Query Processing in Complex Modern Traffic Networks

Gregor Jossé

München 2016

Query Processing in Complex Modern Traffic Networks

Gregor Jossé

Dissertation
an der Fakultät für Mathematik, Informatik und Statistik
der Ludwig–Maximilians–Universität
München

vorgelegt von
Gregor Jossé
aus München

München, den 04.08.2016

Erstgutachter: Prof. Dr. Matthias Schubert
Zweitgutachter: Prof. Dr. Kyriakos Mouratidis
Datum der mündlichen Prüfung: 19.10.2016

Eidesstattliche Versicherung

Hiermit erkläre ich, Gregor Jossé, an Eides statt, dass die vorliegende Dissertation ohne unerlaubte Hilfe gemäß Promotionsordnung vom 12.07.2011, § 8, Abs. 2 Pkt. 5, angefertigt worden ist.

München, 04.08.2016

Im Gedenken an meinen Vater.

Contents

Abstract

The transport sector generates about one quarter of all greenhouse gas emissions worldwide. In the European Union (EU), passenger cars and light-duty trucks make up for over half of these traffic-related emissions[1]. It is evident that everyday traffic is a serious environmental threat. At the same time, transport is a key factor for the ambitious EU climate goals; among them, for instance, the reduction of greenhouse gas emissions by 85 to 90 percent in the next 35 years[2]. This thesis investigates complex traffic networks and their requirements from a computer science perspective. Modeling of and query processing in modern traffic networks are pivotal topics. Challenging theoretical problems are examined from different perspectives, novel algorithmic solutions are provided. Practical problems are investigated and solved, for instance, employing qualitative crowdsourced information and sensor data of various sources.

Modern traffic networks are often modeled as graphs, i.e., defined by sets of nodes and edges. In conventional graphs, the edges are assigned numerical weights, for instance, reflecting cost criteria like distance or travel time. In multicriteria networks, the edges reflect multiple, possibly dynamically changing cost criteria. While these networks allow for diverse queries and meaningful insight, query processing usually is significantly more complex. Novel means for computation are required to keep query processing efficient. The crucial task of computing optimal paths is particularly expensive under multiple criteria. The most established set of optimal paths in multicriteria networks is referred to as path skyline (or set of pareto-optimal paths). Until now, computing the path skyline either required extensive precomputation or networks of minor size or complexity. Neither of these demands can be made on modern traffic networks. This thesis presents a novel method which makes on-the-fly computation of path skylines possible, even in dynamic networks with three or more cost criteria. Another problem examined is the exponentially growth of path skylines. The number of elements in a path skyline is potentially exponential in the number of cost criteria and the number of edges between start and target. This often produces less meaningful results, sometimes hindering usability. These drawbacks emphasize the importance of the linear path skyline which is investigated in this thesis. The linear path skyline is based on a different notion of optimality. By the notion of optimality, the linear path skyline is a subset of the conventional path skyline but in general contains less and more diverse elements. Thus, the linear path skyline facilitates

[1] www.eea.europa.eu//publications/term-report-2015

[2] ec.europa.eu/clima/policies/strategies/2050/index_en.htm

interpretation while in general reducing computational effort. This topic is first studied in networks with two cost criteria and subsequently extended to more cost criteria.

These cost criteria are not limited to purely quantitative measures like distance and travel time. This thesis examines the integration of qualitative information into abstractly modeled road networks. It is proposed to mine crowdsourced data for qualitative information and use this information to enrich road network graphs. These enriched networks may in turn be used to produce routing suggestions which reflect an opinion of the crowd. From data processing to knowledge extracting, network enrichment and route computation, the possibilities and challenges of crowdsourced data as a source for information are surveyed. Additionally, this thesis substantiates the practicability of network enrichment in real-world experiments. The description of a demonstration framework which applies some of the presented methods to the use case of tourist route recommendation serves as an example. The methods may also be applied to a novel graph-based routing problem proposed in this thesis. The problem extends the family of Orienteering Problems which find frequent application in tourist routing and other tasks. An approximate solution to this NP-hard problem is presented and evaluated on a large scale, real-world, time-dependent road network.

Another central aspect of modern traffic networks is the integration of sensor data, often referred to as telematics. Nowadays, manifold sensors provide a plethora of data. Using this data to optimize traffic is and will continue to be a challenging task for research and industry. Some of the applications which qualify for the integration of modern telematics are surveyed in this thesis. For instance, the abstract problem of consumable and reoccurring resources in road networks is studied. An application of this problem is the search for a vacant parking space. Taking statistical and real-time sensor information into account, a stochastic routing algorithm which maximizes the probability of finding a vacant space is proposed. Furthermore, the thesis presents means for the extraction of driving preferences, helping to better understand user behavior in traffic. The theoretical concepts partially find application in a demonstration framework described in this thesis. This framework provides features which were developed for a real-world pilot project on the topics of electric and shared mobility. Actual sensor car data collected in the project, gives insight to the challenges of managing a fleet of electric vehicles.

Zusammenfassung

Verkehrsmittel erzeugen rund ein Viertel aller Treibhausgas-Emissionen weltweit. Für über die Hälfte der verkehrsbedingten Emissionen in der Europischen Union (EU) zeichnen PKW und Kleinlaster verantwortlich[3]. Die Tragweite ökologischer Konsequenzen durch alltäglichen Verkehr ist enorm. Zugleich ist ein Umdenken im Bezug auf Verkehr entscheidend, um die ehrgeizigen klimapolitischen Ziele der EU zu erfüllen. Dazu gehört unter anderem, Treibhausgas-Emissionen bis 2050 um 85 bis 90 Prozent zu verringern[4]. Die vorliegende Arbeit widmet sich den komplexen Anforderungen an Verkehr und Verkehrsnetzwerke aus der Sicht der Informatik. Dabei spielen sowohl die Modellierung von als auch die Anfragebearbeitung in modernen Verkehrsnetzwerken eine entscheidende Rolle. Theoretische Fragestellungen werden aus unterschiedlichen Persepektiven beleuchtet, neue Algorithmen werden vorgestellt. Ebenso werden praktische Fragestellungen untersucht und gelöst, etwa durch die Einbindung nutzergenerierten Inhalts oder die Verwendung von Sensordaten aus unterschiedlichen Quellen.

