2]$. That is, they assure roughly any desired space at the cost of sacrificing another root-logarithmic factor (compared to Djidjev's result).

Approximate Distance Oracles. Zwick [131] showed how to compute all-pairs approximate distances in directed graphs, with stretch $1+\epsilon$, in time $\tilde{O}(n^{2.376})$.

In the seminal work of Thorup and Zwick [124], for weighted undirected graphs with n nodes and m arcs, and any integer $k \in \mathbb{N}$, a data structure of size $O(k \cdot n^{1+1/k})$ is constructed in time $O(k \cdot m \cdot n^{1/k})$ which has stretch $(2k-1, 0)$ and query time $O(k)$. For example, if $k = 2$ then the data structure is preprocessed in time $O(m \cdot n^{1/2})$ and allows for constant-time queries that provide approximate distances with $(3, 0)$ stretch. They also proved that any oracle assuring multiplicative stretch less than $2k+1$, would definitely need $\Omega(n^{1+1/k})$ bits of space. This implies asymptotic optimality of their data structure for dense graphs, provided that the girth conjecture of Erdős is true[2].

Sommer et al. [120] recently provided a trade-off between space, (multiplicative) stretch and query time of approximate distance oracles for sparse graphs. In particular, they proved that any oracle that assures a stretch $(\alpha, 0)$ and responds to queries in time $O(t)$, must use space $n^{1+\Omega(1/(t\alpha))}$. E.g., for $m = O(n)$ if an approximate distance oracle assures stretch $(a, 0)$ and query time $O(1)$, then the required space is $n^{1+\Omega(1/c)}$.

Patrascu and Roditty [103] provided approximate distance oracles of size $n^{5/3}$ and constant query time, for general graphs with stretch $(2, 1)$. Their main trick was to convert the oracle of [124] with space requirement $O(n^{3/2})$ and multiplicative stretch 3, into an oracle with stretch $(2, 1)$. Indeed, exploiting their analysis of the general case, along with a simple trick (that makes all candidate paths of having even number edges) the authors also managed to present a distance oracle of size $n^{5/3}$ and stretch $(2, 0)$, for graphs with $m = O(n)$ edges. And if one insists on constant-query time, then this stretch factor seems to be unavoidable, unless the required space becomes $\Omega(n^2)$. This is because a distance oracle with constant time query and multiplicative stretch < 2 solves the *set intersection* problem. If non-constant query time is also allowed, then [107] provide a data structure of size $O(n \cdot m^{1-\epsilon/(4+2\epsilon)})$ that returns a $(1+\epsilon)-$approximate distance in time $\tilde{O}(m^{1-\epsilon/(4+2\epsilon)})$, for *unweighted undirected* graphs. Additionally, the required path is determined in time proportional to the path length.

Nevertheless, what if the graph has $O(n)$ edges? How about allowing also non-zero additive stretches? These two questions are closely related. For example, any approximate distance oracle with $(\alpha, b)-$stretch for general graphs is implied by the existence of an approximate distance oracle with multiplicative stretch a for graphs with $O(n)$ edges: One has only to divide each edge of the original (possibly dense) graph to $b+1$ edges. Thorup [123] introduced approximate distance oracles for planar graphs. For example, he proved that for any $\epsilon > 0$ and any planar undirected graph on n nodes, there is an approximate distance oracle with stretch $(1+\epsilon, 0)$, using space $O(n\epsilon^{-1}\log(n))$, that assures query response times $O(\epsilon^{-1})$.

[2]The Erdős conjecture states that there exist $n-$node (undirected) graphs with $\Omega(n^{1+1/k})$ arcs and girth $\geq 2k+2$.

REF	GRAPH	SPACE	PREPROCESSING	QUERY	STRETCH	COMMENT
[55]	planar	$O(n)$	$O(n \log(n))$	$O(n)$	1	QUERY computes an SPT
[66]	planar	$O(n)$	–	$O(n)$	1	linear time SPT algorithm
[48]	planar	$O(S)$	$O(S)$	$O(n^2/S)$	1	$S \in [n^{3/2}, n^2]$
		$O(S)$	$O(n\sqrt{S})$	$O(n^2/S)$	1	$S \in [n, n^{3/2}]$
		$O(S)$	$O(n\sqrt{S})$	$O(n \log(n/\sqrt{S})/\sqrt{S})$	1	$S \in [n^{4/3}, n^2]$
[51]	planar	$O(n \log(n))$	$O(n \log^3(n))$	$O(\sqrt{n} \log^2(n))$	1	
[75]	planar	$O(n \log(n))$	$O(n \log^2(n))$	$O(\sqrt{n} \log^2(n))$	1	Improvement of PRE in [51]
[32]	planar	$O(S)$	$O(S)$	$O(n \log^{3/2}(n)/\sqrt{S})$	1	$S \in [n^{4/3} \log^{1/3}(n), n^2]$
		$O(S)$	$O(S)$	$O\left(\frac{n}{n^{4/3}\log^{1/3}(n)} + \frac{k^{2/3}n^{2/3}\log(n)}{n^{4/3}\log^{1/3}(n)}\right)$	1	for k unknown queries
		$O(n+k)$	$O(S)$	$O\left(\frac{n}{n^{4/3}\log^{1/3}(n)} + \frac{k^{2/3}n^{2/3}\log(n)}{n^{4/3}\log^{1/3}(n)}\right)$	1	for k known queries
[90]	planar	$O(S)$	$O(S \log^3(n)/\log\log(n))$	$O(n \log^2(n) \log^{3/2}(\log(n))/\sqrt{S})$	1	$S \in [n \log\log(n), n^2]$
		$O(n)$	$O(n \log(n))$	$O(n^{1/2+\epsilon})$	1	for any constant $\epsilon > 0$
		$O(n \log(n) \log\log(n))$	$O(n^{1+\epsilon})$	$O\left(\min\left\{\begin{array}{c}\ell \log^2(\ell) \log\log(\ell) \\ \sqrt{n}\log^2(n)\end{array}\right\}\right)$	1	for constant $\epsilon > 0$, min distance ≥ 1 and nodes at distance ℓ
[131]	any	$O(n^2)$	$\tilde{O}\left(\frac{n^\omega}{\epsilon} \cdot \log\left(\frac{W}{\epsilon}\right)\right)$	$O(1)$	$(1+\epsilon, 0)$	W is largest arc weight, ω is the exponent of matrix-multiplication complexity
[123]	planar	$O(n \log^2(n)/\epsilon)$	$O(n \log^4(n)/\epsilon^2)$	$O(\log\log(n) + 1/\epsilon)$	$(1+\epsilon, 0)$	directed
		$O(n \log(n)/\epsilon)$	$O(n \log^3(n)/\epsilon^2)$	$O(1/\epsilon)$	$(1+\epsilon, 0)$	undirected
[74]	planar	$O(n)$	$O(n \log^2(n))$	$O(\log^2(n)/\epsilon^2)$	$(1+\epsilon, 0)$	undirected, in ICALP version
		$O(n)$	$O(n \log(nN) \log^3(n)/\epsilon^2)$	$O(\log^2(n) \log^2(nN)/\epsilon^2)$	$(1+\epsilon, 0)$	directed, N is largest (integer) weight, in xArchiv version
	genus g	$O(n)$	$O(n \log(n)[g^3 + \log(n)])$	$O([\log(n)+g]^2/\epsilon^2)$	$(1+\epsilon, 0)$	in ICALP version
[107]	undirected	$O(nm^{1-\epsilon/(4+2\epsilon)})$	$O(mn)$	$\tilde{O}(m^{1-\epsilon/(4+2\epsilon)})$	$(1+\epsilon, 0)$	unweighted graphs
[103]	undirected	$O(n^{5/3})$		$O(1)$	$(2, 1)$	unweighted graphs
		$O(n^{5/3})$		$O(1)$	$(2, 0)$	unweighted sparse graphs
[124]	any	$O(kn^{1+1/k})$	$O(kmn^{1/k})$	$O(k)$	$2k-1$	
[15]	any	$[\epsilon^{-O(\lambda)} + 2^{O\lambda\log(\lambda)}]n$	$[2^{O(\lambda)}\log^3(n) + \epsilon^{-O(\lambda)} + 2^{O\lambda\log(\lambda)}]n$	$O(1)$	$(1+\epsilon, 0)$	metric spaces of doubling dimension λ, approximate preprocessing time

Figure 1: Summary of state-of-art (exact / approximate) distance-oracles.

2.3.2 Preprocessing Techniques for Road Networks

Research on route-planning algorithms in transportation networks has undergone a rapid development over the last years. The common approach is to model the network as a weighted directed graph $G = (V, E)$ and apply graph-based search algorithms. For static road networks, nodes model intersections and edges depict street segments. Typically, edge weights are set according to travel-time spent on the respective street segment. While Dijkstra's algorithm [47] can be used to compute a best route between a given source and target $s, t \in V$ in almost linear time [59], it is too slow for practical applications on huge datasets. Real-world transportation networks commonly consist of several million nodes and edges, and Dijkstra's algorithm explores a large fraction of the network on average because it first discovers all nodes u that are closer to s than t in the chosen edge metric. For many applications such as route planning websites or solvers for real-world vehicle routing problems Dijkstra's algorithm is just too slow. Thus, over the years a multitude of speedup techniques for Dijkstra's algorithm were developed, all following a similar paradigm: In a *preprocessing phase* auxiliary data is computed which is then used to accelerate Dijkstra's algorithm in the *query phase*.

Classical speedup techniques are *bidirectional search* and *goal-directed search* [65, 105]. Bidirectional search starts two (Dijkstra) searches, one at the source s looking at forward-directed edges, one at the target t looking at backward-directed edges. Whenever the two searches meet a tentative distance μ is updated, and when this distance μ cannot possibly be improved the search can stop. Bidirectional search typically achieves a speedup over Dijkstra's algorithm of less than factor two on road networks (think of two balls of half the radius). Goal-directed search changes the order in which nodes are discovered by Dijkstra's algorithm by adding a potential function $p : V \to \mathbb{R}$ on the nodes and changing the edge weights accordingly. The reduced costs of an edge $e = (u, v)$ are defined as $\text{len}_p(e) = \text{len}(e) - p(u) + p(v)$. A potential function can successfully guide the search if the reduced costs of the edges on the shortest s–t path are very low while the reduced costs of other edges are high. A potential function is valid iff the reduced costs of all edges are positive. A classic yet not very good choice for the potential function is the euclidean distance between two nodes divided by an upper limit on the travel speed. Speedups over Dijkstra's algorithm are less than impressive.

The last twelve years, more sophisticated techniques were presented, starting from *geometric containers* [114, 115, 126] and *multilevel search* [115, 116] to *reach* [63], *landmarks* [60], *arc-flags* [79, 77, 88, 67, 68], *multilevel overlay graphs* [69], and *highway hierarchies* [111, 112] and *contraction hierarchies* [57, 58], respectively combinations of those [70, 23, 61, 24]. Lately, the rapid development of preprocessing techniques has culminated in extremely fast shortest path queries on static, time-independent road networks. Transit-node routing [18, 17] and the more refined hub-based labeling algorithm [7] achieve speedups of up to six orders of magnitude over Dijkstra's algorithm, requiring only a few memory accesses to answer a single query on average. However, it should be noted that the fastest techniques developed for static road networks heavily rely on structural properties of road networks with respect to the travel-time metric [8, 5] and their performance degrades significantly on other networks [16, 24].

In the following, we give an overview on some of the aforementioned techniques. See [43, 39] for a more detailed discussion of static road preprocessing techniques.

Landmarks [60, 62] is an improved goal-directed search with a better potential function which exploits the triangle inequality between a selection of landmarks in order to prune the search space of the query. Since A*, landmarks and the triangle inequality are used by the algorithm it is also known as ALT. The preprocessing phase of ALT determines the distance of every node to the set of landmarks. Placing of landmarks plays an important role.

Arc-flags [79, 77, 88, 67, 68] on the other hand guide the search by annotating every edge with information about whether it is important for the current target node, i. e. whether it can possibly lie on a shortest path to the target. Edges that are not important are not relaxed during search, significantly reducing the search space. In order to preprocess the annotated edge data, the graph is partitioned into a set of cells $\mathcal{C}$. From the boundary nodes of each cell a forest of backward

shortest-path trees is grown. For each edge e a flag $AF_C(e)$ for every cell $C \in \mathcal{C}$ is set to `true` or `false` according to whether it lies on a tree edge to that cell or not (non-unique shortest paths can be handled easily). Only edges having their flag set to `true` for the cell containing the target node are discovered during the search. Note that arc-flags of non-important edges may be set to `true` without sacrificing correctness but important edges may never be set to `false`. This insight is used by several extensions to arc-flag routing in order to improve preprocessing at cost of query performance.

Another ingredient common to several techniques are *shortcuts* [112]. Shortcuts are edges inserted into the graph that represent a path of the original graph. The rationale for shortcuts is that evaluating shortcuts instead of each edge they represent on its own reduces the search space (significantly for long shortcuts). For *node contraction* a node u is taken from the search graph by inserting shortcuts between its neighbors. These shortcuts preserve all shortest paths via u. Edges that do not contribute to shortest paths are removed in a *edge reduction* step. The subgraph of contracted nodes is called *component*, the non-contracted nodes form the *core* of the graph. All important shortest paths lie in the core, and shortest path queries from any s to t can mostly be constrained to the core: First, a small forward and backward search is run from s and t, respectively, until all paths are covered by core nodes. Then, a multi-source multi-target search is run between these core nodes constrained to core edges. Multi-source multi-target search is a variant of bidirectional search where not only s and t are initialized with zero (and all other nodes with ∞) but where several nodes are initialized with (arbitrary) starting distances (and all other nodes with ∞).

Contraction hierarchies [57, 58] (CH) is a technique solely based on node contraction. Nodes $u \in V$ are ordered according to an importance heuristic $\text{rank}(u)$ and contracted in ascending order. Shortcuts are inserted between neighbors w, v of a contracted node u if necessary, that is, as long as no other equally long or shorter path exists from w to v in the subgraph $G \backslash u$. A local *witness search* is run for each candidate shortcut to determine its necessity. Note that there is a trade-off between the locality and fastness of the witness search and the number of unnecessary shortcuts inserted. Further note that unnecessary shortcuts do not lead to incorrect results but only to slower query speeds. Often, witness search is restricted to at most ten hops. The node ordering is computed online by a set of heuristics which prioritize nodes according to the added shortcuts they will yield if contracted, the deleted edges, the search space of the respective witness searches after the node's contraction, and the increase in hierarchy height their contraction will result in, among others.

The CH query is a bidirectional Dijkstra search from s and t operating on G, augmented by the shortcuts computed during preprocessing. The forward search only relaxes edges $e = (u, v)$ going upwards, i. e. with $\text{rank}(u) \leq \text{rank}(v)$. The backward search works accordingly only on edges where $\text{rank}(u) \geq \text{rank}(v)$. When both searches meet a tentative distance measure μ is updated and the path with the least μ is shortest. Each search can stop as soon as the minimum element in its priority queue is greater than μ. The search space of a CH query directly correlates to the height of the precomputed hierarchy. On average less than 500 nodes need to be explored to determine the distance of a random shortest-path on a continental-sized road network [57].

Although the preprocessing phase of most of these more sophisticated techniques is heuristic, the query phase is provably exact and returns the same results as Dijkstra's algorithm but quicker. For many preprocessing techniques it has been shown that obtaining optimal results is NP-hard [22, 21] which justifies the use of heuristics.

Since point-to-point queries on static road networks are now so fast that the problem can be considered solved, research has turned to more involved (and realistic) road scenarios in the last couple of years: metrics other than travel-time (e. g. distance, turn-costs, road toll, height restrictions, ...), dynamic updates to the metric (e. g. traffic incidents), time-dependency (i. e., having edge weights that change as a function of the time of day), and optimization of multiple criteria at once. For details on multi-criteria or time-dependent route planning, see Sections 2.3.3 and 2.4, respectively. Roughly speaking, the faster queries a speedup technique enables the more

expensive is its preprocessing phase. When considering frequent changes in the metric this can be a problem.

While most preprocessing techniques have focused on exploiting the hierarchy inherent in road networks with travel-time metric, recently, a preprocessing technique has been devised [37] that is more robust against the choice of metric, allows for quickly changing the whole metric (e. g. switching from travel-time to distance weights) and enables very fast local metric updates (e. g. for traffic incidents). While the core algorithmic ideas are not new [69, 40], the main breakthrough was made possible by the realization that road networks do not only possess a pronounced hierarchy with respect to the travel-time metric but also a pronounced topological structure [38]: the presence of many natural cuts (such as bridges, mountain passes, and ferries), that is, relatively sparse cuts that separate a denser local region from the rest of the network (e. g. think of a city that has only a few outbound roads compared to thousands of inner city roads).

The main idea of Customizable Route Planning (CRP) [37] is to split the preprocessing phase into two phases: In the *partitioning phase* a multi-level balanced partition with small cuts is computed [38]: each cell has a relatively small number of *border nodes* (i. e., nodes incident to *cut edges*), and border nodes of a certain level i are a subset of the border nodes at lower level $j < i$. In the *metric customization phase* a multi-level overlay graph [69] is build based on the chosen metric and the computed partition: for each cell a full clique of its border nodes is calculated such that it is preserving shortest path distances through that cell (*overlay edges*). In the *query phase* a multi-level variant of bidirectional Dijkstra' algorithm is applied to the original graph G annotated with all overlay edges computed during preprocessing: As usual the tentative distance μ is set to infinity. From the source s the forward search explores the original graph restricted to the cell containing s until all paths are covered by border nodes. From there on, only overlay edges of the first level and cut edges are relaxed until all paths are covered by first level border nodes. That way the search is coarsened until a level is reached where both s and t lie in the same cell. A backward search is run in the same fashion from the target t in parallel. Whenever the searches meet the tentative distance μ is updated. The path in the annotated graph that minimizes μ is shortest. The respective path in the original path can be constructed by unpacking the shortcut edges. The standard bidirectional stopping criterion applies.

Carefully engineering the implementation and its data structures, the authors of [37] achieve point-to-point queries that run in under a millisecond on a continental sized network. Experiments show that query performance is almost independent of the metric used. Metric customization takes only about five seconds and the amount of annotated data generated is rather small with only about 33 MiB independent of the metric used. Local updates can be done in just a few milliseconds: only the cliques of the cells containing the updated edges need to be recalculated. This enables quick integration of live traffic data feeds.