Moderne Verkehrsnetzwerke werden häufig als Graphen modelliert, d.h., durch Knoten und Kanten dargestellt. Man unterscheidet zwischen konventionellen Graphen und sogenannten Multiattributs-Graphen. Während die Kanten konventioneller Graphen numerische Gewichte tragen, die statische Kostenkriterien wie Distanz oder Reisezeit modellieren, beschreiben die Kantengewichte in Multiattributs-Graphen mehrere, möglicherweise dynamisch veränderliche Kostenkriterien. Das erlaubt einerseits vielseitige Anfragen und aussagekräftige Erkenntnisse, macht die Anfragebearbeitung jedoch ungleich komplexer und verlangt deshalb nach neuen Berechnungsmethoden. Eine besonders aufwendige Anfrage ist die Berechnung optimaler Pfade, zugleich eine der zentralsten Fragestellungen. Die gängigste Menge optimaler Pfade wird als Pfad-Skyline (auch: Menge der pareto-optimalen Pfade) bezeichnet. Die effiziente Berechnung der Pfad-Skyline setzte bisher überschaubare Netzwerke oder beträchtliche Vorberechnungen voraus. Keine der beiden Bedingung kann in modernen Verkehrsnetzwerken erfüllt werden. Diese Arbeit stellt deshalb eine Methode vor, die die Berechnung der Pfad-Skyline erheblich beschleunigt, selbst in dynamischen Netzwerken mit drei oder mehr Kostenkriterien. Außerdem wird das Problem des exponentiellen Wachstums der Pfad-Skyline betrachtet. Die Anzahl der Elemente der Pfad-Skyline wächst im schlechtesten Fall exponentiell in der Anzahl der Kostenkriterien sowie in der Entfernung zwischen Start und Ziel. Dies kann zu unübersichtlichen und wenig aus-

[3] www.eea.europa.eu//publications/term-report-2015

[4] ec.europa.eu/clima/policies/strategies/2050/index_en.htm

sagekräftigen Resultatmengen führen. Diese Nachteile unterstreichen die Bedeutung der linearen Pfad-Skyline, die auch im Rahmen diese Arbeit untersucht wird. Die lineare Pfad-Skyline folgt einer anderen Definition von Optimalität. Stets ist die lineare Pfad-Skyline eine Teilmenge der konventionellen Pfad-Skyline, meist enthält sie deutlich weniger, unterschiedlichere Resultate. Dadurch lässt sich die lineare Pfad-Skyline im Allgemeinen schneller berechnen und erleichtert die Interpretation der Resultate. Die Berechnung der linearen Pfad-Skyline wird erst für Netzwerke mit zwei Kostenkriterien, anschließend für Netzwerke mit beliebig vielen Kostenkriterien untersucht.

Kostenkriterien sind nicht notwendigerweise auf rein quantitative Maße wie Distanz oder Reisezeit beschränkt. Diese Arbeit widmet sich auch der Integration qualitativer Informationen, mit dem Ziel, intuitivere und greifbarere Routingergebnisse zu erzeugen. Dazu wird die Möglichkeit untersucht, abstrakte Straßennetzwerke mit qualitativen Informationen anzureichern, wobei die Informationen aus nutzergenerierten Daten geschöpft werden. Solche *enriched networks* ermöglichen die Berechnung von Pfaden, die in gewisser Weise das Wissen der Nutzer reflektieren. Von der Datenverarbeitung, über die Extraktion von Wissen, bis hin zum Network-Enrichment und der Pfadberechnung, gibt diese Arbeit einen Überblick zum Thema. Weiterhin wird die Praktikabilität dieses Vorgehens mit Experimenten auf Realdaten untermauert. Die Beschreibung eines Demonstrationstools für den Anwendungsfall der Navigation von Touristen dient als anschauliches Beispiel. Die vorgestellten Methoden sind darüber hinaus auch anwendbar auf ein neues, graphentheoretisches Routingproblem, das in dieser Arbeit vorgestellt wird. Es handelt sich dabei um eine zeitabängige Erweiterung der Familie der Orienteering Probleme, die häufig Anwendung finden, etwa auch im der Bereich der Touristennavigation. Das vorgestellte Problem ist NP-schwer lässt sich jedoch dank eines hier vorgestellten Algorithmus effizient approximieren. Die Evaluation untermauert die Effizienz des vorgestellten Lösungsansatzes und ist zugleich die erste Auswertung eines zeitabhängigen Orienteering Problems auf einem großformatigen Netzwerk.

Ein weiterer zentraler Aspekt moderner Verkehrsnetzwerke ist die Integration von Sensordaten, oft unter dem Begriff Telematik zusammengefasst. Heutzutage generiert eine Vielzahl von Sensoren Unmengen an Daten. Diese Daten zur Verkehrsoptimierung einzusetzen ist und bleibt eine wichtige Aufgabe für Wissenschaft und Industrie. Einige der Anwendungen, die sich für den Einsatz von Telematik anbieten, werden in dieser Arbeit untersucht. So wird etwa das abstrakte Problem konsumierbarer und wiederkehrender Ressourcen im Straßennetzwerk untersucht. Ein alltägliches Beispiel für dieses Problem ist die Parkplatzsuche. Der vorgeschlagene Algorithmus, der die Wahrscheinlichkeit maximiert, einen freien Parkplatz zu finden, baut auf die Verwendung statistischer sowie aktueller Sensordaten. Weiterhin werden Methoden zur Ableitung von Fahrerpräferenzen entwickelt. Die theoretischen Fundamente finden zum Teil in einem hier beschriebenen Demonstrationstool Anwendung. Das Tool veranschaulicht Features, die für ein Pilotprojekt zu den Themen Elektromobilität und Fahrzeugflotten entwickelt wurden. Im Rahmen eines Pilotversuchs wurden Sensordaten von Elektrofahrzeugen erhoben, die Einblick in die Herausforderungen beim Management von Elektrofahrzeugflotten geben.

Part I

Preliminaries

Chapter 1

Introduction

Mobility is the backbone of modern society. Economically and ecologically it is a blemish. Significant improvements in fuel efficiency over the past 25 years have been outweighed by increasing passenger and freight transport demands. While the total energy consumption in the European Union has only increased marginally from 1990 to 2010, the energy consumption in the transport sector has risen by almost 30 percent [35]. In order to meet its goal to reduce greenhouse gas emissions by 80 to 95 percent by 2050 (compared to 1990) [21], the European Union needs to make transportation more ecological. In view of the UN climate and other resolutions, this holds for governments worldwide. Greenhouse gas emissions are not the only negative effect, merely the effect which is measured most frequently and receives the most attention. For instance, in 2014 alone, US residents wasted 6.9 billion hours and 11.7 billion liters of fuel in traffic due to congestion [145]. The cost of this extra time and fuel is estimated at EUR 145 billion – almost four times the estimate of 1982. In spite of all this, efficiency is one of the highest valued goods of our time, and convenience is an expected normality for many people.

If meeting these expectations and solving these problems is possible, is unclear. At least, there are ideas and hints at solutions. For example, a study by the geo-data enterprise NAVTEQ has shown that drivers using navigation devices increase fuel efficiency by 12 percent [96]. Another example, the McKinsey institute estimates around EUR 550 billion in annual consumer surplus from using personal location data. The biggest share coming from saving fuel and improving efficiency due to navigation systems relying on telematics [96].

The great white hope is new technology, new data. The abundance of sensor data collected in traffic, summarized under the term (vehicle) telematics, is expected to optimize traffic, to reduce congestion and to improve logistic efficiency while at the same time increasing usability and convenience for the drivers. New technology is expected to minimize ecological consequences. In a time where combustion engines work as efficiently as never before, electric vehicles are the most promising technological advance; free of emissions altogether but not yet free of doubts concerning their practicability, their economy and their disposal.

In this thesis, some of the theoretical and practical problems of modern traffic are faced.

This work is divided into four parts, this introductory part and three parts presenting research the author of this thesis has significantly contributed to. All research has been revised, restructured and amended for this thesis. The first research part lays theoretical groundwork, it examines traffic networks which allow modeling complex routing decisions. Instead of generating optimal routes with respect to one criterion (e.g., distance or time), these networks enable taking multiple criteria into account. The second research part of this thesis investigates how alternative criteria may be attained and applied. Instead of relying on "hard" metrics, this part surveys the possibility of using crowdsourced data to generate networks enriched with the qualitative knowledge of the crowd. This knowledge may be applied to a novel graph-based routing problem which is proposed, substantiated and solved. In the third and final research part of this thesis, more practical advances are made. Based on sensor data, new queries are introduced and solved which deal with ubiquitous traffic problems, concerning both usability and efficiency.

This introductory part provides principles fundamental to all the research parts. Chapter 2 introduces basic definitions, several established algorithms which are the foundation of this research field and the rest of the thesis. Subsequently, in Chapter 3, the research papers published in the context of this thesis are listed and the contribution of the author of this thesis is emphasized.

Chapter 2

Fundamentals

In this chapter, the foundation for the topics discussed in this thesis is laid. First, definitions are introduced which are widely used throughout this thesis. If not explicitly redefined, the following definitions are applied. Second, practically as well as historically relevant basic algorithms are presented, their properties, limitations and respective complexities are stated. Third, advanced algorithms and techniques are discussed. These methods constitute the state of the art for most routing problems and most have relevance and/or relation to the research presented in this thesis. Finally, the vital concept of multicriteria networks and applicable algorithmic concepts are presented.

2.1 Basic Definitions

The focus of this work is to provide solutions to problems which occur in real-world traffic. Some of the ideas presented can be generalized to other applications such as computer networks or social graphs. What all application areas have in common is that they are usually modeled as graphs. In its most general mathematical form, a graph is a two-tuple consisting of a set of nodes V and edges E. The nodes are also referred to as vertices or points, the edges as arcs or lines. A graph is commonly denoted by $G = (V, E)$. The edges represent connections between the nodes, in general they are subsets of V consisting of two nodes, i.e., edges induce a binary relation on nodes. Any node may occur arbitrarily often in the set of edges and need not occur at all.

When modeling traffic and other networks as graphs, the common approach is to model entities as nodes and connections between entities as edges. In social networks, the users constitute the nodes, and the friendship relation, for instance, may be an edge. In a computer network, the machines constitute the nodes, and an edge may represent some form of communication. In traffic networks, crossings, dead ends and forks commonly constitute the nodes, and the street segments connecting these nodes represent the edges. Alternatively, one may represent the driving lanes by nodes and the connections between segments as edges. This facilitates modeling different traffic rules such as “no left turns” or “no U-turns” but is rarely used compared to the former model. In this thesis, the former

model where street segments correspond to edges is applied. Of course, some streets (or social interactions like "following" or machine communication like "REST requests") are one-way, which is not captured by the general model. Thus, usually directed graphs are used to model networks. Instead of unordered sets, the edges correspond to two-tuples of nodes, i.e., $E \subseteq V \times V$. We call an edge $e = (u, v)$ an *outgoing edge* of u, an *incoming edge* of v, and u and v *neighboring* or *adjacent* nodes.

So far, edges can only be counted and distinguished as to what nodes they connect. Of course this is not the case when thinking of real-world networks. Two distinct streets have different properties, e.g., they differ in length. The same holds for different network requests which might differ in package size or social interactions which might differ in content or relation. In order to model networks appropriately, the concept of weighted graphs is applied. Every edge is assigned a numeric cost value (in graph theory mostly referred to as weight), i.e., there exists a function $c : E \to \mathbb{R}$.