2.3.3 Multi-Criteria Route Planning

We have learned so far that in route planning the goal is to find a shortest path between a source node s and a target node t using a certain weight function associated with the edges of the graph that models the transportation network. Typically the average travel-time spent on an road segment is used as its edge weight. But sometimes it is not enough to know just the fastest route—or the most direct or cheapest route. In many cases the weight function realistically depends on multiple criteria such as the travel time or the energy cost, and there often is a trade-off between these criteria. For instance less travel time can mean more energy cost, but in eCOMPASS we are interested in computing environmentally friendly routes that are not too slow to be rejected by the user. There are two common approaches to make this possible. The first is to optimize a mixed linear goal function $f(\alpha_1, \ldots, \alpha_k) = \alpha_1 \cdot c_1 + \cdots + \alpha_k \cdot c_k$ expressed by the weighted sum of the considered criteria $c_1, \ldots, c_k$. In the context of route planning this boils down to setting each edge weight of the routing graph to the weighted sum of the costs it induces, then running a generalization of Dijkstra's algorithm parametrized by source node s, target node t and proportion

vector $\alpha = (\alpha_1, \ldots, \alpha_k)$.

The second approach is to compute all Pareto-optima. A solution is Pareto-optimal if it is not dominated by any other solution. Given two journeys p and q, we say that p *dominates* q if and only if there is at least one criterion for which p has a better value than q and there is no criterion for which p has a worse value than q. In the context of route planning Pareto-optimal paths can be found by a generalization of Dijkstra's algorithm [64, 86, 122]: instead of a single label each node u holds a Pareto-set S_u of optimal, non-dominated labels. The set at the source s is initialized with the label $(0, \ldots, 0)$, all other sets are initialized empty. Similar to Dijkstra's algorithm a priority queue keeps all unsettled labels in order, here lexicographically. The main step of the algorithm extracts the minimum label L_u from the queue and relaxes all edges $e = (u, v)$ to obtain new labels $L_v = \text{len}(e) + L_u$. For each such new label L_v the algorithm checks whether L_v is dominated by S_v. If not, it is added to the set S_v of Pareto-optima at node v. If labels in S_v are dominated by L_v they get removed as well.

While multi-criteria search using a mixed goal function only returns one result, multi-criteria Pareto-search usually computes several solutions (how many depends on the network and the metric [91]). Hence, a mixed multi-criteria search can be much faster. But if presenting several solutions to chose from to the user is the goal, Pareto-search can be beneficial: in general, for mixed multi-criteria search it is unclear a priori for what values of α paths change. That can make it hard to predict the number of runs necessary to compute a given number of solutions. However, we later on review a technique called Flex-CH [56] that does not have this problem because it precomputes such information, albeit for an easier variant of a mixed goal function. Still, a mixed goal function can cause problems if one of the criteria is dependent on another criterion (as with e. g. travel-time and time-dependent costs).

Each of these two approaches to multi-criteria route planning imposes challenges when trying to adapt static speedup techniques. The preprocessed data has to regard all possible configurations. In the following we review two such adaptations: one following the multi-criteria Pareto-approach, the other following the mixed multi-criteria search approach although only for two criteria.

Pareto Paths with SHARC. Pareto-SHARC [44] is a technique to speedup the computation of Pareto-optimal paths in road networks. It is an adaptation of the preprocessing and query of SHARC [23] which in turn is based on the combination of shortcuts [112] and multi-level arc-flags [88]. For details on these "ingredients" see Section 2.3.2 or the respective publications. In the following we first quickly review SHARC and then continue with the discussion of Pareto-SHARC.

The main idea of SHARC is to combine the computation of multi-level arc-flags with node contraction in order to achieve faster precomputation and better query guidance in the target cell. During a contraction step all contracted edges (i. e. edges incident to contracted nodes) get their arc-flags assigned automatically: edges leading into the component get only their own cell's flag set, while edges leading out of the component get all flags set. While this might be sub-optimal, it only affects the beginning and end of each search but allows for much faster preprocessing: only arc-flags for the added shortcuts and non-contracted edges must be computed normally. The preprocessing of SHARC runs in three phases: First, all nodes of degree two or less are contracted. On the remaining graph a multi-level partition is computed. Let C_j^i be the j^{th} cell at level i. Let $C^i(u)$ be the cell on level i containing u. Second, contraction is followed by arc-flag computation iteratively: on level i nodes in all C_j^i are contracted and edges reduced until further contraction would yield too many shortcuts. On the remaining core, arc-flags for level i are computed by growing partial (i. e. restricted to the core) forest of shortest-path trees from each cell C_j^i. Third, arc-flags in the component are refined and nodes of degree two or less are reattached. Arc-flag refinement is a process of propagating flags on shortcuts towards contracted edges (which have been set to all `true` before). Several other improvements such as partial refinement, partial arc-flag computation, or removal of shortcuts exist [23].

The query of SHARC is a multi-level arc-flag variant of Dijkstra's algorithm: whenever a node

u is extracted from the priority queue, only those of its outgoing edges are relaxed that target the cell $C^i(t)$, where i is the lowest common level of u and t.

These aforementioned ingredients are adapted for Pareto-SHARC in the following way: the multi-level partition used for arc-flag computation is metric independent and can be used as is. A multi-criteria arc-flag $AF_C(e)$ is set to `true` if the edge e is important for at least one Pareto-path towards the cell C. In order to decide this, partial Pareto-path graphs are grown from the border nodes of all C^i_j of level i. The arc-flag $AF_{C^i_j}(e)$ is set to `true` if the edge e partakes in any of these Pareto-paths. As shortcuts only represent paths, node contraction is easily adapted as long as addition is well-defined on the multi-criteria metric. Automatic setting of flags of contracted edges is the same as for uni-criteria SHARC. However, because two paths between the same nodes might not dominate each other, node contraction can lead to the insertion of parallel edges making preprocessing more expensive. Arc-flag refinement is bit more complicated in the multi-criteria case: instead of shortest path trees Pareto-path graphs need to be propagated. In order to limit preprocessing time, only a limited number of priority queue extractions is permitted. Adaptation of edge reduction is straight forward as well: a local multi-criteria witness search is run to determine all Pareto-optimal (local) distances; an edge is unnecessary if its label is dominated by at least one label of the optimal Pareto-set. This routine is usually constrained by an upper bound on the number of priority queue extraction in order to limit computation time. The query of Pareto-SHARC is a generalization of the SHARC-query that maintains Pareto-sets of labels at each nodes and checks for dominance when creating new labels during edge relaxation.

Experiments in [23] show that the runtime of the Pareto-SHARC query increases with the number of target labels (i. e. results) but the overhead when compared to SHARC is more than that number. Rather it seems that the overhead grows super-linear in the number of target labels. Because some combinations of metrics yield very high number of results, the authors propose two measures to reduce their size during preprocessing and query. The first is to determine a low upper limit on acceptable values in the first metric, e. g. to constrain the total travel-time to $1 + \epsilon$ times the shortest path, with ϵ chosen between zero and five percent on one of the tested instances. As the choice of ϵ is rather limited, the authors propose a second measure called pricing: an increase in the first metric (i. e. travel-time) is only allowed if it is accompanied by a *considerable* decrease in the other metrics parametrized by γ. Depending on the choice of *epsilon* and *gamma* reasonably fast multi-criteria queries are achieved on a continental sized road network yielding several Pareto-optimal results each.

Route Planning with Flexible Objective Functions. The approach portrayed in the previous paragraph solved the multi-criteria shortest path problem by finding all Pareto-optimal solutions, but the computational overhead can be considerably large [64, 86]. In this section we discuss a different approach presented in [56], which assumes that the edge weight function is a linear combination of both criteria and is formulated by $w_p(e) := t(e) + p \cdot c(e)$, where $t(e)$ and $c(e)$ are the two nonnegative edge weight functions, e. g. travel time and energy cost, and p is the factor of the trade-off. In this way, the user can choose the parameter p at query time, making the algorithm *flexible*. More formally, the precomputation procedure of the algorithm receives as input a directed graph G, two edge weight functions $t(e)$ and $c(e)$, and a discrete interval $[L, U]$ bounding the parameter p. The resulting flexible query algorithm finds for each parameter $p \in [L, U]$ an exact shortest path for the edge metric $t(e) + p \cdot c(e)$.

One straightforward approach to solve the above problem is to use a variant of Dijkstra's algorithm that evaluates each edge based on the parameter p, but in practice this is very slow for large networks. See Section 2.3.2 for details. Instead, the authors of [56] (called Flex-CH from here on) adapt a combination of Contraction Hierarchies [57, 58] (CH) and Core-based ALT [24] (CALT) to this flexible route planning problem. In this context CALT is essentially the ALT algorithm applied to a small *core* of the original graph produced by (partial) CH. For details on ALT and CH also see Section 2.3.2 or the respective publications. In the following we discuss the adaptations to the

original preprocessing and query phases necessary to make them flexible.

In the preprocessing phase based on *node contraction* a node u is contracted by removing it and connecting its neighbors by *shortcut* edges. In classic CH a procedure known as *witness search* is run in order to decide whether the shortcut is really needed; a *witness* is a path not using u that is not longer than the candidate shortcut. Hence a witness allows to omit the candidate shortcut. Here, such a witness search needs to be run for every integer parameter $p \in [L, U]$. But, if the cardinality of $[L, U]$ is large, too many witness searches are required, and eventually too many shortcuts are inserted for the whole parameter interval.

Instead, the authors of Flex-CH propose to find subintervals of $[L, U]$ where a shortcut is always needed, and subintervals where there is always an alternative witness path alleviating the need of a shortcut. In case a shortcut edge is necessary for a subinterval of the parameter values, the edge is associated with this subinterval which is called the minimal necessity interval $NI(e)$. In some cases a large number of queries is needed to find the minimal necessity interval, so searches are stopped as soon as they reach an arbitrary large cardinality. In addition, during a witness search, the algorithm checks the necessity intervals of edge graphs too, and limits them to the smallest interval necessary.

Another ingredient that needs adaptation is the order in which the nodes are contracted. Experiments done by the authors of Flex-CH show that a single node order does not work well in practice, resulting in the insertion of too many shortcuts. As before, the order of the nodes is re-evaluated after each contraction step. However, the priorization of a node based on a single value of the parameter p may differ a lot from that based on a different value of p. In addition, if computations are made on the whole parameter interval, lots of shortcuts have to be added each time a node is contracted, in order to keep generality. On the contrary, if the interval is split, the problem is divided into two sufficiently different subproblems which can be solved more efficiently. So, at a certain point in the contraction process, when lots of shortcuts would be added only for a limited subinterval, the intervals are split and the preprocessing works independently on both instances from then onward.

Despite interval splitting, the hierarchy tends to degrade when contracting the last top level nodes, and a rapidly increasing number of shortcuts needs to be added between the yet uncontracted nodes. As an result, preprocessing does not terminate within reasonable time. In order to improve on this situation the authors propose to stop the contraction and to apply a different speedup technique on the remaining *core* graph of uncontracted nodes, which is quite smaller than the original graph. They apply CALT [24] which is a core-based ALT where landmarks are only computed for core nodes. Accordingly, the respective preprocessing steps of CALT are incorporated into the preprocessing of Flex-CH, with the necessary adaptations for the parameter p.

The last ingredient which needs to be adapted is the query algorithm. The query phase of Flex-CH follows the classic CH query with small modifications. Given a parameter p, the query relaxes only those edges e that are necessary for this parameter, that means, p resides in the necessity interval $NI(e)$. The classic CH query considers an upward and downward graph where edges point to more important nodes in the upward graph and to less important nodes to the downward graph. A forward and backward search are then executed which meet at the most important node in the shortest path. In this approach, since there exist different orderings of nodes according to different parameter intervals, there are edges which belong to the upward graph for a parameter p and to the downward graph for a different parameter p'. Such edges are split into several parallel edges, and have their interval adjusted. In this way, each edge only belongs to the upward or downward graph and the CH query works as expected. In case the graph is only contracted partially, the query is augmented by a secondary query using CALT as soon as both searches of the classic CH query reach the core graph.

Using the above techniques, Flex-CH is also capable of performing profile queries where it finds all possible shortest paths between a given source and target node for arbitrary values of p in the interval. In addition it can be slightly modified to answer profile queries with sampling, where the

algorithm performs queries on a subset of $[L, U]$ and faster approximated profile queries within a factor of $(1+\epsilon)$.

2.3.4 Vehicle Route Planning

In this section the Vehicle Routing Problem (VRP) is reviewed with its main variants. For each problem a mathematical definition is given and solution approaches are discussed. In all problems the aim is to distribute a number of goods between a depot and the final users (customers). One (or more) vehicle(s) that will deliver the goods are available.

Basic Models and Problem Complexity. In order to introduce formally each problem some basic notation must be presented. All problems are modeled as a graph $G = (V, E)$ where V is the node set, V_0 denotes the depot and the other $|V| - 1$ nodes are the customers, and E is the set of edges. In most cases, a road network is considered which will be used for the delivery of the goods. Each edge e_i connects two nodes and models an actual road segment. In each edge, a cost c_{ij} is used to denote the traveling cost alongside the edge. The cost is usually expressed as the distance or time between two nodes. The cost can also be expressed as the total length of the tour or as the number of vehicles needed to serve all customers. The goal is to find one or more (in case of many vehicles) tour (a path visiting each node at least once) that starts from the depot and ends at the depot that serves all customers and minimizes the total cost.

The VRP is an extension of the well known and studied Traveling Salesman Problem (TSP). It is proven that both TSP and VRP belong to the family of NP-Hard problems. This means that an exact solution is hard to compute as the number of computations needed grows exponentially with the input size. However, for small and medium problem instances an exact solution can be computed in a reasonable amount of time. For large instances, the approach is the use of heuristics that approximate the optimal solution. Solution approaches and techniques for the Vehicle Routing Problem and will be presented after all variants of VRP.

Capacitated Vehicle Routing Problem. The Capacitated Vehicle Routing Problem (CVRP) is the simplest an most studied variant. In CVRP the demands are deterministic and known in advance. The vehicles are homogeneous (they have the same characteristics), each vehicle has a maximum capacity Q and there is a single depot. The objective is to minimize the total cost to serve all customers and respect to the constraint that the capacity of the goods carried per vehicle must not exceed Q.

Vehicle Routing Problem with Time Windows. The Vehicle Routing Problem with Time Windows (VRPTW) is an extension of CVRP. Capacity constraints of vehicles are also present and each customer is associated with a time interval $[a, b]$ known as a time window. The additional constraint is that all customers must be serviced in their time windows. The exact time that vehicles leave the depot and travel time t_e for each edge are given in advance as input. There is also a given service time s_i for each customer. The goal is again to find a tour that minimizes the total cost that respects each vehicle's capacity Q and respects the time windows of each customer.

Vehicle Routing Problem with Backhauls. The Vehicle Routing Problem with Backhauls (VRPB) is another extension of CVRP. The difference here is that customers are partitioned in two groups. The first group consists of n customers, each of them expecting goods to be delivered from the depot. The second group consists of m customers from which a certain quantity of goods must be picked up. Obviously, the total number of customers is $N = n + m$. Notice that a precedence constraint exists. In a tour that serves both types of customers all deliveries must be made before pick ups.

Vehicle Routing Problem with Pickup and Delivery. The Vehicle Routing Problem with Pickup and Delivery (VRPPD) associates each customer i with two quantities d_i, p_i. They represent for each customer the demand that has to be delivered and picked up respectively. It is assumed that in each customer location the delivery is performed before the pick up. When the origin and the destination of demands are common the problem is called Vehicle Routing Problem with Simultaneous Pickup and Delivery (VRPSPD). The difference in these variants of the problem is that the vehicles must now deliver and pick up goods and not exceed their maximum capacity. Also, the position of the goods in the vehicle (e.g. truck) must be considered due to the priority of deliveries over pick ups.

Time Dependent Vehicle Routing Problem. The Time Dependent Vehicle Routing Problem (TDVRP) is an interesting variant. In the literature is closely related to the Dynamic or Real time Vehicle Routing Problem. In the Dynamic VRP modeling, one can consider customer orders that arise after the routes have initially been computed. In TDVRP, the model takes into account real-time information regarding the location of the vehicles and information on road conditions (traffic due to rush hours,accidents, bad weather ...). So there may be a need to recalculate the routes in the aspect of the new information available. These models are very close to real applications where traffic information is present and customer orders may arise after the initial computation of the routes.

These variants also belong to the NP-Hard class of problems, since a static VRP must be solved each time a new immediate request is received. An additional challenge is to find and use online algorithms since immediate requests are likely to take place.

Other Variants. Apart from all the above categories presented, there are also other variants of VPR. Most of these variants regard the constraints of the problem. One variant is if the fleet is homogeneous (all vehicles have the same specifications) or heterogeneous (vehicles have different capacities, dimensions, environmental consequences). In some applications different vehicles are required due to the road network. For example a big truck can not access a small road or must follow a specific route due to weight issues. Furthermore, vehicles that carry dangerous materials must not travel near urban cities. In all problems there is another additional constraint: drivers must not exceed the maximum number of hours that they can work due to fatigue reasons. For long trips rest areas for the drivers (parking spots, hotel areas) must be taken into consideration.

Another variant is that of multi depots. Instead of one depot in this variant there are many depots and there may exist constraints that certain goods must be delivered to certain depots. Stochastic VRP is also a common variant. In this case, the customer demand is not known until the vehicle arrives at a customer. Tours must be planned using a probability distribution of the demand at any costumer. There must also be a strategy present when the vehicle runs out of the commodity that is carrying before it serves all customers.

In some cases the vehicles are not required to return to the depot after visiting the final costumer. In this case tours are open paths. This scenario may appear if a fleet of vehicles is hired by another party to distribute the goods. This variant is known as Open Vehicle Routing Problem (OVRP). The Period Vehicle Routing Problem (PVRP) is a variant where all routes for all days of a T-day period must be computed. Again the objective is to minimize the total cost.

Solution Approaches. The most common approach to this kind of problems is to use linear programming techniques. Three different basic models have been proposed in the literature for VRP: vehicle flow formulation, commodity flow formulation and set partitioning formulation. Another approach is to use heuristics which in many real applications compute fast a solution that is close to the optimal one. Popular and extensively used heuristics are: tabu search and ant colony optimization.