Definition 2.1. *A graph or network is a directed and weighted graph $G = (V, E, c)$, where V is a set of nodes and $E \subseteq V \times V$ is a set of edges. $c : E \to \mathbb{R}_{\geq 0}$ is a cost (or weight) function which maps every edge onto its non-negative traversal cost.*

Note that throughout this thesis, cost functions are assumed to be non-negative – an explanation will be given shortly. There are diverse problems in graph theory. In the context of this work, we focus on routing problems, the group of graph theory problems most related to traffic. The answer to most routing problems is a path or route or way, most commonly the cost-optimal path connecting two input nodes. The terminology is not fully consistent except that a path or route or way is usually a consecutive sequence of edges. Some authors require paths to be cycle-free, i.e., every vertex is visited exactly once, others require the same for routes.

Definition 2.2. *The term route refers to any sequence of consecutive edges, the term path refers to a route where no vertex is visited twice.*

Note that a route may be denoted as sequence of edges (i.e., $(e_1, \ldots, e_n)$) or a sequence of two-tuples of nodes (i.e., $((v_1, v_2), (v_2, v_3), \ldots, (v_{N-1}, v_N))$). These notations are of course interchangeable but depending on the context, one might be more convenient than the other. The notion of the cost function extends naturally to paths.

Definition 2.3. *The cost of a path $p = (e_1, \ldots, e_n)$ is the summed cost of its edges e_i, $c(p) = \sum_{i=1}^{n} c(e_i)$.*

Classic routing problems are for example the Hamiltonian path problem or the Traveling Salesman Problem. The Hamiltonian path problem is the question whether a path exists that visits each node exactly once; it is NP-complete. For the Traveling Salesman Problem a set of nodes (cities, in the analogy) is given, and the task is to find the shortest path visiting each of these nodes exactly once. The problem most related to those in this thesis is the shortest path problem, also called point-to-point or P2P problem. Given a *start (source)* and a *target (destination)* node, the task is to find the path from start to target

which minimizes the accumulated cost. Note that the result need not be the shortest path if the cost is not distance. Hence, in this thesis the term *cost-optimal path* is preferred, sometimes using shortest path synonymously. There exist variations to the problem which are not always clearly distinguished, for instance, finding the shortest paths from one node to all other nodes or finding the shortest paths for all pairs of nodes. There exist numerous algorithms to solving the shortest path problem and its variations in different graphs with different properties. The most common methods are presented in the next section.

2.2 Shortest Path Algorithms

The most common algorithms for the shortest path problem are Bellman-Ford [6, 40], Floyd-Warshall [38, 153] and Dijkstra's algorithm [29]. Based on the fundamental algorithms are numerous variations, mostly optimizing runtime complexity by introducing additional constraints (e.g., integer weights) or advanced data structures (e.g., Fibonacci heaps). The improvements only work to a certain extent. In general, the acceleration methods presented in Section 2.3 have greater impact, although most cannot guarantee a better worst case complexity than the original algorithms. We introduce the three classic shortest path algorithms as they are relevant historically as well as in the context of this work. In the following, let n denote the number of nodes and m the number of edges.

The Bellman-Ford algorithm computes the shortest paths and their distances from a given start node to all other nodes. For every node it maintains a distance d and a predecessor variable. The distance is initialized as infinity except for the start node where it is 0, the predecessors are initialized as null. The algorithm iterates over all nodes, and in each iteration it iterates over all edges; the time complexity is $\mathcal{O}(nm)$. When processing an edge (u, v) with cost $c(u, v)$, it is checked whether $d(v) > d(u) + c(u, v)$, i.e., whether the path via u is cheaper than the current path. If so, $d(v)$ is reduced accordingly (*relaxed*) and u is set as predecessor. The labels at each node are refined iteratively, thus, Bellman-Ford is a label-correcting algorithm. Proceeding in this manner, the algorithm will find the shortest paths from the start node to all other nodes.

The Floyd-Warshall algorithm computes the shortest path distances between all pairs of nodes. In order for it to compute the actual paths, slight modifications are needed. It initializes a $n \times n$ distance matrix D where $D_{u,u} = 0$ and for neighboring nodes u and v, $D_{u,v} = c(u, v)$. Then, for all pairs u, v and for all nodes w, it is checked whether $D_{u,v} > D_{u,w} + D_{w,v}$, i.e., if the path via w is cheaper. If so, $D_{u,v}$ is reduced accordingly (relaxed). Obviously, the runtime complexity is $\mathcal{O}(n^3)$. If every node in the graph is reachable from some node, the number of edges is at least $n - 1$, in a real-world road network, the number of edges is usually between $2n$ and $3n$. Thus, in general Floyd-Warshall is faster than Bellman-Ford.

Dijkstra's algorithm in its original form computes the shortest paths and their distances from a given start node to all other nodes. However, it can easily be modified to compute the shortest path between a start and a target node. Every node that is flagged as processed has been reached via the shortest path from the start node. This property is sometimes

referred to as Dijkstra property. Due to the Dijkstra property, it is possible to terminate the search once the target node is reached, the shortest path and its distance are final. Although the original version of Dijkstra's algorithm [29] does not use a priority queue, it is most commonly assumed to use one (ideally, implemented as a Fibonacci heap [41]). In the following, we always assume Dijkstra's algorithm to use some priority queue. The queue maintains all unprocessed nodes w.r.t. their distance to the start node, initially, the start node is added to the queue (with distance 0). Also, every node is initialized with a distance d (infinity) and a predecessor variable (null) just as in the Bellman-Ford algorithm. In every iteration of the algorithm, the top element u is dequeued and all its neighboring nodes are examined (not processed). When a node v is examined, it is checked whether $d(v) > d(u) + c(u, v)$. If so, $d(v)$ is relaxed, u is set as predecessor. If v was previously not in the queue, it is enqueued, otherwise its priority is set to the decreased value of $d(v)$. After having examined all of its neighbors, u is flagged as processed. Note that because of the priority queue, nodes are dequeued with increasing distance to the start node. In particular, every node is dequeued and in turn processed only once and must have been reached via the shortest path, otherwise it would have been processed earlier. Thus, the Dijkstra property indeed holds. Consequently, Dijkstra's algorithm is a label-setting algorithm. Depending on the implementation of the queue, Dijktra's algorithm runs in $\mathcal{O}(n^2)$ (list), $\mathcal{O}((m + n) \log n)$ (binary search tree) or $\mathcal{O}(m + n \log n)$ (Fibonacci heap) to determine the shortest paths from one to all other nodes.

The difference between the three algorithms are the possible inputs. Dijkstra's algorithm can process directed and undirected graphs with non-negative costs. Floyd-Warshall and Bellman-Ford can process directed and undirected graphs with any costs. However, if there exist cycles with negative cost (i.e., $c((v_1, v_2), (v_2, v_3), \ldots, (v_k, v_1)) < 0$), there may be start and target nodes for which no shortest path exists, as each rotation in the cycle reduces cost. In this case, Floyd-Warshall and Bellman-Ford may not return a result but may be used for cycle detection. For almost any application where the graph weights represent costs, it is the norm to assume costs to be non-negative. If costs correspond to distance or travel time, for instance, non-negativity is obvious, in any other scenario it is plausible. For some cases, there may exist negative costs, for instance, when considering the state of charge in electric vehicles. However, there may not exist negative cycles. It is impossible to travel a non-trivial distance and end up at a previous node with more energy. This would imply a perpetual motion machine. Furthermore, Cherkassky et al. have shown that the shortest path problem in a graph with no negative cycles is equivalent to the shortest path problem in a graph with non-negative edge costs [20]. In accordance with this result and most traffic-related research, we assume costs to be non-negative. When assuming non-negative costs, shortest paths cannot contain cycles. Dijkstra's algorithm is the standard shortest path algorithm (if no acceleration method is needed). This thesis abides by this convention.

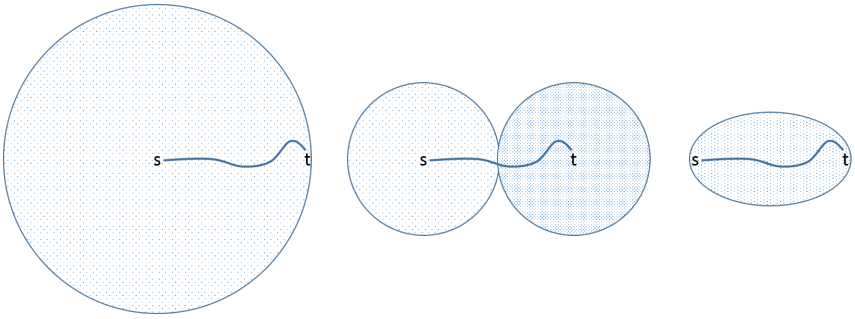


Figure 2.1: Schematic illustration of search spaces for shortest path searches using Dijkstra, bidirectional Dijkstra and $A^\star$ search (from left to right).

2.3 Acceleration Methods

Even in the fastest implementation, Dijkstra's algorithm may become infeasible at a certain graph size when requiring fast response times. On a city scale it is hardly worth the effort to implement acceleration methods. On a state scale, Dijkstra's algorithm might be sufficient, on a country scale, however, it is usually insufficient. Nodes are dequeued with increasing distance to the start node, hence, the search space (i.e., the visited nodes) is a ball around the start node with growing radius, as illustrated in Figure 13.2. If assuming uniformly distributed nodes, the search space is quadratic in the distance between start and target. Notwithstanding, the search space is limited by the overall graph size. The shortest path for the same start and target pair will be computed faster in a smaller graph (where the search is rather global) than in a large graph (where the search is rather local). This is because the Dijkstra search visits more irrelevant nodes in a large graph.

One may reduce the search space of Dijkstra's algorithm by employing bidirectional search [22, 106, 54]. Instead of maintaining one distance label at each node, a forward and a backward label are maintained. Two Dijkstra searches are conducted. The so-called forward search is as before, from the start node following all outgoing links. The other is called the backward search, it starts at the target node and follows all incoming links. Figuratively, the bidirectional Dijkstra search expands two Dijkstra balls, one around the start and one around the target node, as illustrated in Figure 13.2. When the two expansions meet, the search terminates. Assuming uniformly distributed nodes, the search space is quadratic in half the distance between start and target.

The most established and effective acceleration method for shortest path searches is guiding the search towards the target, often called goal-direction. Dijsktra's algorithm expands circularly, i.e., it scans all nodes with a distance smaller than the distance from start to target. The idea of goal-direction is to prioritize nodes which are closer to the target, this can limit the search space (illustrated in Figure 13.2) and produce results faster. There are several methods of goal-direction.

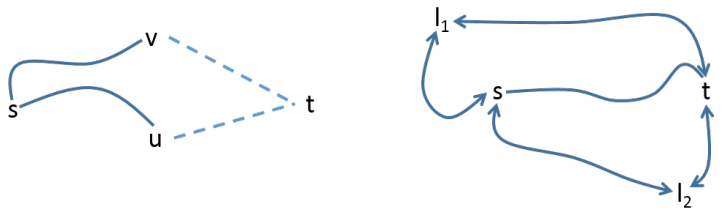


Figure 2.2: Illustration of lower bound approximations using Euclidean distance (left) and landmarks (right).

2.3.1 $A^\star$

The first and prevalent goal-direction method is the $A^\star$ algorithm [62]. It essentially runs a Dijkstra search with modified priority values in the queue. Instead of processing nodes with increasing distance from the start node, $A^\star$ also takes the distance to the target node into account. The priority value f of a node n is set to $f(n) = g(n) + h(n)$, where $g(n)$ is the distance from the start node to n and $h(n)$ is a heuristic estimating the remaining distance, i.e., from n to the target. If the heuristic never overestimates the remaining distance, it is called a *lower bound*. In this case, the algorithm indeed returns the shortest path. There are two extreme cases for $h(n)$. First, if $h(n) = 0$ for all n, the heuristic yields no benefit, and the algorithm is equivalent to Dijkstra's algorithm. Second, if $h(n)$ is the exact distance from n to the target, h is called an *optimal lower bound*, and the search space would be limited to the shortest path if unique. Note that there exist bidirectional variants of $A^\star$ but to ensure correctness more precautions have to be taken [53].