The models of vehicle flow formulations use integer variables. They are associated with each edge of the graph and their purpose is to count the number of times each edge is traversed by a vehicle. Usually, binary variables x are used where x_{ij} takes the value 1 if edge $(i, j) \in E$ belongs to the optimal solution, 0 otherwise. This approach is suited when the cost of the solution can be expressed as the sum of the cost associated with each edge. One disadvantage of this model is that the linear programming relaxation can lead to weak solutions.

In the model of commodity flow formulation additional variables are associated with edges of the graph and represent the flow of commodities along the paths. An example of this model is the following: when a vehicle travels from node i to node j two non negative variables y_{ij} and y_{ji} are associated with each edge, denoting the vehicle's load and the residual capacity respectively. Also a graph transformation is necessary. An extended complete digraph $G' = (V', E')$ is created which contains an additional copy of the depot (now node V_{n+1}). Now feasible solutions are paths starting from node V_0 (original depot) and ending to node V_{n+1} (artificial depot). This leads to an integer formulation of the VRP [125].

In the set partitioning model the VRP is formulated as a Set Partitioning Problem (SPP). A collection of circuits (closed paths starting and ending at the depot visiting each node once) with a minimum cost must be determined. The main advantage of this method is that it allows general route costs, depending on the edges and on the vehicle type. However, models of this type require an exponential number of binary variables.

Tabu search belongs to the family of local search methods. The main idea of these methods is the following: take a potential solution and check solutions that are similar except for some details (neighbor solutions) and try to find an improved one. The problem is that local search methods tend to stuck in suboptimal regions. In Tabu search a set of rules is provided by the user. If a potential solution violates a rule it is marked as "taboo" so it will not be considered in the future. In [50] a tabu search algorithm was used to solve the CVRP. The algorithm used was developed in C and experiments were conducted using a Pentium M at 1.4GHz. Three different data sets were used for the experiments. The results are encouraging solving instances with: 27 vertices and 51 edges in 2.5 seconds, 50 vertices and 97 edges in 5.1 seconds and 140 vertices and 190 edges in 129.9 seconds.

Ant Colony Optimization (ACO) is a metaheuristic that is becoming more and more popular. The main idea of these types of algorithms is inspired by ants. Ants use pheromone trails to exchange information regarding shortest paths to food. An initial ant lays pheromone in different quantities in the ground. Another ant detects a previously laid pheromone and if it decides to follow the trail it reinforces it with its own. So the more pheromone the most attractive the path is for other ants. In [108] an ACO algorithm is used called Ant Colony System (ACS). It is used to solve a VRPTW. The problem was to distribute goods to 600 supermarket chains in Switzerland. There were 3 different types of vehicles available with different capacities: truck (17 pallets), trucks with trailers (35 pallets) and tractor units with semi-trailers (33 pallets). Data regarding road network was considered using digital maps. An additional constraint was that all routes must be performed in one day. The implementation of the ACS algorithm was compared to man-made tours. The algorithm managed to minimize the total number of routes from 2056 (man-made) to 1807. Also the algorithm minimized the total distance covered by the vehicles from 147271 to 143983 kilometers. The numbers correspond to a 20 day period. Every day the algorithm took an average of 5 minutes to run compared to the 3 hours of human planners.

Multiobjective Vehicle Routing Problems. Multiobjective Vehicle Routing Problems are an extension of VRPs. In all previous VRPs there was a single objective function that had to be minimized respecting some constrains. Now, there are $f_1(x) \dots f_n(x)$ objective functions that have to be minimized. A multiobjective problem (MOP) can be described as follows:

$$\text{(MOP)} = \{ \min F(x) = (f_1(x) \dots f_n(x)) \text{ , s.t. } x \in D \}$$

where D is the feasible solution space and $F(x)$ is the objective vector.

There is a variety of extensions of classic academic/real problems to multiobjective problems [72, 71]. The most interesting ones will be presented below. Usually the first objective function is the total cost that needs to be minimized. As a second objective many researchers propose the makespan (length of the longest tour) that has to be minimized. Also, let l_{max} (l_{min}) denote the length of the maximum (minimum) route. An objective function is the minimization of $L = l_{max} - l_{min}$ bringing a concept of balance (justice) among routes. If the average length of the tour is about the same, there will not be many complains from the drivers regarding difficult/long routes.

Solution Approaches for Multiobjective Vehicle Routing Problems. A solution to a MOP is to calculate the Pareto set (typically exponential in size) where one solution is better regarding one objective and worse in another. The solutions that belong to the Pareto set are called Pareto optimal solutions. There are three main strategies that can tackle a MOP[72, 71]. These are: scalar methods, Pareto methods and methods that do not belong to any of the above such as genetic algorithms, lexicographic ordering, ant colony optimization and tabu search.

A widely used scalar method is weighted linear aggregation. Let $f_1(x), f_2(x)$ are the two objective functions. In this method the two objective functions are combined to one $F = w_1 f_1(x) + w_2 f_2(x)$ where w_1, w_2 are carefully chosen weights. The main advantage of weighted linear aggregation is that it is easy to implement. However, it has many weaknesses. First, the method is unable to find all the Pareto optimal solutions, it only finds the solution on the convex hull of the Pareto set. Secondly, it is not an easy task to choose the weights for each function and it depends on the exact application. A second scalar approach is the ε-constrained method. Here, only one objective is optimized and the others are considered as constants expressed as $f_i(x) \leq \varepsilon_i$.

Pareto methods use the notion of Pareto dominance directly. Many researchers combine Pareto methods with evolutionary algorithms in order to solve multiobjective vehicle routing problems. The main disadvantage of this method is that the size of the Pareto set is exponential. Hence, instead of trying to compute an exact solution many authors try to find a solution that is near the optimal one.

Other methods are genetic algorithms such as VEGA [72]. At each iteration the algorithm divides the population into n subpopulations where n is the number of objectives. Then they are mixed together to get a smaller population on which genetic operations are applied. In lexicographic methods, every objective has a certain priority. The problem is solved in order of decreasing priority. When an objective is optimized, its value will not change so it becomes a new constraint for the problem. The methods of tabu search and ant colony optimization were discussed in a previous section and can also be used in the multiobjective case of VRPs.

Vehicle Routing Problem and Environmental Issues. The impact on the environment of a tour for any VRP (and its variants) has not been extensively studied [113]. In the literature, most work is focused on the minimization of economic costs. The objective of minimizing the total distance traveled leads to the protection of the environment due to reduction in fuel consumption and pollutants (CO_2 ...). However, this is generally not measured or emphasized. For example TDVRP is likely to lead to environmental benefits because road segments with traffic are avoided, but there has not been any study in order to see the benefit.

Conclusions. In this section the Vehicle Routing Problem with its main variations was presented. Solution techniques and approaches were also presented. Then the problem was extended to the multiobjective Vehicle Routing Problem presenting problems and solution strategies. It seems that due to the complexity of these kinds of problems (NP-Hard) it is very difficult to find exact solutions in a reasonable amount of time for many customers. The obvious road to follow is the use of heuristic techniques (Tabu search, Ant Colony Optimization) that compute near optimal

solutions in a reasonable amount of time, especially when dealing with problems of hundreds nodes and edges.

An interesting observation is the lack of research of vehicle routing problem regarding environmental consequences. A multiobjective approach where one of the objectives is a function related to environmental footprint of the tour (that has to be minimized) has not been studied and may lead to "good" solutions from an environmental point of view. In this direction, the time dependent model seems to provide the best basis for the minimization of the environmental footprint of a route.

2.4 Time-Dependent Route Planning

For realistic route planning scenarios it is often not enough to assign static arc lengths of average travel-time because traffic has a huge impact on the travel-time experienced by actual travelers. Two approaches exist: one is to periodically update the routing graph with new arc lengths according to traffic incidents (cf. Section 2.3.2), the other is to look at historical traffic measurements and extract foreseeable trends such as rush hours and use that to find the fastest route w. r. t. the expected traffic situation. In other words, the arc-costs in the underlying network are now functions of time, rather than fixed values. Here, we discuss this latter approach. First we give an overview to the formal problem definition when all costs are considered to be *continuous* functions of time. The model and some theoretical results are presented in in Section 2.4.1, and we review experimental results in Section 2.4.2. In Section 2.4.3 we also provide a case study from a previous EU project, in which an implementation of a *discrete-time-dependent* shortest paths algorithm was implemented.

2.4.1 The Time-Dependent Shortest Path Problem

Problem Statement. The *time-dependent shortest path (TDSP)* problem is a variant of the classical shortest path problem, in which the arc lengths are not fixed values, but are provided as functions of the arrival time at the tail of each arc. The goal is to find an earliest-arrival path (to the destination) for *any* possible departure time from a source node. In particular, we are given a directed graph $G = (V, E)$, and a pair of origin-destination nodes $s, d \in V$. For each arc $e \in E$ we incur a delay while traveling from its tail to its head, whose exact value is given by the corresponding *arc-delay* function: $\forall e \in E,\ D[e]$ is a *non-negative* function providing the actual delay $D[e](t)$ to traverse e as a function of the arrival time t at the tail of e. The *arrival-time* function of e is defined as $A[e] : t \mapsto t + D[e](t)$.

For any particular path $p \subseteq E$, the delay function of p is the aggregation of delays as one moves from the tail of p towards the head of p. Similarly, the arrival-time function of p is given as the *composition* of the arrival-time functions of the arcs comprising it. For example, assuming that $p = \langle e_1, e_2, \ldots, e_k \rangle$ we have:

$$\begin{aligned} D[p](t) &= D[e_1](t) + D[e_2](A[e_1](t)) + D[e_3](A[e_2](A[e_1](t))) \\ &\quad + \ldots + D[e_k](A[e_{k-1}](A[e_1](t))) \\ \\ A[p](t) &= A[e_k](A[e_{k-1}](\cdots(A[e_2](A[e_1](t)))\cdots)) \\ &= A[e_k] \circ A[e_{k-1}] \circ \cdots \circ A[e_1](t) \end{aligned}$$

For two particular nodes $u, v \in V$, we denote by P_{uv} the set of $uv-$paths in G. The function $A[u, v]$ is the *earliest-arrival-time* function from u to v, i.e., the minimum over all $uv-$paths among the arrival-time functions:

$$A[u, v](t) = \min_{p \in P_{uv}} \{A[p](t)\}$$

The goal of the *Time Dependent Shortest Path (TDSP)* problem is, given a particular pair of source-destination nodes $s, d \in V$, to compute (and store) the entire function $A[s, d]$. We typically

refer to the *complexity* of this function, meaning the number of distinct values of $A[s,d]$ as the time variable varies in the non-negative axis. Clearly the complexity of the earliest-arrival-time function determines the size of the data structure that should be used to store it. This problem is quite important, particularly in large-scale road networks, where it might be prohibitively expensive to query directly the earliest-arrival-time $A[s,d](t)$ to the destination d, departing from the source s at a given time t, even though theoretically this could be done in slightly superlinear time (e.g., if the underlying network satisfies the FIFO property, see next subsection). We would definitely prefer to exploit, if possible, properly constructed data structures (oracles) of reasonable size, that would allow for example sublinear (e.g., constant or polylogarithmic) query times. Such a data structure is called a *travel planning map*, or earliest-arrival profile in the literature.

Typically when considering arc-delay functions, the provided data are in the form of piecewise linear (pwl) functions which are actually the interpolants of periodical sample points of delay values in the corresponding arcs. If the arc-delays are pwl functions, then it is easily seen that the same holds also for the arc-arrival functions for arcs, as well as for their compositions which express path-arrival functions. Additionally, the minimization operation (to determine earliest-arrival-time functions between nodes) produces again a pwl function. That is, if the arc-delays are pwl functions of time, then the same holds for all the involved delay and arrival-time functions for paths and pairs of nodes. Therefore, the complexity of the function $A[s,d]$ is actually characterized by the *number of breakpoints* describing it. Observe also that for each leg (between two consecutive breakpoints) $A[s,d]$ is actually linear, and the corresponding shortest $sd-$path remains the same (although its aggregate delay changes with time).

TDSP has been studied since 1969, when Dreyfus [49] observed that the classical algorithm of Dijkstra can be used to compute time-dependent shortest paths, for any *given* departure time from the source node. In his work, Dreyfus implicitly assumed unrestricted waiting policy at intermediate nodes, meaning that one may wait at a node for the optimal (w.r.t. earliest-arrival-time function) departure time from the tail of the next arc to follow. Indeed, the restriction (or even exclusion) of waiting policies may have severe implications on the complexity of computing a time-dependent shortest $sd-$path for a given departure time from s. For example, Orda and Rom [100]) proved that if waiting is forbidden anywhere in the network, and the delay functions do not possess the FIFO property, then there may exist time-dependent shortest $sd-$paths that violate the subpath optimality property. Indeed, there may even exist paths with an infinite number of edges which nevertheless have finite delay value. Therefore, neither Dijkstra's algorithm nor Bellman-Ford algorithm can be used in these cases. In some cases of restricted waiting policies and non-FIFO behavior of the network, it was also shown [117] that computing a time-dependent shortest path for a given departure time from the source is NP hard problem.

Observe that in time-dependent networks a non-FIFO arc is equivalent to a FIFO arc, if unrestricted waiting is allowed in the network. Observe also that the FIFO property for arcs is expressed as follows:

$$\forall t, t' \geq 0,\ \forall e \in E, \qquad t < t' \rightarrow t + D[e](t) \leq t' + D[e](t')$$

Foschini et al. [54] recently proved the following results for TDSP in FIFO networks with pwl arc delays: The earliest-arrival-time function $A[s,d]$ may have $n^{\Theta(\log n)}$ BPs. The lower bound is actually a direct consequence of a breakpoint-preserving reduction from a related problem, the *parametric shortest path (PSP)* problem: Every arc has a length function $L[e](\gamma) = a_e \cdot \gamma + b_e$. The length of a path p is the sum of the arc-lengths for the arcs comprising it. The goal is to compute the entire *shortest-path length* function $L[s,d]$ between the source node and the destination node, in the entire domain of the *real-valued* variable $\gamma \in (-\infty, \infty)$. It is well known that there exist instances for PSP in which the number of breakpoints for $L[s,d]$ is $n^{\Omega(\log n)}$ [92]. It is proved in [54] that there cannot exist more than $K \cdot n^{O(\log n)}$ breakpoints, where K is the number of *primitive* breakpoints (i.e., the ones caused by breakpoints on the arc delays, and not due to the minimization operation).

An output-sensitive algorithm is also described in [54], that constructs $A[s,d]$ in which the *construction time per breakpoint* is polynomial in the input size (description of the instance).

A polynomial-time construction is also described in [54], of a *travel-time* function $\tilde{D}[s,d]$ that provides a $(1+\epsilon)-$approximation with respect to the *smallest-delay* function $t \mapsto A[s,d](t) - t$, whose number of breakpoints is $O\left(K\frac{1}{\epsilon}\log\left(\frac{D_{\max}}{D_{\min}}\right)\right)$. Observe that the complexity of $\tilde{D}[s,d]$ is now *independent* of the network size. $D_{\max}, D_{\min}$ are the maximum and minimum delays that may appear (over all possible departure times from the source) along the shortest $sd-$path in the network.

Finally, [54] provided a simple $O(K(n\log(n)+m))-$time algorithm for computing the *min-delay path*, over a *given* interval of possible arrival times.

2.4.2 Time-Dependent Route Planning on Road Networks

In this section, we discuss recent approaches to time-dependent route planning on road networks obtained by analysis and experimentation on real-world road networks. A road network is modeled as a weighted, directed graph $G(V,E)$ the same way as in Section 2.3.2 but the length $\text{len}(e)$ of an edge $e=(u,v)$ is a *periodic piece-wise linear* function of the time of day (or more generally any given time period Π) mapping departure time at the tail u to the travel-time on e (*delay function*). Two types of queries are common: *earliest arrival* (EA)—finding the shortest path for a given departure time t, and *profile search*—finding the shortest path for every departure time in Π. Earliest arrival queries can be solved by a generalization of Dijkstra's algorithm that evaluates the length of an relaxed edge $e=(u,v)$ based on the arrival-time at node u. Evaluation of the delay function can be done by binary search but in practice linear search or linear search with initial guessing of the right interpolation point is much faster. Profile queries can be solved by a label-correcting generalization of Dijkstra's algorithm that propagates arrival time function instead of scalars as labels. When relaxing an edge $e=(u,v)$ it *links* the arrival time function $\text{dist}(u)$ of the tail with the delay function $\text{len}(e)$ of the edge to obtain a new arrival time function $\text{dist}'(v)$ at the head node which might improve on the current function $\text{dist}(v)$ but possibly only for certain times in Π. Therefore, we have to *merge* $\text{dist}'(v)$ and $\text{dist}(v)$. Linking two functions f,g is defined as $f \oplus g := f + g \circ (\text{id} + f)$, i. e. $(f \oplus g)(\tau) := f(\tau) + g(\tau + f(\tau))$. Merging finds the piece-wise minimum of two functions f,g.

The practical performance of profile search is bounded by the number of label reinsertions and the growth of complexity of the label functions. The growth of interpolation points when linking and merging is at worst linear in the input: $\mathcal{O}(|I^f| + |I^g|)$, which for practical applications can be overwhelming. While EA-Dijkstra search shows only a slight speeddown over static Dijkstra search of about 20–28%, Profile-Dijkstra search becomes impractical on large road networks because to much memory is needed.

However several preprocessing techniques have been adapted from the static scenario and allow for the fast computation of earliest arrival and profile queries on large time-dependent road networks. The following gives a short overview of how common ingredients of static preprocessing techniques are adapted. For details on these ingredients, see Section 2.3.2. For a more detailed discussion on time-dependent techniques, see [45].

Landmarks. Correctness of goal-directed search is based on the positivity of the potential-reduced costs. Thus, increased edge costs do not invalidate a potential. Hence, landmarks can be computed based on the lower-bound graph $\underline{G}$ where all edges of G have their scalar length set to the lower bound $\underline{\text{len}(e)}$. This yiels correct results but as the potential is less tight this modification achieves less speedup.

Bidirectional Search. The departure time of an earliest arrival query is given but the arrival time is unknown. Hence, it is unclear at what time to start the backward search. But a backward

search on the lower- and upper-bound graphs $\underline{G}$, $\overline{G}$ can be used to extract a subgraph of G relevant for forward search, which only has to relax edges $e = (u, v)$ with $\underline{\text{len}(e)} + \underline{\text{dist}(v, t)} \leq \overline{\text{dist}(u, t)}$ (*time-corridor search*). Note that a time-corridor can also be formulated for the forward direction ($\underline{\text{dist}(s, u)} + \underline{\text{len}(e)} \leq \overline{\text{dist}(s, v)}$) and can be of use to guide profile queries.