For the original shortest path problem there exists a convenient heuristic. The Euclidean distance never overestimates the exact distance in a network graph, as illustrated in Figure 2.2. However, if the cost function is not distance, there is rarely an effective equivalent. Consider travel time, for instance. Of course it is possible to lower bound the exact travel time for a particular street by the distance of the street divided by the maximum possible speed. The higher the maximum possible speed, the looser the bound. In general the bound is moderately effective. Consider fuel consumption or toll fees, in these cases no non-trivial heuristics exist at all.

2.3.2 ALT

Since no straightforward heuristic exists for most cost criteria, a different approach has to be taken. Using precomputed information, it is possible to derive lower (as well as upper [80]) bounds which are independent of the type of criterion. The core concept is summarized by the acronym ALT, $A^\star$, landmarks and triangle inequality [53]. During preprocessing, a fixed number of landmarks is chosen, and for each node the shortest path cost to and from each of the landmarks is computed, as illustrated in Figure 2.2. Lower bounds may be derived from the precomputed distances using the triangle inequality. Let $c(n, m)$ denote the cost of the shortest path from n to m, let l_i denote the landmarks,

let s and t denote start and target nodes, respectively. The cost from s to t may be approximated by

$$c(s,t) \geq \max\{c(s,l_i) - c(t,l_i), c(l_i,t) - c(l_i,s)\}$$

By taking the maximum over all these values for varying l_i, correctness holds and the bound is tightened. Note that this method is applicable to arbitrary cost criteria, defining $c(m,n)$ as the cost of the cost-optimal path from m to n w.r.t. the specific criterion. Even when used for the original cost criterion distance, ALT outperforms $A^\star$ in terms of efficiency (as it provides tighter bounds) and runtime (as at query time the precomputed distances can be retrieved instead of computing a Euclidean distance for each node in the priority queue) [53]. Of course, memory usage is $\mathcal{O}(nk)$ where n is the number of nodes and k is the number of landmarks. However, significant improvement can be achieved with a moderate number of landmarks (< 20). The increase in performance and the applicability to arbitrary cost criteria make ALT a strong advancement of Dijkstra's algorithm and $A^\star$.

Another well-established acceleration method to the shortest (or cost-optimal) path problem is Arc Flags [77]. During preprocessing, the graph is partitioned, e.g., split up into a grid. The idea behind Arc Flags is that in order to reach a node within a certain grid cell, at least one of the edges (arcs) coming into the particular cell has to be passed. These edges are called boundary arcs. Each boundary arc is assigned a bit vector of length number of grid cells. The i-th bit is set if the arc lies on some cost-optimal path to some node in cell i. When computing a path from s to t, boundary arcs can be pruned if they do not have the bit which belongs to the cell of t. There are several variations to this approach, e.g., multilevel arc flags [98]. In general, Arc Flags yields a great leap in response times which, however, comes at the cost of extensive precomputation. Another related and less popular acceleration method uses geometric properties of the graph, sometimes called Geometric Containers [123, 152]. The basic idea is to precompute cost-optimal paths. For each edge (u,v) all nodes are stored which are reachable through cost-optimal paths starting at u whose first edge is (u,v). In order to make the precomputation of all possible cost-optimal paths reasonably feasible, geometric pruning rules are applied.

Besides ALT and Arc Flags, Hierarchical approaches are the most common and effective acceleration method. The idea behind these approaches stems from the observation that road networks have an inherent hierarchy. For example, in Germany, the *Autobahn* allows for fast travel for great distances, *Landstraße* connects small cities and villages in rural areas, from the cities to the Autobahn, there are feeder streets, and in the cities there exists another hierarchy of streets. When traveling (longer distances), drivers usually ascend through the hierarchy, choosing the fastest path to an "entry point" of the next level of the hierarchy. They usually proceed this way until reaching the highest level and travel there for the longest possible time (as its normally most effective), then descend through the hierarchies when reaching the vicinity of their destination. It has been conjectured that commercial navigation service rely on such routing but in a heuristic manner which may lead to suboptimal paths [123]. In order to optimize efficiency and/or result quality, several approximate and exact heuristic approaches have been published in the research community, most prominently Contraction Hierarchies [50], Highway Hierarchies [120],

Highway Node Routing [122] and Reach [54, 55].

In conclusion, there exists a wide array of acceleration methods for shortest or cost-optimal path computation of which only the most established have been touched in this section. In the context of this thesis, efficiency is particularly central to Part II, which addresses the computation of cost-optimal paths in networks with more than one cost criterion, so-called multicriteria networks which are motivated and introduced in the following.

2.4 Multicriteria Networks

In the networks considered so far, cost-optimality is an intuitive notion. The path with minimum accumulated cost is preferred above all alternative paths (in the special case that it is non-unique, one path may be chosen non-deterministically). In reality, routing decisions in traffic and elsewhere are usually more complex. For instance, in order to avoid a particular toll road, a driver might be willing to accept additional travel time. But how much travel time equals how much toll fee? Or, in a friendship graph, is a friendship where only messages are exchanged as close as a friendship where only public interaction is displayed, such as leaving comments and liking posts? In order to model such decisions adequately, it usually does not suffice to rely on one cost criterion like travel time or number of words exchanged. Instead, different aspects have influence on a decision or on the outcome of a comparison. Multicriteria networks allow for incorporating multiple cost functions. Each edge holds a vector of (non-negative) cost values, one for each criterion. In traffic networks, possible cost criteria are for instance distance, travel time, energy consumption or toll fees. In social networks, possible cost criteria may be derived from call duration, mail volume or interaction frequency.