Arc-Flags. Instead of setting the flag $AF_C(e)$ to `true` if the edge e is on a shortest path to cell C, the flag is set to `true` if it is on a shortest path to C at least once during the time period. In order to preprocess this information one could grow profile graphs from the border nodes of each cell, but that is too slow. Faster preprocessing can be achieved by using lower and upper bounds to determine flags: A flag $AF_C(e)$ of an edge $e = (u, v)$ is to `true` if $\underline{\text{len}(e)} + \underline{\text{dist}_C(v)} < \overline{\text{dist}_C(u)}$.

Contraction. Node contraction is independent of the metric and does not need to be adapted. But shortcuts inserted during the contraction step need to represent linked functions which can cause tremendously increased memory overhead. Also, for witness search a (local) profile query needs to be run, which is much slower. Witness search can be accelerated by first checking if $\underline{\text{len}(e)} > \overline{\text{dist}(u, v)}$ because then the shortcut $e = (u, v)$ cannot possibly be necessary at all. It is also possibly to employ *shortcut approximation*: Only lower $\underline{\text{len}(e)}$ and upper bound $\overline{\text{len}(e)}$ are saved on a shortcut e (instead of the whole delay function). During the query time-corridor search is used to determine if a particular shortcut might be necessary. If so, the real weight is obtained by unpacking the shortcut on-the-fly.

Using these adaptations several time-dependent speed-up techniques have been devised, among these: time-dependent ALT [94], time-dependent Core-ALT [42], time-dependent SHARC [35], and time-dependent Contraction Hierarchies [19] with approximations [20] (TCH, ATCH). While the time-dependent variants of ALT and Core-ALT have very mild preprocessing overhead, time-dependent SHARC and ATCH achieve practical query performance but their preprocessing takes up to several hours. Since preprocessing is not locally constrained as with [37] such long preprocessing can be prohibitive when integration of current traffic incidents is added to the scenario. Also, to our knowledge none of these time-dependent techniques have been generalized to a time-dependent multi-criteria setting.

2.4.3 A Case Study in a Related Research Project

An attempt to consider time-dependent shortest path calculations appeared also in the EU funded project GOOD ROUTE[3], focusing on *discrete-time-dependent* arc-cost functions. The algorithm used for the calculation of the optimal route (minimum cost; with the term cost we mean either the economic, risk-related, or combined cost) is a modification of the *discrete time-dependent* shortest path algorithm of [130].

Time-dependency of costs means that cost values that are variable with time. Such data are the risk-related cost that has been calculated by the Risk Estimation Module and the calculated economic cost. In the GOOD ROUTE approach, the day is divided into a number of time intervals of the same length. The number of the time intervals can be set by the system operator and depends on the data that are available. In this way we can take into account the variations of traffic loads and of population distribution. Moreover, some other parameters that the Risk Estimation Module takes into account and affect its results, like weather conditions, are also variable with time.

The variability of the data (traffic loads, population density, weather conditions, etc.), is one point. The other point is the number of time intervals that the Optimum Route Calculation Algorithm uses internally. The number of those time intervals should be an integer multiple of the number of time intervals of our data. The larger that number is, the more accurate results are

[3] http://www.goodroute-eu.org/

produced. That happens because the truncation error is smaller when the time interval duration is smaller. However, the time of calculation also grows, when the number of time intervals grows. There are two versions of the algorithm, one with fixed departure time and another with fixed arrival time. The latter can be very useful in our case, as it addresses the frequent problem of a delivery that should be done at a certain time.

Algorithm Description (Fixed Departure Time)

Let $G = (V, E)$ be a V node finite directed graph with E directed edges connecting the nodes. Let $d_{ij}(t)$ be the non-negative time required to travel from node i to node j , when departure time from node i is t , $c_{ij}(t)$ be the non-negative total combined cost of the travel from node i to node j, when departure time from node i is t ; $d_{ij}(t)$ and $c_{ij}(t)$ are real-valued functions defined for every $t \in S$, where $S = \{t_0, t_0 + \delta, t_0 + 2\delta, \ldots, t_0 + M\delta, \}$. t_0 is the earliest possible departure time from any origin node in the network, δ is a small time interval during which some perceptible change in traffic conditions may occur, and M is a large integer number such that, the interval from $t + 0$ to $t_0 + M\delta$ is the time period of interest.

We denote by node N the destination node of interest in the network. The algorithm calculates the time-dependent lowest combined cost paths from every node i , at every time step t , to destination node N.

At each step of the computation, denote by $\lambda_i(t)$ the total combined cost of the current lowest cost path from node i to node N, at time t. Let $\Lambda_i = [\lambda_i(t_0), \lambda_i(t_0+\delta), \lambda_i(t_0+2\delta), \ldots, \lambda_i(t_0+M\delta)]$ be an M-vector label that contains all the labels $\lambda_i(t)$ for every time step $t \in S$ for the node i. Every finite label $\lambda_i(t)$ from node i to node N is identified by the ordered set of nodes $P_i = \{i = n_1, n_2, \ldots, n_m = N\}$.

$\lambda_i(t)$ is defined by the following functional equation:

$$\lambda_i(t) = \begin{cases} \min_{i \neq j}\{c_{ij}(t) + \lambda_j(t + d_{ij}(t))\}, & i = 1, 2, \ldots, N-1; t \in S \\ 0, & i = N; t \in S \end{cases} \tag{3}$$

Instead of scanning all the nodes at every iteration, a list of Scan Eligible (SE) nodes is maintained, containing the nodes with some potential to improve the labels of at least one other node. The algorithm operates in a label correcting fashion. Therefore, the label vectors are just upper bounds to the lowest cost paths until the algorithm terminates.

Initially, the SE list contains only the destination node N . At the first iteration, all the nodes that can directly reach N are updated according to equation 2, and inserted to the SE list.

$$\lambda_i(t) = c_{ij}(t) + \lambda_j(t + d_{ij} \leq (t)), i \in \Gamma^{-1}\{N\} \tag{4}$$

where $\Gamma^{-1}\{N\}$ is the set of nodes that can directly reach N. The rest of the labels are set equal to infinity.

Next, the first node i of the SE list is scanned according to the following equation:

$$\lambda_j(t) = \min_{i \neq j}\{\lambda_j(t), c_{ji}(t) + \lambda_i(t + d_{ji}(t))\}, j \in \Gamma^{-1}\{N\} \tag{5}$$

for every time step $t \in S$. If at least one of the components of Λ_j is modified, node j is inserted into the SE list. This scheme is repeated until the SE list is empty, and the algorithm terminates. Here is the algorithm in steps:

Step 1. Create the SE list and initialize it by inserting into it the destination node N. Inititalise the label vectors with the following values:

$$\Lambda_N(0, 0, \ldots, 0) \tag{6}$$

$$\Lambda_N(\infty, \infty, \ldots, \infty) \forall i = 1, 2, \ldots, N-1. \tag{7}$$

Step 2. Select the first node i from the SE list, name it "Current Node" and delete it from the list. If the SE list is empty, go to step 4.

Scan the current node i according to relation 3, by examining each node j, $j \in \Gamma^{-1}\{i\}$.

Specifically, for every time step $t \in S$ do the following:

Check if $\lambda_j(t)$ is greater than $c_{ji}(t) + \lambda_i(t + d_{ji}(t))$. If it is, replace $\lambda_j(t)$ in the label vector Λ_j at position t with a new value. If at least one of the M labels of node j has been improved, insert node j in the SE list.

Step 3. Repeat step 2.

Step 4. Terminate the algorithm. The M-dimensional vectors Λ_j for every node i in the network contain the combined costs of the time-dependent lowest-cost paths, from every node i to the destination node N for each time step $t \in S$.

Upon termination of the algorithm, every element of the vector label is either an infinite number, meaning that no path exists from this node to the destination node at the corresponding time step, or a finite number that represents the lowest combined cost path from this node and time step to the destination node.

Algorithm Description (Fixed Arrival Time)

This is the second version of the algorithm, which could be very useful in the real-life problem. In this case, the truck company/driver requests a new transport, from node i_d to node i_a, with desirable arrival time t_a at node i_a.

Step 1. Run the fixed departure time version of the algorithm.

Step 2. Check $\lambda_{i_d}(t)$ for $t \leq t_a$ (starting from t_a and going backwards), to find the latest departure time t that results in arrival at destination node i_a at time t_a.

Of course, like in the fixed departure time version of the algorithm, the number of time intervals plays a trivial role in the accuracy of the results. However, in this version of the algorithm, the importance of it is even bigger.

2.4.4 Public Transit and Multi-Modal Route Planning

So far in this section on time-dependent route planning we have only considered road networks. Other networks with time-dependent delay functions exist, for instance public transit networks. The modeling of time-dependency in these networks is somewhat similar to road networks insofar as piece-wise linear functions can be used. But the behavior of these functions under linking and merging is rather different (and more well-behaved). Earliest arrival and profile queries are also relevant for public transit networks, and profile queries have a lower overhead on those networks. As we are aiming to solve rich multi-modal scenarios in the eCOMPASS project we are very aware of recent results on public transit and multi-modal route planning and we have a clear understanding by which steps to approach our goal. Because Deliverable D2.1 is on road networks only we refer the interested reader to Deliverable D3.1 for details.

2.5 Alternative and Robust Routes

In this section, we review the current state of the art in the case of searching for alternative and/or robust routes. We are interested not only in finding *one* route from a given start to a given end but also finding *several* alternative others. There are several reasons for this to be advantageous. These may be based on user preferences due to differences in driving difficulty, scenic value, risk of traffic jams, fuel consumption etc., or unforeseen events like unavailability of some roads due to construction work, storms or traffic jams. Therefore, by having an option to change the initial route significantly without increasing too much the initial cost of the route is much desired. In this section, we review the most important recent techniques for computing alternative and robust routes. In particular, in Section 2.5.1 we present a review of [6] where alternative paths are evaluated

for their admissibility, and an initial approach for their computation is presented. In Section 2.5.2 this approach is further optimized in [85] by partitioning the underlying graph. In Section 2.5.3 we present a review of [13], where alternative paths are evaluated globally, using alternative graphs (a concatenation of alternative paths). Finally, in Section 2.5.4 we review an approach presented in [41] for solving a slightly different problem, where an alternative route has to be computed during a query, due to unintended deviations of the driver from the shortest path.

2.5.1 Alternative Routes in Road Networks

An attempt to formalize the notion of a *good* alternative path is presented in [6]. Intuitively, a good alternative path should not only have a small difference in length from the optimal path but also share no more than a small fraction of edges of the actual optimal path, in order to be considered a *different path*. Therefore there is a need to formalize the conditions that must be satisfied in order for a path to be considered admissible as an alternative to the shortest path.

Let Opt be the shortest path between a source node s and a target node t on the graph, and let $l(Opt)$ be the length of the shortest path. An alternative path P must share a small number of edges with the original shortest path Opt. That is because it must be significantly different from Opt to be considered an alternative. This condition is called *limited sharing* and is formalized as $l(Opt \cap P) \leq \gamma \cdot l(Opt)$ where $0 \leq \gamma \leq 1$.

Moreover, the path must be *reasonable*, with no unnecessary detours. While driving on it, every local decision must make sense. Therefore it must be *locally optimal* within a length T. Formally, every subpath P' of a T *locally optimal*(T-LO) path P with $l(P') \leq T$ must be a shortest path.

Last but not least, a local optimal path is reasonable but is not yet a candidate admissible path since it also has to be sufficient. Thus, its length should be as close as possible to the actual shortest path, formally, it must have limited stretch. A path P has $(1+\epsilon)$-*uniformly bounded stretch* ($(1+\epsilon)$-UBS) if every subpath (including P itself) has stretch at most $(1+\epsilon)$.

In summary, given three parameters $0 < \alpha < 1$, $\epsilon \geq 0$, $0 \leq \gamma \leq 1$, the formal conditions for a path to be considered as an alternative are:

1. $l(Opt \cap P) \leq \gamma \cdot l(Opt)$ (limited sharing)
2. P is T-locally optimal for $T = \alpha \cdot l(Opt)$ (local optimality)
3. P is $(1+\epsilon)$-UBS (uniformly bounded stretch)

Given an algorithm to list all admissible paths, one can select the best one according to one arbitrary objective function. However, these paths may be numerous, making it hard to find one efficiently.

One alternative is to select them from a subclass of admissible paths. An easily defined, analysed and amenable to practical implementation subclass is the *single via paths* subclass. Given a vertex v, the *via path through* v, P_v is the concatenation of two shortest paths, s-v and v-t. These paths have the following interesting properties:

- They are the shortest of all paths s-t that pass through v, therefore they have the lowest stretch.
- Since they are a concatenation of shortest paths, they are locally optimal through their span except, maybe around v. A simple quick algorithm (T-*test*) to test whether a via path is locally optimal around v and by extension on its whole is proposed in [6].
- They can be efficiently enumerated and tested for admissibility.

As a result, it is more efficient to find and test all admissible *via paths* than all actual admissible paths. Moreover it is easier to devise an efficient algorithm to enumerate all admissible *via paths*.

All proposed algorithms share the same core approach: They grow shortest path trees from s and t using the Bidirectional Dijkstra's algorithm. Each vertex v scanned by both searches defines a single via path P_v which can then be tested for admissibility. Of course there is no need to grow the full shortest path trees since there is also the *limited stretch* requirement. Therefore, each search is stopped as soon as it advances more than $(1+\epsilon)l(Opt)$ from its origin. This approach is called *BDV*.

An enhancement to BDV is the recognition of large *plateaus*, an idea borrowed from the *choice routing algorithm*[84]. A plateau is a maximal path that appears in both search trees simultaneously. A plateau consists of via paths and unless it resides on the shortest path it has no shared edges with it. Therefore, admissible paths with large plateaus are preferred to ones with smaller plateaus.

Apart from the BDV approach, two faster alternatives are proposed in [6]. Both prune unimportant vertices during the searches. The first is pruning nodes based on their *reach* value and is called *REV*. The reach of a node v is defined as the maximum, over all shortest u-w paths containing v, of $\min\{dist(u,v), dist(v,w)\}$. The second is using the *contraction hierarchies* method (*CH*), as described in [57], and is called *CHV*. Using each method, the algorithms execute the same steps as BDV apart from the pruning during the search phase. The main advantage of both methods is that they significantly reduce the number of via nodes they have to consider. The disadvantage is that for each via node considered, two additional queries s-v and v-t have to be performed, since pruning nodes may yield suboptimal paths.

Unfortunately neither of the above methods is quick enough to be used in practical applications. This is mainly due to the fact that each of these methods enumerates first all admissible alternative paths and then selects the best of them according to an objective function. A much faster approach is to consider each candidate *via path* in order of discovery and stop as soon as the first admissible one is found. In order for this approach to be effective, paths are considered in a heuristic order that prefers paths that have smaller length, less sharing and including larger plateaus.

A comparison of the algorithms in terms of both query performance and path quality is conducted in [6]. The path quality is given by the *success rate* of each algorithm which is the rate at which the algorithm finds as many alternatives as desired. The result was that there is a clear trade-off between success rates and query times. BDV is successful more often, while it has also the highest query times, which makes it unsuitable for practical applications. CHV is faster than BDV, however it has a success rate of 50% for finding just one alternative path. However, alternative paths, when found, tend to have similar quality, regardless of the algorithm.

2.5.2 Candidate Sets for Alternative Routes in Road Networks

Based on the algorithms proposed in [6], a new approach presented in [85] tries to engineer efficient alternative algorithms. These algorithms are based on the observation that the algorithms in [6] work in two steps. At first, a call of the bidirectional *Exploration*(CH) Dijkstra's algorithm finds all via node candidates. Then, these are sequentially tested for admissibility using point-to-point shortest path queries (calls of *Target*(CH) Dijkstra's algorithm). The calls of the Target Dijkstra's algorithm can be further optimized by using speed-up techniques. In [85] the CHASE variant is used that computes these queries by exploiting additional arc flags [24]. The resulting algorithm is called *X-CHASEV*.

A critical observation is that the proposed algorithms from [6] work very well on a class of graphs in which most shortest paths leaving a *region* go through a small set of nodes. This means that all shortest paths leaving the region are *covered* by this small node set. This is also true for the plateaus, and by extension for all the admissible paths of the algorithm in [6]. Therefore, these node sets can be used to partition the graph. A shortest path switching partitions is bound to go through these via node sets, thus to compute actual alternative paths it suffices to compute alternative paths between these boundary nodes.

The pre-computation of the via node candidates starts with a partitioning of the underlying road graph. Such partitioning schemes are described in [34] and [110]. For each pair of partitions

the set of via node candidates is computed in a greedy fashion. When computing an alternative route between two non-neighbouring partitions, the associated via nodes to this pair are all tested for admissibility. The first admissible via node path is returned as a result. If no admissible alternative path is found then no path is returned. In case of an s-t query within the same partition or between neighbouring ones the algorithm X-CHASEV is used instead. This is actually faster than just testing precomputed via nodes due to the small size of the search.

At this stage, the algorithm does not compute via node candidates within the same partition or neighbouring ones. To use the advantage provided by precomputed via node candidate in such cases, a multilevel partitioning scheme is proposed. Each initial partition on the coarsest level is partitioned again into smaller partitions with finer detail. Each finer partition is included exactly once in a coarser partition. This process is gradually repeated in multiple levels.

The pre-computation of the via node candidates between pairs of partitions then is defined as follows. In the coarsest level the pre-computation is the same as with the single level approach described before. When moving to finer detail, the partitions are becoming too many to consider computing via node candidates between each fine pair. Therefore, the pre-computations are limited to pairs of partitions that are close to each other. These are the pairs that are not neighbouring but exist in the same coarser partition or on neighbouring coarser ones.

During a query, the algorithm first checks if s and t belong to different non-neighbouring coarser partitions. If this is the case then the algorithm selects the via node candidates between this pair of partitions and tests them for admissibility as with the single level scenario. In any other case, it recurs in the next finer level in the multilevel structure and continues from the start. Only when it reaches the finest level of partitions and s and t belong to the same or neighbouring partitions does it fall back on the X-CHASEV algorithm. In such case the fine partitions are much smaller and the fall-back runs so fast that it poses no time penalty.

Last but not least, an advantage of the proposed algorithm is that it can use any existing legacy algorithm apart from X-CHASEV in the finest partitions. Moreover, it can fully omit the pre-computation stage and learn candidate via node paths on the fly, from previously executed queries. This makes it suitable to be run in an online scenario on top of a legacy system that implements a baseline algorithm with few or no modifications.

In an experimental evaluation in [85], the proposed algorithm was tested against the algorithms proposed in [6]. The results were that not only does X-CHASEV improve considerably on query times compared with the previous approaches but also it improves the success rate, while the path quality remains at a high level.

2.5.3 Alternative Route Graphs in Road Networks

A different approach for computing alternative routes is presented in [13], when a graph (instead of a path) of alternative routes is computed. The main difference from [6] is that the approach in [13] considers alternative routes globally and tries to measure their quality rather than computing one reasonably good path at a time. The disadvantage of the sequential testing of alternative paths is that their admissibility depends on the order in which they are inserted.

In [13], the notion of *alternative graphs* is explored. An *alternative graph* (AG) is a union of several paths from a source node s to a target node t. An edge $e = (u, v)$ is included in the AG$= (V', E')$ of a graph $G = (V, E)$ if there is a path from u to v in G and there is a path from s to t in G that contains the path corresponding to e. Edges on alternative graphs have weight equal to the path's weight they represent.

A *reduced* AG is an alternative graph where each node that has $indegree = 1$ and $outdegree = 2$ is contracted. The resulting alternative graphs are very small in size, which means that even computationally expensive algorithms can be applied on them. The drawback is that eventually the edges in the alternative graph have to be unpacked into the original paths in G. However, alternative graphs are usually very small, and even expensive unpacking algorithms can be invoked efficiently.

The approach of measuring the quality of the paths considers all alternative routes globally, as an alternative graph. The quality measures of an alternative graph are as follows:

$$\begin{aligned} totalDistance &:= \sum_{e=(u,v)\in E'} \frac{w(e)}{d_H(s,u) + w(e) + d_H(v,t)} \\ averageDistance &:= \frac{\sum_{e=(u,v)\in E'} w(e)}{d_G(s,t) \cdot totalDistance} \\ decisionEdges &:= \sum_{v\in V'\setminus\{t\}} outdegree(v) - 1 \end{aligned} \tag{8}$$

where d_G denotes the shortest path distance in graph G and d_H the shortest path distance in the alternative graph. The total distance measures the extend to which the routes are non-overlapping. The average distance measures the stretch of the alternative paths. The decision edges measure the complexity of the alternative graph. Usually, the function to be maximized is $totalDistance - \alpha(averageDistance - 1)$ after limiting the number of $decisionEdges$ and $averageDistance$. More metrics can be appended, like the variance of paths, in order to further analyse the alternative graphs.

After defining the attributes of the alternative graphs that directly affect their quality, there is a way to compare different methods of computing alternative paths with respect to this quality. Most methods rely on heuristics to compute an AG rather than enumerating all single paths. Usually, each method computes the shortest path, adds it to the AG and then tries to complement it by gradually adding paths to the alternative graph. What follows is a study on the different approaches to compute alternative paths.

A widely used approach is to compute the k *shortest paths* between a source node s and a target node t. However the computed routes are usually too similar to be useful. Good alternatives occur often only for k being very large. This is especially true when driving through a city, where a small detour in a neighbourhood yields a slightly different path with almost the same cost as the shortest path which is, however, in no case considered as a distinct alternative by humans.

Another classical approach to compute alternatives is *Pareto optimality*. Of course, since there exists only one weight function on the graph, it needs to be complemented with another one. The second weight function is zero when an edge is not contained in the AG and equal to the edge's weight when the edge is part of the AG. These values are set this way in order for edges outside the AG not to be dominated by edges in AG and be favoured for inclusion. A generalized Dijkstra algorithm can then compute all Pareto-optimal paths with a few modifications. Last but not least, the algorithm can be calibrated to decrease the number of computed paths by tightening the domination criteria.

Another approach is the *Plateau* method. In this approach, two shortest path trees are being grown from both s and t. A plateau is a maximal path that appears in both shortest path trees. Since there are many plateaus, there is a need of selecting the best of them. Therefore, they can be ranked by the formula *rank = (path length − plateau length)*. A plateau reaching from s to t would have rank equal to 0, which is the best value.

In the last method, the *Penalty approach*, a shortest path is firstly computed and added to the AG. Then it has its edge weights increased before starting the next s-t path computation. The process is finished when the AG has a satisfactory number of alternative paths. The most crucial point of this method is the way of adjustment of the edge weights after each shortest path computation. Adding a constant value favours longer edges over shorter ones. Therefore, instead of a constant, it is preferred to add a fraction of the original edge weight to the weight of an edge (*penalty factor*). A second problem that arises is that parts of shortest paths that still exist in later iterations, are likely to get increased multiple times during the process. For example, this could happen on a highway before entering a city. Through the city there are many alternative paths, but since the highway is the only reasonable route into the city, it is unintuitive to penalize this part of

the path multiple times. Therefore, the method avoids increasing the same edges more than a few times. The main limitation of the approach is that the new shortest paths may be quite similar to the previous one apart from many small detours (hops). These look unpleasant to humans and contain no real alternative. For this reason, when increasing a shortest path's weights, the weights of edges around the shortest path, that leave and join the current AG are furthermore penalized (*rejoin-penalty*).

Concluding, since the penalty method works on a pre-computed set of alternative routes, it can be combined with any other method. A very promising combination is the Penalty and Plateau method which yields superior results to other single methods.

All the above approaches were evaluated in terms of their resulting graph qualities. The evaluation was conducted in two different ways. At first each method was evaluated by the base target function $totalDistance - (averageDistance)$. Then, a user survey was conducted, asking users to identify real-life alternative routes they were familiar with. Each method's results were checked for matching against the user defined paths. In both the above evaluation approaches the combination of the Penalty approach with the Plateau approach yielded higher quality results.

2.5.4 Robust Mobile Route Planning with Limited Connectivity

A slightly different problem from the above is studied in [41]. The main problem is to find an alternative path leading back to the shortest path in case of an unintentional deviation during a parsing of the shortest path. Such deviation in the context of drivers and road networks suggests a mistake on behalf of the driver which must be corrected as soon as possible.

The scenario in this study is that a driver has limited connectivity with the server computing the shortest paths, and can have information only about the actual shortest path and some local segments around it (a *corridor*) in order to quickly correct his route in case of a deviation. These constraints are a result of a mobility scenario, where the devices held by the drivers have limited resources, and frequently communicating with the server is impossible. The goal is to provide the driver a local sub-graph along with the shortest path, in order for him to auto-correct his route, and to make this sub-graph as robust as possible.

A straightforward solution for this problem in case a driver was heading to a target node t, would be to send him the whole shortest path tree $T(t)$ leading to t. In such case, when there is a deviation, the driver can always continue through another branch of the shortest path tree to t. Of course, that would be an enormous amount of data to be sent for each query. Thus, a more sensible solution is to send a sub-tree of $T(t)$ that is as small as possible but also robust against deviations. The actual shortest path augmented with this sub-tree is called a *corridor*.

An obvious first choice of a corridor is to include all vertices that are "close" to the shortest path in terms of cost. A corridor containing all nodes that are at most τ away from the shortest path is called τ-*perimeter corridor* (τ-PC). This definition unfortunately does not work as expected since in some cases, such as in urban areas, there are too many nodes nearby, but no important roads.

Another approach is a *turn corridor* (TC). This approach does not take into consideration the cost of the edges, rather than the actual unintentional deviations this corridor is robust to. At first a shortest path P_{st} is augmented by all vertices adjacent to P_{st} and their respective paths on the shortest path tree into t, forming the corridor TC_1. This way, in case of just one deviation, the driver is still on the shortest path tree and this solution is robust against one deviation. In order to make this solution robust to more than one deviation, the process is recursively repeated and the corridor TC_i, $i \geq 1$, is augmented by the vertices adjacent to it (*deviation vertices*) and their respective paths. This process repeated k times yields a turn corridor TC_k which is robust against k deviations.

The next step is a discussion on how to compute corridors efficiently. The first straightforward approach is to construct the full $T(t)$ with Dijkstra's algorithm and then iteratively append nodes and paths from $T(t)$ to the shortest path. The main drawback is the prohibitive time to build

the shortest path tree. Therefore, this has to be done using techniques that optimize the plain Dijkstra's algorithm. Several proposed techniques are described below.

A promising optimization is the use of the PHAST algorithm described in [36]. This algorithm exploits parallelization on CPUs or even on GPUs to faster compute shortest path trees. This approach, however, is still not fast enough on CPUs. Implementation on GPUs is also problematic since there are not only great delays in the communication between CPU and GPU but also great difficulty in transforming existing algorithms to work in parallel.

Another approach is using a point-to-point speed-up technique of Dijkstra's algorithm to efficiently compute the shortest paths towards t from each node of a corridor without actually computing the whole shortest path tree $T(t)$. Such a speed-up technique is *contraction hierarchies* [57]. Using CH, a query is run for each deviation vertex, and the unpacked shortest path is added to the corridor. The main drawback is that the upward search from t in the CH query is visiting the same nodes multiple times, thus, it performs many unnecessary computations.

This can be remedied by storing the upwards search space from t. The upward search is run only once and every scanned node distance is stored. Then an upward search from each deviation vertex is performed, stopping at scanned nodes. This has roughly 30% less computations than the previous approach. However, even this approach is not entirely efficient, since the resulting paths have to be unpacked multiple times.

Finally, trying to remedy all the above drawbacks, an efficient algorithm for computing corridors, namely *TCC*, is suggested in [41]. The search space from t is still stored and nodes added to the corridor are marked as *final*. Their paths are added in the corridor unpacked. The search from new derivation vertices is stopped as soon as it reaches the *final* vertices already in the corridor and the relative shortcuts are partially unpacked extending the already unpacked paths inside the corridor.

In a performance evaluation in [41], TCC outperforms any other algorithm for computing corridors with k turn tolerance larger than 1. Even for higher k it is very fast, making it efficient for practical applications. In terms of quality of the computed corridors, the corridors proposed by [41] are very robust. By randomly changing directions with a possibility of 5 at each intersection, and using a 3-turn corridor around the shortest path, the rate of staying inside the corridor and continuing to the correct shortest path was 95%.

2.6 Driver eco-Coaching

The term *driver eco-coaching* refers to providing active feedback to car drivers regarding the efficiency of their driving style with respect to fuel consumption and emissions. Such systems can either be "live", i.e. they provide real-time feedback while the driver is on the road, or "post-drive", which means that drivers can access information about their trips afterwards, usually provided on a portable device like a PND or a smartphone, or on a website.

Quite clearly, eco-coaching systems have the potential to play a significant role in reducing the emissions of car traffic. In this section, we shall analyze eco-coaching solutions currently available on the market, take a look at other ongoing research projects on the matter, and finally conclude with opportunities and challenges regarding eco-coaching for eCOMPASS. We start by presenting the eco-coaching approaches in the navigation market (cf. Section 2.6.1), and then proceed with a presentation of currently running research projects related to this issue (cf. Section 2.6.2).

2.6.1 Eco-Coaching in the Navigation Market

Several navigation providers offer eco-coaching features and systems on the market today. In this section, we will describe how three representative systems aid their users to be more eco-friendly.

Most PNDs sold by **Garmin** today come standard with a so called ecoRoute[4] feature. When activating it, users need to enter the average fuel consumption of their vehicle on different road

[4] `http://www.garmin.com/buzz/ecoroute/`

classes, and their current cost for of fuel per liter. Besides the option to compute special eco-friendly routes (as described in Section 2.2.1), this feature also includes basic live and post-drive eco-coaching. While driving, a leaf is shown in one corner of the screen, next to a score from zero to one hundred, updated every few seconds. The leaf is green when the score is high, yellow when medium, and red when very low. Mainly, the displayed values relate to how efficiently the car is assumed to be operating at the current speed, and how volatile the speed has been in the past few minutes of the drive. Sharp braking and acceleration diminish the score, while cruising at constant, moderate speed increase it.

When reaching the programmed destination, a summary of values for average and total fuel used, cost, and carbon footprint of the last trip is displayed. Moreover, a feature called ecoChallenge summarizes all of these values into a score between zero and one hundred in an attempt to stimulate the user to try to do (even) better on the next trip. For an example of these screens, see Figure 2.

Figure 2: Screenshots of Garmin's ecoRoute (left) and ecoChallenge (right) features.

The ecoRoute feature can be enhanced to display more accurate data with an accessory called ecoRoute HD[5], a dongle which attaches to the vehicle's onboard diagnostics connector and connects to the Garmin PND via bluetooth. This way, live car and particularly drivetrain data, like engine revolutions, current gear, etc. can be viewed on the PND and used in consumption computations for the ecoRoute and ecoChallenge features.

TomTom Business Solutions offers an eco-coaching product called ecoPLUS[6] to their fleet management customers. Quite similar to Garmin's ecoRoute HD, it is based on an accessory connecting to a vehicle's onboard diagnostic connector. It supports both live and post-drive feedback to the driver on his PND via bluetooth, but also the live transmission of fuel consumption and CO_2 emission data to the fleet management system's server application via GPRS. In the server application, advanced features to compute and display consumption and emission statistics are available. Different vehicles as well as different drivers or different trips can be compared, and the development of their data over time can be monitored. Available consumption and emission statistics include various average values and shares due to idling, traffic, sharp acceleration, etc. See Figure 3 for an illustration.

In a current pilot project in cooperation with the private lease company Mijndomein Auto[7], TomTom also evaluates the use of its ecoPLUS product by private drivers. Two hundred brand new Peugeot 107 vehicles are equipped with a semi-integrated TomTom PND with ecoPLUS features

[5] http://www.garmin.com/pr/ecoroutehd/
[6] http://business.tomtom.com/en_gb/products/accessories/ecoplus/
[7] http://corporate.tomtom.com/releasedetail.cfm?ReleaseID=627607

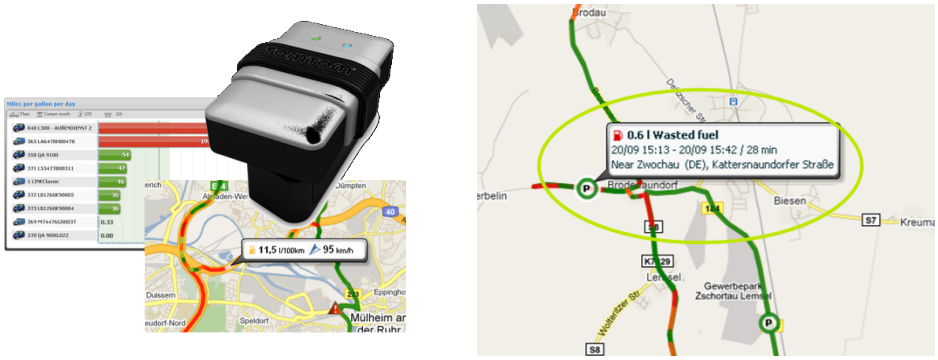


Figure 3: TomTom Business Solution's ecoPLUS product, an accessory connecting to a vehicles onboard diagnostic connector to deliver real-time fuel consumption and CO_2 emissions to a web-based server application.

and TomTom Live Services including HD Traffic, and can be leased through Mijndomein for €219 a month in the city of Amsterdam.

An interesting example of a purely post-drive eco-coaching system is **Fiat**'s eco:drive[8], a feature of the company's blue&me infotainment system. The car records detailed journey data on each trip, which is then copied onto a USB stick in the car. When plugging the USB stick to a computer with the eco:drive application, journey data is retrieved and the user can browse through various analyses, e.g. how well he did on gear shifts, braking, etc. in terms of eco-friendliness. According to the personal outcome, the systems suggests interactive video tutorials, e.g. to learn about more efficient gear shifts (see Figure 4 for a screenshot).

Figure 4: Screenshots of Fiat's eco:drive post-drive eco-coaching system; on the left, a user takes a tutorial on how to shift gears for less fuel consumption, on the right he can compare his eco-friendliness in driving within an online community.

Another important component of the eco:drive system is a social networking feature called eco:ville. Here, eco:drive users can compare their statistics and improvements in an online community, giving an additional incentive to work on their eco-friendliness.

According to a study made public by Fiat [12], covering 428,000 journeys by 5,700 drivers over 150 days in 5 countries, the average driver was able to reduce his fuel consumption by 6%, with the top 10% drivers achieving an average of even 16%. Key factors in these improvements were earlier gear changes, accounting for 31% of the savings, smoother acceleration (29%) and more efficient

[8] `http://www.fiat.com/ecodrive/`

deceleration, i.e. releasing the gas pedal earlier instead of short sharp braking (25%). The study points out that eco-coaching can reduce emissions on a level with technological innovation, but at virtually no cost.

A particularly interesting claim in the Fiat study, which also relates to our conclusions in Section 3.2, is that avoiding traffic is a major factor in reducing emissions as well. The authors claim that according to their analyses, increasing the average speed of drivers in urban areas through avoiding traffic by as little as 3 kilometers per hour, emissions could be reduced by as much as 11%.

2.6.2 Related Research Projects

There are several other ongoing research projects funded by the Seventh Framework Programme (FP7) of the European Union related to the topic of eco-coaching.

Project **eCoMove** [3] aims at developing solutions for drivers, fleet managers, and traffic managers to make route choices, driving performance, and traffic management more efficient in terms of fuel consumption, and thereby emissions. One application developed in the project is called ecoSmart Driving, comprising a dynamic green routing engine as well as live eco-coaching for the driver, providing dynamic advice for both driving and other in-vehicle systems like air conditioning, tire pressure control, etc.. A second focus of eCoMove is ecoPostTrip, a service to provide post-drive feedback on how the driver has been doing in terms of eco-friendliness, also providing this information anonymously to traffic control centers to facilitate eco-aware traffic management.

Project **ecoDriver** [2] targets a 20% reduction of CO_2 emissions and fuel consumption in road transport by fostering eco-friendly driving. The focus here is on advancing graphical and other human-machine interfaces to make feedback to the driver more tangible and compelling, helping to convince drivers of their abilities to make a difference. Among other goals, the project will explore how to maximize the acceptance of eco-coaching among European drivers, compare the effectiveness of built-in versus portable navigation devices for this purpose, and conduct a social cost-benefit analysis for such system to validate its economic feasibility.