Definition 2.4. *A d-dimensional multicriteria network or graph is a directed and weighted weighted graph $G = (V, E, c_1, \ldots, c_d)$, where V is a set of nodes and $E \subseteq V \times V$ is a set of edges. The cost vector of an edge e is the vector of the values in all cost criteria (sometimes called dimensions), i.e., $(e_1, \ldots, e_d)^T = (c_1(e), \ldots, c_d(e))^T \in \mathbb{R}^d_{\geq 0}$. Consequently, any path p can also be assigned a cost vector, the component-wise sum of the cost vectors of its edges.*

Note that we usually compare paths w.r.t. their corresponding cost vectors. While for every path there exists a unique cost vector, the converse does not hold. Although in reality unlikely, two paths can have the same cost vectors. Let us stress that speaking of a given cost vector, we implicitly assume there exists a given path with which the cost vector is associated. For reasons of brevity we might omit explicitly mentioning the path which constitutes a given cost vector. Also, we take the liberty to slightly abuse notation and denote a path as well as its cost vector by the same letter, e.g., p.

When comparing paths (more precisely, their cost vectors), we always assume them to start and end at the same nodes within the underlying network. Note that different problem definitions exist. For instance, an alternative early work on paths in multicriteria networks [100] evaluates different locations within the network. The problem has practical

relevance in logistics. Given a query location (e.g., a port) and several related locations (e.g., possible future warehouse locations) it is possible to compare the eligibility of each of the locations. In this case, it makes sense to compare paths starting at the same but ending at different nodes. In this thesis, however, paths are compared which start and end at the same nodes.

When referring to multiple paths or cost vectors, we either choose different letters (e.g., p, q) or superscript their indices (e.g., p^i). Subscript indices usually refer to the different cost dimensions. For given start and target nodes $s, t \in V$, respectively, we denote the set of all paths from s to t by $\mathcal{P}(s,t)$ or $\mathcal{P}$ if s and t are clear from context.

In a multicriteria network, the optimal path w.r.t. one of the cost criteria can be determined by the same algorithms as in a single-criterion network. As implied before, optimality often does not depend on a single criterion but on a combination of criteria. Another possibility is therefore to combine the multiple criteria into a numeric value – a combined criterion – utilizing what we refer to as a combination function. Given a combination function, optimal path computation can be solved as before.

Definition 2.5. *Any function $f : \mathbb{R}^d_{\geq 0} \to \mathbb{R}_{\geq 0}$ is called a combination function.*

$f((x_1, \ldots, x_n)^T) = \sum x_i$ is an example of such a combination function. In the context of this thesis, combination functions are mostly required to be either monotone or linear.

However, no universal notion of optimality exists. The standard example (for vectors, not paths) is the decision for a hotel which is preferably both, close to the beach and cheap. However, hotels close to the beach have the tendency to be more expensive. For two hotels in equal distance to the beach the decision is simple. For instance, if hotel A has the properties $(500\text{m}, 100\$)$, and $B = (500\text{m}, 150\$)$, A yields the better trade-off. However, when comparing A to $C = (100\text{m}, 250\$)$, the decision is not clear. A is preferred if the trade-off tends towards a lower price, C is preferred if the trade-off tends towards closeness to the beach. If the user is aware of their subjective trade-off, they may be able to specify the respective combination function. In this example, finding one's personal trade-off might be relatively easy. In general, finding the right combination function is highly subjective and application-dependent. There exist infinitely many possible functions and the personal preference is typically not numerically grasped. Explicitly choosing a function for complex problems of higher dimensionality is neither convenient nor straightforward. Alternatively, one may compute a set of paths and let the user select a result according to their liking. The set of vectors or paths which is most established in the research community is the (path) skyline [80, 158] or synonymously the set of pareto-optimal vectors or paths [61, 94]. The problem of determining this set can be formulated as a multi-objective optimization problem.

Originally, the problem was formulated for vectors. According to a definition of *domination*, all non-dominated vectors in a set form the skyline [8] or the pareto-front/-set. In the above example, both A and C are part of the skyline while B is dominated by A. The concept of a vector skyline may be extended to paths in a graph. Associated with every path is a cost vector. We therefore say a path dominates another path (connecting the same start and target) if the cost vector of the former dominates the cost vector of

the latter. The crucial difference between both problems is that in the case of vectors the respective set of vectors is given. In contrast, in the case of paths, only start and target nodes are given. It would be possible to apply algorithms for vector skylines to path skylines. However, one would have to compute the entirety of paths from start to target and use these cost vectors as input. This is, of course, impractical and possibly infeasible. The number of paths between two nodes is exponential and it contains unreasonable paths, for instance, arbitrarily long detours. Thus, although the definitions of both problems coincide largely, the problems are significantly different. The following definition pertains to paths but is analogous for cost vector.

Definition 2.6. *The set of pareto-optimal paths (path skyline) uses the following notion of* domination*, sometimes referred to as* local domination*. A path p dominates a path q, denoted by $p \prec_{dom} q$, iff*

$$p_i \leq q_i \; \forall \; 1 \leq i \leq d \wedge \; \exists \; 1 \leq j \leq d : p_j < q_j$$

Note that p_i refers to the cost value of p in the i-th cost criterion.

Definition 2.7. *Let $\mathcal{P}$ be the set of all paths connecting given start and target nodes. The set of pareto-optimal paths $\mathcal{S}$ is defined as*

$$\mathcal{S} = \{p \in \mathcal{P} \mid \nexists \; q \in \mathcal{P} : q \; \prec_{dom} \; p\}$$

There exist several works on computing the set of pareto-optimal paths in networks with two [61, 115] or more criteria [80, 34, 94]. In the case of two criteria slightly different terminology and special properties exist but the approaches share a commonality. They keep a local skyline at each node, containing all non-dominated paths from the start node to that particular node. This is similar to Dijkstra's algorithm where each visited node is labeled with the distance from the start node and the respective predecessor. When computing the set of pareto-optimal paths, there usually exist numerous non-dominated paths to each node, which have to be stored in their entirety. Let s denote the start, t the target node and v an arbitrary distinct node. Any locally (from s to v) non-dominated path may extend into a globally (from s to t) non-dominated path. Hence, all non-dominated paths have to be maintained and extended, often diminishing performance. Another problem is that the number of result paths increases exponentially. These problems are tackled in Part II of this thesis.