Finally, project **EcoNav** [4], Ecological Aware Navigation and Usable Persuasive Trip Advisor for Reducing CO_2-Consumption, aims at increasing travelers' eco-friendliness by providing them with personalized multi-modal trip planning tools, presenting different alternative means of transport together with their eco-footprint in order to make users more aware of the impact of their transport decisions. One of the project's objectives is the development of an intelligent trip purpose recognition mechanism, based on real-time GPS data, to minimize the need for user interaction, thereby increasing user acceptance. It will include a dynamic user model, personalizing recommendations based on prior choices and preferences. Another goal is the development of persuasive interface strategies to visualize individuals' impact on the environment and make eco-friendly behavior visible and attractive.

2.7 Load-Balanced Vehicle Route Assignments

2.7.1 Motivation

Balancing is a typical requirement in real-world problems. Especially in urban problems, logistic companies wish to plan their transport orders in a balanced way. This can often be seen as, e.g., the result of existing business contracts, available capacities or service level quality. Nowadays balancing becomes even more attractive to logistic companies with respect to the up-coming electric vehicle fleets (eFleet). For instance, in such a case, typical balancing objectives that have to be taken into account include: the avoidance of complete discharge of individual vehicles, the balancing of the work load of the eFleet (thus achieving more flexibility), the homogeneous usage of the efleet that will allow well planned loading schedules, taking advantage of a homogeneous battery wear level of the fleet vehicles.

Although eCOMPASS focuses on conventional fleets of vehicles, the above show the importance of balancing objectives in vehicle routing problems (VRPs).

In the rest of this section, we shall review the state-of-the art approaches regarding balancing issues in VRPs with emphasis on balancing the load (or cargo) of vehicles.

2.7.2 Problem description

Recall from Section 2.3.4 that in the general (classical) vehicle routing problem (VRP) several objectives are to be taken into account with the most prominent ones being: the minimization of the total distance, the minimization of the number of tours or vehicles, and the minimization of cost.

Classical approaches (see e.g., [33]) address each objective individually. However, there is a recent shift towards the so-called *Rich* VRPs that are extended versions of the classical VRP taking various practical aspects into account (e.g., time windows, multiple depots, pick-up and delivery, VRP with backhauls, etc) – see Section 2.3.4.

In the case of balancing VRPs [80, 73, 78] the usual objectives considered are:

- Balancing of routes length
- Balancing of the workload among employees (drivers)
- Balancing of vehicle load (or cargo)

In the research literature, balancing problems have already been tackled to some extent. The most prominent studies of balancing VRPs are reviewed below.

Balancing problems in existing literature

The first study explicitly considering the notion of balanced load/cargo appeared in [80] and concerns the minimization of the total distance and the balancing the workload among employees. In particular, the objectives in [80] concern the minimization of the shortest traveled path and the best load assignment between drivers. The constraints set in [80] were the following:

- All demand points must be served within a time constraint
- There exists only the process of delivering at each node
- Each vehicle makes only one round trip
- There is a capacity constraint for each vehicle
- There is no fixed number of vehicles
- There is a unique depot

The heuristic study in [80] provides good and stable results, both w.r.t. the shortest travel distance and w.r.t. the driver load-balance in comparison to the optimal solution.

The minimization of the total length of routes and the balancing of routes length has been considered through a bi-objective approach in [73]. In that paper, a bi-objective VRP is considered with the objectives of minimizing the total route length, and also minimizing the difference between the maximal and minimal route length. This actually leads to some sort of balancing of routes. The constraints considered in [73] were the following:

- There is a capacity restriction for each vehicle
- There is a unique depot

- Each customer must be served with a quantity of goods

The solution in [73] is based on a meta-heuristic approach that approximates the Pareto set. The results of computational experiments indicate that the values for the total length objective are close to the best-known ones, and demonstrate good performance in route balanced values.

The first work that considers the balanced cargo VRP with time windows has been studied in [78]. In particular, in that paper the objective is to balance the load carried by each active vehicle, to compute the optimal number of routes and the optimal sequence of customers visited by each vehicle, while taking into account the vehicles' capacity, the service times and the time windows for delivery. This is a rich set of objectives with a couple of constraints necessary for real-life routing problems. Load balancing should generate equal travel plus service times among the whole fleet. The constraints set in [78] were:

- There is a homogeneous fleet
- There is a capacity constraint per vehicle
- There is a central depot
- There are time window constraints

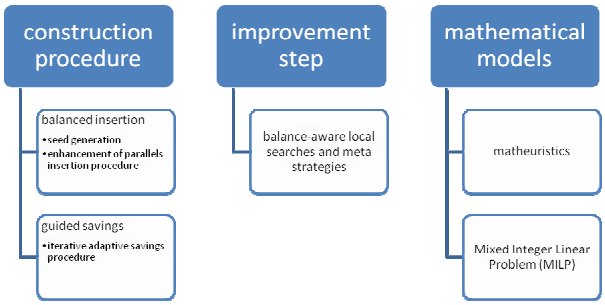
The solution approach in [78] follows the so-called data envelopment analysis (DEA) framework. First, a set of feasible routes is generated for a certain starting point, which is followed by an iterative method for selecting efficient routes. The results in [78] indicate improvements in the number of vehicles used, while using much better loaded vehicles.

Another work on cargo balancing appeared very recently in [119], and appears to be the first one that takes into account the minimization of CO_2 emissions as a central objective. This work, however, concerns multi-modal cargo transportation and hence does not appear to be relevant for urban space environments.

The approach in [119] is a heuristic one based on genetic algorithms and solves heuristically a bi-objective VRP with prime objective the minimization of the tour cost and as a secondary objective the minimization of CO_2 emissions. The obtained solution is then compared to the classical extension of Dijkstra's algorithm for solving multi-objective shortest paths.

2.7.3 Solution approaches

The generic solution approaches tacitly followed for solving VRP and/or its extended versions [33] are shown schematically in the following picture:

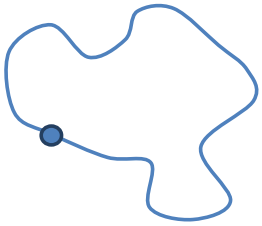


The construction procedure and the improvement step are heuristic approaches, while the mathematical models constitute exact solution approaches enhanced with relaxation, metaheuristics, or decomposition techniques. In the following, we elaborate on these approaches with a focus on those techniques that have been used so far by the eCOMPASS industrial partners.

2.7.4 Construction procedure

Route construction procedures work by inserting customers one at a time into partial routes until a feasible solution is obtained. Since several constraints exists in VRP (or in the rich VRP), the main problem is to what extent these constraints (or restrictions) will be taken into account when balanced VRP variants are studied. For instance, in the classical VRP all restrictions are taken into account, whereas restrictions taken into account for the VRP might not be mandatory for the balancing variants. The starting point for all described balancing VRPs is an opening procedure, which returns a valid solution for the tour planning problem. The next step is the local improvement procedure, e.g., through a granular tabu search (GTS). Two different construction procedures can be used: balanced insertion, or guided savings.

Balanced insertion. In this construction procedure, there is the pre-condition to have 1 or more tours where stops can be inserted.



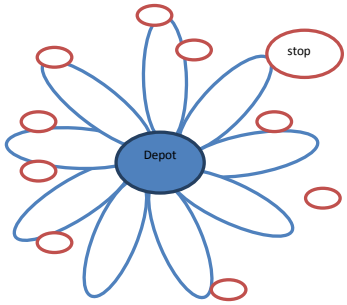
Seed generation procedure. The use of the seed generation procedure requires a given set of stops and a unique depot. Furthermore, a well-filled distance matrix for assessing the travel times is necessary along with the number of vehicles (corresponding to the number of tours) and the given number of tour parts.

The first phase of this procedure generates tours which will be reduced into different seed-tours depending on the given stop location. Thereby, pre-defined tours have to be considered due to the fact that stops of a pre-defined tour may later (at least) not lie in different tours.

The second phase of this procedure generates a giant-tour. After splitting the giant-tour in several tours, one has to choose a fitting seed tour for each individual tour. In some cases it can happen that there are some remaining tours without a stop, or there are some remaining stops without any tour.

In summary, the various steps of the seed generation procedure steps are:

- Count the number of vehicles
- Generate tours (number corresponding to the number of vehicles)
- Depending on the stop location tours are build
- Plan stops to tours
- Some tours can remain without stop
- Some stops remain without tour



Guided savings. The procedure of guided savings builds upon the standard savings heuristic procedure. After planning tours in the first step, there will be a check of the tours' balance level. The second step is the adjustment of the tour duration parameter. Adjusting this parameter generates a new number of vehicles and tours, which will be probably more balanced than the originally ones.

In summary, the various steps of the guided savings procedure are:

- Perform the standard savings procedure
- Plan tours
- Check the balance level of tours
- Adjust parameter tour duration
- Count vehicle number and count tour number

As an example, including a restriction for the tour duration might indicate that more vehicles could be needed.

The balancing procedures have two main problems: (i) The result cannot be fully controlled, e.g., we want to plan tours with 8 vehicles, but the algorithm returns solutions with 7 and 9 vehicles. (ii) There is a tendency for single spike results among groups of well-balanced tours.

2.7.5 Improvement step

Improvement heuristics iteratively improve an initial feasible solution by performing exchanges while maintaining feasibility. The process normally stops when no further exchange can be made without deteriorating the solution.

For the VRPs the target function is the reduction of distance. Tour number and vehicle number remain. Thus, the optimization concerns the reduction of the amount of distance to be driven by the vehicles. This post optimization does not allow unnecessary tour distance extensions to balanced tours.

Optimal solutions often cannot be determined efficiently. Heuristics offer valid but in general not optimal solutions. For that case it is often helpful to use a local search algorithm which is, according to our problem, balance-aware. With the help of this procedure the determination of a good-performing solution within a subset of tours is possible by investigating the neighborhood starting from a valid solution. The balance-aware local search algorithm is a procedure of post optimization.

2.7.6 Mathematical models

Matheuristics

Matheuristics are optimization algorithms made by the inter-operation of metaheuristics and mathematical programming (MP) techniques. An essential feature is the exploitation in some part of the algorithms of features derived from the mathematical model of the problems of interest.

Metaheuristic algorithms and frameworks, such as tabu-search, genetic algorithms, VNS, etc., were in fact usually proposed in years when Mixed Integer Programming (MIP) was seldom a viable option for solving real-world problem instances, or significant sub-problems thereof. However, research on mathematical programming, and in particular on discrete optimization, has led to a state of the art where MIP solvers or customized MP codes can be effective even in a heuristic context, both as primary solvers or as sub-procedures.

Mixed Integer Linear Problems

Mixed-integer linear programming (MILP) covers problems that contain continuous as well as integer decision variables (x and y in the following description). The objective function has to be a linear expression (therefore, terms such as x^2 or $\max\{x_1, x_2\}$ cannot appear in it). The problem is subject to linear equality or linear inequality constraints.

The following chart (cf. [109, p. 59]) shows the canonical form of a linear problem (LP) and a MILP (note that equality and inequality constraints can be converted into each other):

LP

maximize	$c^T x$
subjec to	$Ax \leq b$
	$x \geq 0$

MILP

maximize	$c^T x + d^T y$
subject to	$Ax + By \leq b$
	$x \in \mathbb{N}_0$
	$y \geq 0$

Here, c and d are coefficient vectors of the objective function. A and B are the coefficient matrices of the constraints. b is the vector of all right sides of the constraints. In the MILP all x variables can only take on integer values greater than or equal to 0. All values of y only have to be greater than or equal to 0.

3 New Prospects for Route Planning in Urban Areas

When considering eco-aware, robust routes for individual drivers and fleets, along with possible meaningful alternatives, two possible perspectives are the following:

Pre-Trip Recommendation: The goal is to provide an overview of all relevant alternatives with route metrics, statistical information of traffic, indicators of robustness and eco-awareness of routes; also to convey trade-off functions that the drivers might use in real-time for making up their decisions for robustness and/or eco-awareness.

During-Trip Recommendation: At precomputed decision points, defined from a set of route alternatives, the goal is to provide the driver with up-to-the-minute information on his alternatives (current traffic patterns, indicators of robustness and eco-friendliness) so as to evaluate the trade-offs and make an informed and conscious decision.

The main axes of our research within WP2 should focus on the following:

(i) Consideration of the appropriate *traffic prediction models* (and methods to dynamically update them), given on the real traffic patterns as measured by the navigation system itself. A crucial issue in this direction is how to cope with traffic incident dependencies. The obvious approach is to consider all traffic incidents as statistically independent, but the truth is that in reality they are (at least locally) dependent.

(ii) Investigation of appropriate models to explicitly account for various *measures of appropriateness/likelihood* of routes, such as fuel consumption, travel time, environmental footprint, route likeliness, for either individually moving vehicles or entire fleets of vehicles. The goal is to have – rather than fixed trade-offs between these measures – trade-off functions to be evaluated in real-time, given not only the traffic predictions, but also the last-minute traffic patterns communicated to the navigation devices by the navigation system's master control.

(iii) Consideration of the *temporal aspects* of the route planning parameters, e.g., by allowing time-dependent functions characterizing the various measures of efficiency for road segments.

(iv) Assurance of *robustness* and/or *elasticity* of the proposed solutions, by either the potential of recovering a temporally lost connection, or the provision of prior or real-time reconsiderations of the drivers' decisions via (subgraphs of) alternative routes. The focus should either be on routes that are "most unlikely to be congested", or on routes that come with "sufficient detours available".

The following subsections present our main prospects for future research within eCOMPASS, related to route planning for both private cars and fleets of vehicles, in accordance with the relevant state-of-art presented in Section 2.

3.1 Route Planning Under Uncertainty

Techniques for route planning under uncertainty, as those presented in Section 2.1, usually rely on the idea that it is possible to devise good solutions for a future instance based on a model for data from the past. When considering private vehicles in urban environments, it is near impossible to derive an exact model for unexpected disturbances. For example, traffic jams may not only happen during the rush hours, they may also be induced by accidents that are impossible to predict. We cannot hope to devise a universal model that is able to predict such situations. The situation becomes even more complicated when travel time is not the only parameter that we wish to optimize when planning a route. This is particularly the case if we strive to minimize the environmental impact of the chosen route.

These issues are evident when looking at typical features of standard techniques to handle uncertainty in shortest path problems. In worst-case optimization, for example, we usually consider a restricted set of all possible realizations, and we look for solutions that are acceptable for this set only. In this situation, a natural question is how such a set can be chosen such that it is representing all likely situations. For example, should we choose to capture the variation of possible delays of a road network by using snapshots of past traffic data? When is it better to use interval ranges for edge lengths? To properly answer these questions, we need to have a detailed understanding of the network in which our approach will be applied. We have to provide a justification of why the chosen model is more suitable than others. Such a justification will generally not be easy to obtain for real-world situations. Altogether, worst-case optimization tends to be overly conservative, and it usually leads to solutions that are expensive, just to be acceptable even in the worst case.

Stochastic optimization suffers from similar issues. An exact probability distribution for the uncertainty in the data is usually near impossible to obtain, therefore simplifying assumptions have to be made. As before, it is hard to understand what assumptions we can make without altering the nature of the problem significantly. Why did we choose discrete random distribution to model edge costs? When is it better to use continuous distributions? How can we handle the absence of a connection (for example, due to road work) without affecting the computed route? In real-world settings, we do not have a sufficiently detailed description of the environment that allows us to effectively judge when the chosen model is the most suitable.

The same issue arises in models where some parameters are used to control the robustness of a solution. We need to tune the parameters according to some heuristic in order to find acceptable compromises between quality and robustness of a solution. None of the known techniques related to route planning under uncertainty come with a self-assessment of the produced solution: they assure good results only with respect to the given input, and not with respect to a future reality.

Our aim is to develop a novel method to compute routes under uncertainty while avoiding these issues. In particular, we want to be able to exploit the information contained in the historical data *without* making any assumptions regarding the reality that generated the data. We believe that this will allow us to compute routes that are *provably* good, together with a measure of confidence that the solution will be good for future instances from the same generator.

To achieve this goal, we need to address properties specific to route planning of private vehicles in urban environments. That is, we must be able to provide routes that are acceptable also with respect to the users experience. Typically, an experienced driver will not accept a route that is very different from the historical standard. We have to balance the goal to obtain best-possible results with what the user is willing to accept. This may lead to multi-criteria optimization. We may try to compute "alternatives" to the routes that are not too far from those common to the user.

Multi-criteria optimization will be necessary also if we are asked to find routes that are not only fast, but also minimize environmental impact. For shortest paths, optimizing a bicriteria objective function composed of "eco-friendliness" and travel time, can generally be a hard task. Minimizing the environmental footprint might also lead to objective functions that are non-additive. For example, a long trip on a highway usually has much less impact on the environment than returning to the highway many times for short segments. For many situations like this, no efficient solutions are known and we will have to design novel approaches and specialized solutions.

3.2 Eco-Aware Cost Models for Routing

From Sections 2.2.1 and 2.2.2, a few challenges in developing an eco-aware cost model for routing in eCOMPASS become apparent. As confirmed by both the real-life test in [121] and the more theoretic experiments in [95], optimizing routes solely for fuel-consumption leads to quite undesirable routes which would not be accepted by users. Hence, a proper trade-off between the eco-friendliness, the travel time and convenience of a route has be found.

It is also obvious that, in order to accurately regard the fuel-consumption of a vehicle for routing, quite elaborate data from the vehicle needs to be collected. Different vehicles react very differently

to different road attributes, e.g., more powerful cars are most likely less sensitive to road elevation. Unfortunately, it is unclear how such data can be obtained for eCOMPASS applications, as they will run on a portable device without direct access to the car's computer system and data.

Moreover, drivers' sensitivity to road attributes can vary as well. While some use every opportunity to accelerate to speed limit, others exhibit a smoother driving style. Hence, a certain self-learning behavior of the cost-model seems mandatory to adapt to these differences in driving behavior.

In the quest for an eco-aware routing cost model, there are a number of clear opportunities and possibilities for eCOMPASS. The lack of access to detailed car data due to the nature of a portable device can possibly be compensated by the use of an accelerometer, as most smartphones have it today. Some very promising research in this area has recently been conducted in the CO2GO project at the Massachusetts Institute of Technology [1]. Here, a stock smartphone is used to sense a user's current mode of transportation solely by use of the phone's accelerometer. Also, an advanced set of vehicle settings in an application could be used to enable a more precise understanding of the vehicle's consumption behavior in routing.

Even in the absence of car-specific emission models, there is an opportunity for algorithmic improvements. Building on the approach reviewed in Section 2.2.2, we aim for more detailed emission modeling and corresponding algorithms. These take into account speed variance along a route and real-time traffic conditions. We want to employ realistic modeling for inner city route planning that takes into consideration turn costs, right-of-way, and traffic lights for full cities (to the extent that data is available, e.g., from OSM or governmental sources). We aim to assign not only *one* velocity (and thus a single emission value) to each edge but to evaluate all Pareto-optimal emission/travel-time pairs, that is, the emissions at several speeds on the same road segment. In the presence of traffic predictions this can quickly lead to complex scenarios where it might be beneficial to drive faster at first in order to avoid traffic later on, saving emissions despite the higher speed. Our algorithmic approaches will reflect these considerations, and we will explore the exact nature of such complex models in the course of the project.

Finally, a very important fact regarding eco-aware routes was revealed in [121] as detailed in Section 2.2.1: In many situations, and in particular in dense urban areas, the most successful and most user-friendly way to reduce emissions might be to avoid traffic as effectively as possible. Consequently, advanced routing cost models which take into account the likeliness of congestion in certain parts of the road network, and routing engines that are able to react to traffic information in real-time, constitute a somewhat different but very promising variant of an eco-aware routing cost model as well.

Eco-aware route plans in time-dependent networks. As we saw in Section 2.2.3, the main idea in the presented approach was the use of a fuel consumption function B which is integrated in the travel cost function β of the time-dependent network. Then the path cost function b is computed, which is the sum of the travel cost function β plus the travel time function τ. Then, two algorithms DOT* and TD-APX were presented. The DOT* algorithm finds an exact optimal solution while the TD-APX approximates the optimal solution. From the experimental evaluation, the DOT* algorithm takes a lot of time to find an optimal solution. The use of piecewise linear functions reduces the time needed to find an optimal solution by an order of magnitude, but still the algorithm is far too slow for applications where there is a need to compute fast fuel-optimal paths fast. The behavior of TD-APX is better and can be used to compute fast almost fuel-optimal paths. Another advantage of TD-APX is that large time frames can be considered.

In the quest for better approaches, we plan to proceed along two main axes: (i) we shall investigate the possible extension of the currently best state-of-the-art approaches in order to become more efficient and suitable for real-time applications; (ii) we shall investigate the adaptation of speed-up techniques that work well in other contexts (e.g., when edge costs represent travel times or distances) to the case of computing fuel-optimal paths.

3.3 Time-dependent Multi-criteria Route Planning

All theoretically established results concerning generic techniques for fast shortest path calculations in sparse graphs, as those mentioned in section 2.3.1, mainly focus on static (time-independent) networks. Nevertheless, there is now a clear picture on how the trade-off between preprocessing time/space and query time behaves in these networks. We shall explore how similar trade-offs behave in the case of time-dependent networks. That is, our goal is to provide mainly approximate distance oracles for time-dependent networks, that divide the cost/space between a (typically expensive) preprocessing phase and a (preferably fast) query phase, and look for the best possible values for these trade-offs. The main question to investigate is how the time-dependence of the edge-costs affects these trade-offs. Various techniques will be investigated theoretically, and the most mature of them will possibly be tested and also compared experimentally.

Similarly, from our review of applied preprocessing techniques for road networks in Section 2.3.2, it is clear that the main body of research in this field has focused on static time-independent networks with travel-time metric, that is, uni-criterial optimization for fastest routes without concern for, e.g., rush hours (time-dependency) or traffic reports (dynamic networks). For eCOMPASS we plan to extend these results in several dimensions: emission metric, dynamics, multi-criteria optimization, and time-dependency. Since we aim to optimize for emissions as well as for travel-time we will evaluate known preprocessing techniques on how well they perform for emission-efficient route planning, i.e., how hard the preprocessing is on an a emission-based metric, what level of speedup can be achieved on that metric, and how well each technique can be adapted to more involved optimization scenarios (cf. Section 2.2.2).

As eCOMPASS aims to account for real-time traffic conditions (in order to avoid congested areas ensuring fast, energy-efficient and environmentally-friendly travel) our algorithms must perform well on dynamic networks where travel costs (e. g. emissions, travel-time, incident likeliness) frequently change. Most of the preprocessing techniques reviewed in Section 2.3.2 do not behave well in such dynamic scenarios. Those techniques that do perform reasonably on dynamic time-independent networks will be adapted by us to the more realistic, dynamic multi-criteria time-dependent scenarios considered in WP2. Success of these adaptations will be evaluated experimentally and validated during the pilot. In order to achieve this extension we will build on algorithmic ideas from the techniques reviewed in Section 2.3.3 and Section 2.4. No multi-criteria technique that we are aware of has so far been tested on time-dependent criteria. This is relevant to us, since some criteria of our setting (e.g., eco-friendliness, travel-time) are both time-dependent. We expect that the faster of the two reviewed techniques [56] will be seriously hampered in such a scenario. One of our next steps regarding multi-criteria route planning will be to experimentally evaluate this assumption. Based on this evaluation we will proceed to propose alterations and work on improvements.

While several adaptations of time-independent preprocessing techniques to the time-dependent route planning scenario exist (see Section 2.4), the trade-off between preprocessing time, space and query performance is worse than for the original scenario. Those techniques that allow for reasonably fast queries require long preprocessing of several hours and do not allow for quick updates of the graph in case of traffic delays. This is inadequate for our more realistic scenarios. We plan to improve on this situation by two means: by transferring results from Foschini et al. [54] and by leveraging latest results on preprocessing-light techniques for time-independent shortest paths. This way, we aim to greatly improve performance for time-dependent scenarios closing the gap between very rich results on time-independent route planning on the one hand and more realistic scenarios such as multi-criteria and time-dependent route planning on the other hand.

Finally, another goal of eCOMPASS is the integration of route planning for road networks with public transport ("park & ride") in order to save on emissions or travel-time. Techniques discussed in this review are tailored to unimodal, purely road scenarios. We will extended this to "park & ride"-type multi-modal scenarios based on recent algorithmic progress in the respective research field. For details see Deliverable D3.1.

3.4 Alternative & Robust Routes

The proposed approaches in [6] and [13] will be the basis of our research. Even though these approaches have different objectives, they share the same basic concepts. Our goal is to extend the core methods of these approaches by using different speed-up techniques of Dijkstra's algorithm.

The first step is the identification of the major drawbacks of the reviewed approaches and the attempt to remedy them using known techniques. Since there is an interest in deploying the eCOMPASS services on mobile devices, the performance of the algorithms is of critical importance. Any proposed techniques must be able to compute alternative routes in real time.

By examining the evaluation results in [85], it seems that a performance speed-up can be achieved by extending the basic idea behind [6] using different speed-up techniques based on Dijkstra's algorithm. Unfortunately, the main concept behind most of Dijkstra's speed-up techniques, which is reducing the search space of the algorithm, is counter-productive in this case. This is because alternative routes are omitted in purpose from the search space of Dijkstra's algorithm. The quality degradation is apparent in the performance evaluation in [6], where algorithms developed using the *Contraction Hierarchies* speed-up technique fail to identify alternative routes in almost half the cases. Hence, our primary goal is to evaluate and identify the best speed-up techniques of Dijkstra's algorithm providing adequate alternative routes' quality, while having a speed performance that compares favorably with real time applications.

Another approach would be the identification of alternative routes using the *penalty* method described in [13]. This approach considers dynamic changes of the edge weights of the graph, in order to omit temporarily the shortest path from the search space. In order to use this approach efficiently, speed-up techniques using preprocessing information capable of changing dynamically must be employed. Our goal is to evaluate the existing techniques and extend previously static techniques to a dynamic scenario.

For the problems studied in eCOMPASS, good alternative route graphs should include robust options, that is, routes that are unlikely to incur traffic, together with a rich set of detours which allow a driver to recover from unexpected traffic incidents. The objectives used in previous research in order to determine the quality of good alternative routes do not account for robustness. E.g., both [6] and [13] maximize the disjointness of alternative paths (by either minimizing overlaps or maximizing total distance under limited average distance) — but disjoint paths do not provide any recovery options. Hence, in the eCOMPASS project we will examine new formulations that allow for the computation of *robust* alternatives.

The new approaches that will be developed will be used for the computation of both alternative routes and alternative graphs. Apart from the new speed-up techniques, our research will be twofold. Firstly, the quality of the new speed-up techniques will be evaluated and checked against real-life situations using the alternative graph approach in [13]. Secondly, their performance will be evaluated using the selection techniques described in [6].

It is finally noted that there seems to be an overlap with Section 3.1 w.r.t. the overall goal, but the approaches proposed are very different. We will carefully explore possibilities to share algorithmic techniques between both approaches. E.g., it might be worthwhile to use methods proposed in this section in order to accelerate computations in Section 3.1 by first extracting a rich but much smaller "alternative" subgraph.

3.5 Driver Eco-Coaching

As is obvious from the vital activity in both the navigation market and research community on the topic, eco-coaching is widely perceived as an important topic in future trip planning and driver support applications and services. As Fiat's researchers also comment in [12], great attractiveness of the approach lies in the fact that it can potentially reduce emissions on the level of significant technological advancement at a fraction of the cost. Moreover, the possibility to include a social

networking component in post-drive coaching can create a great additional incentive for eco-friendly behavior, potentially viralizing the attractiveness of eco-coaching.

The eCOMPASS specific approach of delivering services and application on familiar portable devices leads to both distinct challenges and opportunities. On the one hand, the lack of access to detailed online car data might make it harder to evaluate driving behavior in great detail. On the other hand, portable devices are perfect to deliver immediate post-drive feedback, e.g., while walking from a parking spot to the final destination. Also, portable devices are especially suitable to convey multi-modal transport options for a trip before the user gets in his car.

Finally, the lack of direct access to detailed car data could be compensated in various ways. For one, additional sensors common in today's smartphones and other mobile devices, like an accelerometer can be used to collect data regarding driving style or transport mode. A research project called CO2GO [1] at the Massachusetts Institute of Technology currently demonstrates possibilities in this direction. See also Section 3.2 of this document. Finally, portable devices provide the unique possibility to switch transportation modes on the go, while maintaining continuous feedback on the eco-friendliness of a trip. E.g., a driver feedback application might be used on the first leg of a trip (while driving a car), and a continuous reassuring feedback on eco-friendliness could be provided after changing over to public transportation at a "park & ride" terminal, creating a truly seamless eco-coaching experience.

3.6 Load Balanced Vehicle Route Assignments

Based on the so far existing approaches for load-balanced VRPs discussed in Section 2.7, we plan to proceed along two main directions.

First, we shall investigate the case of a bi-objective approach, where the first objective function is the balancing of the load/cargo while the second objective function will be the minimization of the number of tours. The first objective function (balancing of the load/cargo) leads indirectly to an improvement in the number of vehicles used and guarantees equal time travels for each vehicle. The second objective function (minimization of the number of tours) is environmentally-friendly, since fewer tours result to fewer emissions from the vehicles.

Second, we shall investigate the case of a multi-objective approach where the objective functions are: balancing of cargo, minimization of the number of tours, minimization of cost, balancing of the workload among drivers. Then, contrary to previous approaches that mainly use heuristics for the solution of such problems, we shall pursue a lexicographic approach to compute the Pareto set by assigning priorities to each function.

In both research directions, protection of the environment is accomplished by balancing the load/cargo in every vehicle and by minimizing the number of tours needed to serve all customers.

References

[1] Co2go project. SENSEable City Lab of the Massachusetts Institute of Technology, `http://senseable.mit.edu/co2go/`.

[2] ecodriver - supporting the driver in conserving energy and reducing emissions. `http://www.ertico.com/ecodriver`.

[3] ecomove - cooperative mobility systems and services for energy efficiency. `http://www.ecomove-project.eu/`.

[4] Econav - ecological aware navigation: Usable persuasive trip advisor for reducing co2-consumption. `http://www.econav-project.eu/`.

[5] Ittai Abraham, Daniel Delling, Amos Fiat, Andrew V. Goldberg, and Renato F. Werneck. VC-Dimension and Shortest Path Algorithms. In *Proceedings of the 38th International Colloquium on Automata, Languages, and Programming (ICALP'11)*, volume 6755 of *Lecture Notes in Computer Science*, pages 690–699. Springer, 2011.

[6] Ittai Abraham, Daniel Delling, Andrew V. Goldberg, and Renato F. Werneck. Alternative Routes in Road Networks. In Festa [53], pages 23–34.

[7] Ittai Abraham, Daniel Delling, Andrew V. Goldberg, and Renato F. Werneck. A Hub-Based Labeling Algorithm for Shortest Paths on Road Networks. In Pardalos and Rebennack [102], pages 230–241.

[8] Ittai Abraham, Amos Fiat, Andrew V. Goldberg, and Renato F. Werneck. Highway Dimension, Shortest Paths, and Provably Efficient Algorithms. In Moses Charikar, editor, *Proceedings of the 21st Annual ACM–SIAM Symposium on Discrete Algorithms (SODA'10)*, pages 782–793. SIAM, 2010.

[9] INFRAS AG. Handbook of Emission Factors for Road Transport 2.1, February 2004.

[10] Ravindra K. Ahuja, Thomas L. Magnanti, and James B. Orlin. *Network flows: theory, algorithms, and applications*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1993.

[11] James Aspnes. Competitive analysis of distributed algorithms. In Amos Fiat and Gerhard Woeginger, editors, *Online Algorithms*, volume 1442 of *Lecture Notes in Computer Science*, pages 118–146. Springer Berlin / Heidelberg, 1998. 10.1007/BFb0029567.

[12] Fiat Group Automobiles. Eco-driving uncovered.
`http://www.fiat.co.uk/uploadedFiles/Fiatcouk/Stand_Alone_Sites/EcoDrive2010/ECO-DRIVING_UNCOVERED_full_report_2010_UK.pdf`.

[13] Roland Bader, Jonathan Dees, Robert Geisberger, and Peter Sanders. Alternative Route Graphs in Road Networks. In Alberto Marchetti-Spaccamela and Michael Segal, editors, *Proceedings of the 1st International ICST Conference on Theory and Practice of Algorithms in (Computer) Systems (TAPAS'11)*, volume 6595 of *Lecture Notes in Computer Science*, pages 21–32. Springer, 2011.

[14] Amotz Bar-Noy and Baruch Schieber. The canadian traveller problem. In *Proceedings of the second annual ACM-SIAM symposium on Discrete algorithms*, SODA '91, pages 261–270, Philadelphia, PA, USA, 1991. Society for Industrial and Applied Mathematics.

[15] Yair Bartal, Lee-Ad Gottlieb, Tsvi Kopelowitz, Moshe Lewenstein, and Liam Roditty. Fast, precise and dynamic distance queries. In *Proc. of 22nd ACM-SIAM Symp. on Discr. Alg. (SODA '11)*, pages 840–853. SIAM, 2011.

[16] Hannah Bast. Car or Public Transport – Two Worlds. In Susanne Albers, Helmut Alt, and Stefan Näher, editors, *Efficient Algorithms*, volume 5760 of *Lecture Notes in Computer Science*, pages 355–367. Springer, 2009.

[17] Holger Bast, Stefan Funke, Domagoj Matijevic, Peter Sanders, and Dominik Schultes. In Transit to Constant Shortest-Path Queries in Road Networks. In *Proceedings of the 9th Workshop on Algorithm Engineering and Experiments (ALENEX'07)*, pages 46–59. SIAM, 2007.

[18] Holger Bast, Stefan Funke, Peter Sanders, and Dominik Schultes. Fast Routing in Road Networks with Transit Nodes. *Science*, 316(5824):566, 2007.

[19] 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.

[20] Gernot Veit Batz, Robert Geisberger, Sabine Neubauer, and Peter Sanders. Time-Dependent Contraction Hierarchies and Approximation. In Festa [53], pages 166–177.

[21] Reinhard Bauer, Tobias Columbus, Bastian Katz, Marcus Krug, and Dorothea Wagner. Preprocessing Speed-Up Techniques is Hard. In *Proceedings of the 7th Conference on Algorithms and Complexity (CIAC'10)*, volume 6078 of *Lecture Notes in Computer Science*, pages 359–370. Springer, 2010.

[22] Reinhard Bauer, Gianlorenzo D'Angelo, Daniel Delling, and Dorothea Wagner. The Shortcut Problem – Complexity and Approximation. In *Proceedings of the 35th International Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM'09)*, volume 5404 of *Lecture Notes in Computer Science*, pages 105–116. Springer, January 2009.

[23] Reinhard Bauer and Daniel Delling. SHARC: Fast and Robust Unidirectional Routing. *ACM Journal of Experimental Algorithmics*, 14(2.4):1–29, August 2009. Special Section on Selected Papers from ALENEX 2008.

[24] Reinhard Bauer, Daniel Delling, Peter Sanders, Dennis Schieferdecker, Dominik Schultes, and Dorothea Wagner. Combining Hierarchical and Goal-Directed Speed-Up Techniques for Dijkstra's Algorithm. *ACM Journal of Experimental Algorithmics*, 15(2.3):1–31, January 2010. Special Section devoted to WEA'08.

[25] Richard Bellman. On a Routing Problem. *Quarterly of Applied Mathematics*, 16:87–90, 1958.

[26] Shai Ben-David and Allan Borodin. A new measure for the study of on-line algorithms. *Algorithmica*, 11(1):73–91, 1994.

[27] Dimitri P. Bertsekas, Angelia Nedić, and Asuman E. Ozdaglar. *Convex analysis and optimization*. Athena Scientific optimization and computation series. Athena Scientific, 2003.

[28] Dimitri P. Bertsekas and John N. Tsitsiklis. An analysis of stochastic shortest path problems. *Mathematics of Operations Research*, 16:580–595, 1991.

[29] Dimitris Bertsimas and Melvyn Sim. Robust discrete optimization and network flows. *Mathematical Programming*, 98:49–71, 2003.

[30] Allan Borodin and Ran El-Yaniv. *Online computation and competitive analysis*. Cambridge University Press, New York, NY, USA, 1998.

[31] Christina Büsing. Recoverable robust shortest path problems. *Networks*, 59(1):181–189, 2012.

[32] Sergio Cabello. Many distances in planar graphs. *Algorithmica*, 62(1-2):361–381, 2012. Preliminary version in ACM-SIAM SODA 2006.

[33] Jean-Francois Cordeau, Gilbert Laporte, Martin Savelsbergh, and Daniele Vigo. Vehicle Routing. In C. Barnhart and G. Laporte, editors, *Handbook in OR & MS*, volume 14, chapter 6, pages 367–428. Elsevier, 2007.

[34] D. Delling, A.V. Goldberg, I. Razenshteyn, and R.F. Werneck. Graph partitioning with natural cuts. In *In Proc. of 25th International Parallel and Distributed Processing Symposium (IPDPS '11)*. IEEE Computer Society, 2011.

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

[36] Daniel Delling, Andrew V. Goldberg, Andreas Nowatzyk, and Renato F. Werneck. PHAST: Hardware-accelerated shortest path trees. *Journal of Parallel and Distributed Computing*, 2012. To appear.

[37] Daniel Delling, Andrew V. Goldberg, Thomas Pajor, and Renato F. Werneck. Customizable Route Planning. In Pardalos and Rebennack [102], pages 376–387.

[38] Daniel Delling, Andrew V. Goldberg, Ilya Razenshteyn, and Renato F. Werneck. Graph Partitioning with Natural Cuts. In *25th International Parallel and Distributed Processing Symposium (IPDPS'11)*, pages 1135–1146. IEEE Computer Society, 2011.

[39] Daniel Delling, Andrew V. Goldberg, and Renato F. Werneck. Shortest Paths in Road Networks: From Practice to Theory and Back. *it—Information Technology*, 53:294–301, December 2011.

[40] Daniel Delling, Martin Holzer, Kirill Müller, Frank Schulz, and Dorothea Wagner. High-Performance Multi-Level Routing. In Demetrescu et al. [46], pages 73–92.

[41] Daniel Delling, Moritz Kobitzsch, Dennis Luxen, and Renato F. Werneck. Robust Mobile Route Planning with Limited Connectivity. In *Proceedings of the 14th Meeting on Algorithm Engineering and Experiments (ALENEX'12)*. SIAM, 2012. To appear.

[42] Daniel Delling and Giacomo Nannicini. Core Routing on Dynamic Time-Dependent Road Networks. *Informs Journal on Computing*, 2012. Journal version of ISAAC'08. To appear.

[43] Daniel Delling, Peter Sanders, Dominik Schultes, and Dorothea Wagner. Engineering Route Planning Algorithms. In Jürgen Lerner, Dorothea Wagner, and Katharina A. Zweig, editors, *Algorithmics of Large and Complex Networks*, volume 5515 of *Lecture Notes in Computer Science*, pages 117–139. Springer, 2009.

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

[45] Daniel Delling and Dorothea Wagner. Time-Dependent Route Planning. In Ravindra K. Ahuja, Rolf H. Möhring, and Christos Zaroliagis, editors, *Robust and Online Large-Scale Optimization*, volume 5868 of *Lecture Notes in Computer Science*, pages 207–230. Springer, 2009.

[46] 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.

[47] Edsger W. Dijkstra. A Note on Two Problems in Connexion with Graphs. *Numerische Mathematik*, 1:269–271, 1959.

[48] Hristo Djidjev. Efficient algorithms for shortest path queries in planar digraphs. *Graph-Theoretic Concepts in Computer Science, Lecture Notes in Computer Science, Springer-Verlag*, LNCS1197:51–165, 1997.

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

[50] Richard Eglese and Jose Brandao. A deterministic tabu search algorithm for the capacitated arc routing problem. 2005.

[51] Jittat Fakcharoenphol and Satish Rao. Planar graphs, negative weight edges, shortest paths, and near linear time. *J. Comp. Sys. Sci.*, 72:868–889, 2006.

[52] Y. Fan, R. Kalaba, and J. Moore. Arriving on time. *Journal of Optimization Theory and Applications*, 127:497–513, 2005.

[53] Paola Festa, editor. *Proceedings of the 9th International Symposium on Experimental Algorithms (SEA'10)*, volume 6049 of *Lecture Notes in Computer Science*. Springer, May 2010.

[54] Luca Foschini, John Hershberger, and Subhash Suri. On the complexity of time-dependent shortest paths. In *Proc. of 22nd ACM-SIAM Symp. on Discr. Alg. (SODA '11)*, pages 327–341. ACM-SIAM, 2011.

[55] Greg N. Frederickson. Fast algorithms for shortest paths in planar graphs, with applications. *SIAM J. on Comp.*, 16:1004–1022, 1987.

[56] Robert Geisberger, Moritz Kobitzsch, and Peter Sanders. Route Planning with Flexible Objective Functions. In *Proceedings of the 12th Workshop on Algorithm Engineering and Experiments (ALENEX'10)*, pages 124–137. SIAM, 2010.

[57] Robert Geisberger, Peter Sanders, Dominik Schultes, and Daniel Delling. Contraction Hierarchies: Faster and Simpler Hierarchical Routing in Road Networks. In Catherine C. McGeoch, editor, *Proceedings of the 7th Workshop on Experimental Algorithms (WEA'08)*, volume 5038 of *Lecture Notes in Computer Science*, pages 319–333. Springer, June 2008.

[58] Robert Geisberger, Peter Sanders, Dominik Schultes, and Christian Vetter. Exact routing in large road networks using contraction hierarchies. *Transportation Science*, 2011. Accepted for publication.

[59] Andrew V. Goldberg. A Practical Shortest Path Algorithm with Linear Expected Time. *SIAM Journal on Computing*, 37:1637–1655, 2008.

[60] Andrew V. Goldberg and Chris Harrelson. Computing the Shortest Path: A* Search Meets Graph Theory. In *Proceedings of the 16th Annual ACM–SIAM Symposium on Discrete Algorithms (SODA'05)*, pages 156–165. SIAM, 2005.

[61] Andrew V. Goldberg, Haim Kaplan, and Renato F. Werneck. Reach for A*: Shortest Path Algorithms with Preprocessing. In Demetrescu et al. [46], pages 93–139.

[62] Andrew V. Goldberg and Renato F. Werneck. Computing Point-to-Point Shortest Paths from External Memory. In *Proceedings of the 7th Workshop on Algorithm Engineering and Experiments (ALENEX'05)*, pages 26–40. SIAM, 2005.

[63] Ronald J. Gutman. Reach-Based Routing: A New Approach to Shortest Path Algorithms Optimized for Road Networks. In *Proceedings of the 6th Workshop on Algorithm Engineering and Experiments (ALENEX'04)*, pages 100–111. SIAM, 2004.

[64] P. Hansen. Bricriteria Path Problems. In Günter Fandel and T. Gal, editors, *Multiple Criteria Decision Making – Theory and Application –*, pages 109–127. Springer, 1979.

[65] Peter E. Hart, Nils Nilsson, and Bertram Raphael. A Formal Basis for the Heuristic Determination of Minimum Cost Paths. *IEEE Transactions on Systems Science and Cybernetics*, 4:100–107, 1968.

[66] Monika Henzinger, Philip Klein, Satish Rao, and Sairam Subramanian. Faster shortest-path algorithms for planar graphs. *J. Comp. Sys. Sci.*, 55:3–23, 1997.

[67] Moritz Hilger, Ekkehard Köhler, Rolf H. Möhring, and Heiko Schilling. Fast Point-to-Point Shortest Path Computations with Arc-Flags. In Camil Demetrescu, Andrew V. Goldberg, and David S. Johnson, editors, *9th DIMACS Implementation Challenge - Shortest Paths*, November 2006.

[68] Moritz Hilger, Ekkehard Köhler, Rolf H. Möhring, and Heiko Schilling. Fast Point-to-Point Shortest Path Computations with Arc-Flags. In Demetrescu et al. [46], pages 41–72.

[69] Martin Holzer, Frank Schulz, and Dorothea Wagner. Engineering Multi-Level Overlay Graphs for Shortest-Path Queries. *ACM Journal of Experimental Algorithmics*, 13(2.5):1–26, December 2008.

[70] Martin Holzer, Frank Schulz, and Thomas Willhalm. Combining Speed-up Techniques for Shortest-Path Computations. In *Proceedings of the 3rd Workshop on Experimental Algorithms (WEA'04)*, volume 3059 of *Lecture Notes in Computer Science*, pages 269–284. Springer, 2004.

[71] Nicolas Jozefowiez, Frederic Semet, and El-Ghazali Talbi. From single-objective to multi-objective vehicle routing problems: Motivations, case studies, and methods. 2008.

[72] Nicolas Jozefowiez, Frederic Semet, and El-Ghazali Talbi. Multi-objective vehicle routing problems. *European Journal of Operational Research*, 189:293–309, 2008.

[73] Nicolas Jozefowiez, Frederic Semet, and El-Ghazali Talbi. An evolutionary algorithm for the vehicle routing problem with route balancing. *European Journal of Operational Research*, 195(3):761–769, 2009.

[74] Kenichi Kawarabayashi, Philip Nathan Klein, and Christian Sommer. Linear-space approximate distance oracles for planar, bounded-genus, and minor-free graphs. In *Proc. of 38th Int. Col. on Aut., Lang. and Progr. (ICALP '11)*, LNCS 6755, pages 135–146. Springer, 2011.

[75] Philip N. Klein. Multiple-source shortest paths in planar graphs. In *Proc. of 16th ACM-SIAM Symp. on Discr. Alg. (SODA '05)*, pages 145–155, 2005.

[76] S. Kluge. *On the computation of fuel-optimal paths in time-dependent networks.* PhD Thesis, 2010.

[77] Ekkehard Köhler, Rolf H. Möhring, and Heiko Schilling. Acceleration of Shortest Path and Constrained Shortest Path Computation. In *Proceedings of the 4th Workshop on Experimental Algorithms (WEA'05)*, volume 3503 of *Lecture Notes in Computer Science*, pages 126–138. Springer, 2005.

[78] Manolis N. Kritikos and George Ioannou. The balanced cargo vehicle routing problem with time windows. *International Journal of Production Economics*, 123(1):42–51, 2010.

[79] Ulrich Lauther. An Extremely Fast, Exact Algorithm for Finding Shortest Paths in Static Networks with Geographical Background. In *Geoinformation und Mobilität - von der Forschung zur praktischen Anwendung*, volume 22, pages 219–230. IfGI prints, 2004.

[80] Tzong-Ru Lee and Ji-Hwa Ueng. A study of vehicle routing problems with load-balancing. *International Journal of Physical Distribution & Logistics Management*, 29(10):646–657, 1999.

[81] Christian Liebchen, Marco Lübecke, Rolf Möhring, and Sebastian Stiller. *The concept of recoverable robustness, linear programming recovery, and railway applications*, chapter 1, pages 1–27. LNCS 5868. Springer, 2009.

[82] Richard J. Lipton and Robert E. Tarjan. A separator theorem for planar graphs. *SIAM J. on Applied Mathematics*, 36(2):177–189, 1979.

[83] Ronald P. Loui. Optimal paths in graphs with stochastic or multidimensional weights. *Commun. ACM*, 26(9):670–676, September 1983.

[84] Cambridge Vehicle Information Technology Ltd. Choice routing. `http://www.camvit.com`, 2005.

[85] D. Luxen and D. Schieferdecker. Candidate sets for alternative routes in road networks. In *In proc of 11th International Symposium on Experimental Algorithms (SEA '12)*, number 7276 in Lecture Notes in Computer Science (LNCS), 2012.

[86] Ernesto Queiros Martins. On a Multicriteria Shortest Path Problem. *European Journal of Operational Research*, 26(3):236–245, 1984.

[87] Gary L. Miller. Finding small simple cycle separators for 2−connected planar graphs. *J. Comp. Sys. Sci.*, 32:265–279, 1986.

[88] Rolf H. Möhring, Heiko Schilling, Birk Schütz, Dorothea Wagner, and Thomas Willhalm. Partitioning Graphs to Speedup Dijkstra's Algorithm. *ACM Journal of Experimental Algorithmics*, 11(2.8):1–29, 2006.

[89] Roberto Montemanni and Luca M. Gambardella. Robust shortest path problems with uncertain costs. Technical Report Technical Report No. IDSIA-03-08, Istituto Dalle Molle di studi sullintelligenza artificiale (IDSIA / USI-SUPSI), Galleria 2, 6928 Manno, Switzerland, 2008.

[90] Shay Mozes and Christian Sommer. Exact distance oracles for planar graphs. In *Proc. of 23rd ACM-SIAM Symp. on Discr. Alg. (SODA '12)*, 2012.

[91] Matthias Müller–Hannemann and Karsten Weihe. Pareto Shortest Paths is Often Feasible in Practice. In *Proceedings of the 5th International Workshop on Algorithm Engineering (WAE'01)*, volume 2141 of *Lecture Notes in Computer Science*, pages 185–197. Springer, 2001.

[92] Ketan Mulmuley and Pradyut Shah. A lower bound for the shortest path problem. *J. Comp. Sys. Sci.*, 63:253–267, 2001.

[93] Ishwar Murthy and Sumit Sarkar. Stochastic shortest path problems with piecewise-linear concave utility functions. *Management Science*, 44(11):pp. S125–S136, 1998.

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

[95] Sabine Neubauer. Planung energieeffizienter Routen in Stra?ennetzwerken. Master's thesis, Universität Karlsruhe (TH), Fakultät für Informatik, March 2010.

[96] Evdokia Nikolova. Optimal route planning under uncertainty. In *ICAPS*, pages 131–141. AAAI, 2006.

[97] Evdokia Nikolova. High-performance heuristics for optimization in stochastic traffic engineering problems. In *LSSC*, pages 352–360. Springer, 2009.

[98] Evdokia Nikolova. *Strategic Algorithms, MIT*. PhD thesis, 2009.

[99] Evdokia Nikolova, Jonathan A. Kelner, Matthew Brand, and Michael Mitzenmacher. Stochastic shortest paths via quasi-convex maximization. In *Proceedings of European Symposium of Algorithms*, pages 552–563. Springer, 2006.

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

[101] Christos H. Papadimitriou and Mihalis Yannakakis. Shortest paths without a map. *Theor. Comput. Sci.*, 84:127–150, July 1991.

[102] 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*. Springer, 2011.

[103] 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.

[104] Hansen Pierre. Bicriterion path problems. In *Lecture Notes in Economics and Mathematical Systems*, volume 177, pages 109–127. Springer-Verlag, 1979.

[105] Ira Pohl. Heuristic Search Viewed as Path Finding in a Graph . *Artificial Intelligence*, 1(3):193–204, 1970.

[106] George H. Polychronopoulos and John N. Tsitsiklis. Stochastic shortest path problems with recourse. *Networks*, 27:133–143, 1996.

[107] 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.

[108] A.E. Rizzoli, R. Montemanni, E. Lucibello, and L.M. Gambardella. Ant colony optimization for real-world vehicle routing problems - from theory to applications. *Swarm Intell.*, pages 135–151, 2007.

[109] F. Rothlauf. Application and adaptation of heuristic optimization methods. Springer, 2009.

[110] P. Sanders and C. Schulz. Engineering multilevel graph partitioning algorithms. In *Proc. of 19th Eur. Symp. on Alg. (ESA '11)*, number 6942 in Lecture Notes in Computer Science (LNCS). Springer, 2011.

[111] Peter Sanders and Dominik Schultes. Highway Hierarchies Hasten Exact Shortest Path Queries. In *Proceedings of the 13th Annual European Symposium on Algorithms (ESA'05)*, volume 3669 of *Lecture Notes in Computer Science*, pages 568–579. Springer, 2005.

[112] Peter Sanders and Dominik Schultes. Engineering Highway Hierarchies. In *Proceedings of the 14th Annual European Symposium on Algorithms (ESA'06)*, volume 4168 of *Lecture Notes in Computer Science*, pages 804–816. Springer, 2006.

[113] Abdelkader Sbihi and William Eglese. The relationship between vehicle routing and scheduling and green logistics - a literature study, 2007.

[114] Frank Schulz, Dorothea Wagner, and Karsten Weihe. Dijkstra's Algorithm On-Line: An Empirical Case Study from Public Railroad Transport. In *Proceedings of the 3rd International Workshop on Algorithm Engineering (WAE'99)*, volume 1668 of *Lecture Notes in Computer Science*, pages 110–123. Springer, 1999.

[115] Frank Schulz, Dorothea Wagner, and Karsten Weihe. Dijkstra's Algorithm On-Line: An Empirical Case Study from Public Railroad Transport. *ACM Journal of Experimental Algorithmics*, 5(12):1–23, 2000.

[116] Frank Schulz, Dorothea Wagner, and Christos Zaroliagis. Using Multi-Level Graphs for Timetable Information in Railway Systems. In *Proceedings of the 4th Workshop on Algorithm Engineering and Experiments (ALENEX'02)*, volume 2409 of *Lecture Notes in Computer Science*, pages 43–59. Springer, 2002.

[117] 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.

[118] Melvyn Sim. *Robust optimization, MIT*. PhD thesis, 2004.

[119] Winson S.H. Siu, Chi-Kong Chan, and Henry C.B. Chan. Green Cargo Routing Using Genetic Algorithms. In *Proc. Int'l Multiconference of Engineers and Computer Scientists - IMECS 2012*, pages 170–175, 2012.

[120] 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.

[121] Alexander Strobel. Drive my spar. NAVIconnect 3/2009, WEKA Media Publishing GmbH, 2009.

[122] Dirk Theune. *Robuste und effiziente Methoden zur Lösung von Wegproblemen*. PhD thesis, Universität Paderborn, 1995.

[123] Mikkel Thorup. Compact oracles for reachability and approximate distances in planar digraphs. *J. of ACM*, 51(6):993–1024, 2004. Preliminary announcement in FOCS 2001.

[124] Mikkel Thorup and Uri Zwick. Approximate distance oracles. *J. of ACM*, 52:124, 2005.

[125] Paolo Toth and Daniele Vigo. *The Vehicle Routing Problem*. SIAM, 2002.

[126] Dorothea Wagner, Thomas Willhalm, and Christos Zaroliagis. Geometric Containers for Efficient Shortest-Path Computation. *ACM Journal of Experimental Algorithmics*, 10(1.3):1–30, 2005.

[127] Hande Yaman, Oya E. Karasan, and Mustafa Ç. Pinar. The robust shortest path problem with interval data. Unpublished Manuscript, 2001.

[128] Gang Yu and Jian Yang. On the robust shortest path problem. *Computers & Operations Research*, 25(6):457 – 468, 1998.

[129] Paweł Zieliński. The computational complexity of the relative robust shortest path problem with interval data. *European Journal of Operational Research*, 158(3):570 – 576, 2004.

[130] A.K. Ziliaskopoulos and H.S. Mahmassani. A time-dependent shortest path algorithm for real-time intelligent vehicle/highway systems applications. *Trans. Res. Rec.*, 1408:94–100, 1993.

[131] Uri Zwick. All-pairs shortest paths using bridging sets and rectangular matrix multiplication. *J. of ACM*, 49:289–317, 2002.