Chapter 3

Publications and Coauthorship

The research presented in this thesis has been conceptualized, formalized and evaluated by the author of this thesis and his supervisor PD Dr. Matthias Schubert, in cooperation with students as well as researchers at the Database Systems Group at LMU Munich and external researches which were partially on visit to the Database Systems Group at LMU Munich. The published content has been revised, restructured and amended for this thesis by its author. In the following, the author's contribution to the publications is highlighted.

Part II: Optimal Paths in Multicriteria Networks. The problems presented in Part II have been researched in [130, 125, 128] as a cooperation with PD Dr. Matthias Schubert and Michael Shekelyan who at the time was a student at LMU Munich and is now a researcher at the Free University of Bozen-Bolzano, Italy. The author of this thesis was involved in conceptualizing, compiling and writing the papers which originated from this cooperation. For the context of this thesis, the content has been restructured and several proofs have been revised (e.g., Theorem 7.1, Lemma 8.2, Lemma 8.3).

Part III: Network Enrichment and Paths in Enriched Networks. Chapter 11 has been conceptualized and written for publication in this thesis and is currently undergoing peer review for the Springer journal *GeoInformatica* (upon invitation to a special issue on the best papers of SSTD 2015). Preliminary results have been published in [136, 66, 135] and were developed in cooperation with external researchers (Prof. Andreas Züfle, Prof. Dieter Pfoser), external researchers on visit to LMU Munich (Dr. Georgios Skoumas and Prof. Mario A. Nascimento) as well as researchers from LMU Munich. The author of this thesis has contributed significantly to the ideas and writing of the published papers. The as of yet unpublished extension which constitutes Chapter 11 was developed in great parts and written entirely by the author of this thesis. Furthermore, the author of this thesis wrote great parts of the demonstration papers [66, 68] and, based on the MARiO framework [58], developed and implemented the back ends for both demos. The demonstration framework in Chapter 12 was previously published in [68] and was awarded *Best Demonstration Paper* at the International SSTD 2015. The work in Chapter 13 is currently undergoing peer review. It is a cooperation with external researchers (Ying Lu, Prof. Matthias Renz, Dr.

Cyrus Shahabi and Dr. Ugur Demiryurek) as well as researchers from LMU Munich. The author contributed significantly to the solution approach, coordinated the experimental evaluation and is exclusively responsible for presentation and writing.

Part IV: Sensor Data Applications and Query Processing. The work in Part IV has been published in [4, 67, 71]. Preliminary results have been published in [70, 69]. The author of this thesis was deeply involved in the process of conceptualizing, formalizing and writing the papers. [67] have been devised, structured and written by the author of this thesis, building on valuable input from and discussions with PD Dr. Matthias Schubert. Some of the content of [4] has been revised for this thesis. The front end of the demo [71] has been designed in cooperation with a student (Ludwig Zellner), the back end has been developed and implemented by the author of this thesis.

Part II

Optimal Paths in Multicriteria Networks

Chapter 4

Introduction

In recent years, querying network data has gained significant importance in many application areas such as transportation systems, social graphs, or computer networks. One of the most central tasks in networks is computing cost-optimal paths between a given pair of nodes. In spatial networks like road or other transportation networks, computing cost-optimal paths is the core requirement of routing. In social networks, like co-authorship or friendship graphs, cost-optimal paths measure the proximity of people within the network. Depending on the given network and application, the cost of an edge has different meaning. In traffic networks, the cost might represent distance, travel time, energy consumption or toll fees. To describe the proximity of people in a social graph, call duration, mail volume or interaction frequency are aspects which may be transformed to cost criteria.

In contrast to single-criterion networks where optimality is easily defined, this notion does extend to multicriteria networks. The standard approach is to compute a set of optimal paths from which the user may choose. The most established set is the path skyline [80, 158] or synonymously the set of pareto-optimal paths (cf. Definitions 2.6 and 2.7). Computing the path skyline does not require an explicit combination function, but there are other practical limitations. The number of result paths may increase exponentially w.r.t. the number of cost criteria as well as w.r.t. the number of edges between start and target node. This has two major consequences. First, processing time and memory consumption usually degenerate in complex networks. Second, returning an abundance of possible paths might confuse the user, in particular, because result paths are often similar to each other.

In this part, both problems are tackled. First, we provide a novel method for the computation of optimal lower bounds in multicriteria networks which can be used for goal-direction. These bounds can, for instance, be used to significantly accelerate state-of-the-art path skyline algorithms. Second, we propose computational methods for the so-called linear path skyline, which is a subset of the conventional skyline. Computing the linear path skyline in general reduces the overall number of results and diversifies results, therefore facilitating interpretation and increasing usability. By employing the aforementioned lower bounds, the computation of linear path skylines becomes highly efficient. In the following, we give a more detailed introduction to both topics.

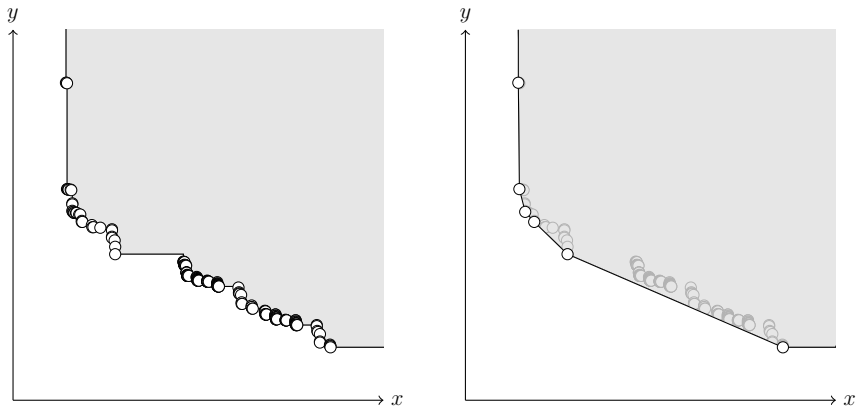
4.1 Introduction to Lower Bounds

In order to make path skyline computation in complex networks feasible, goal-direction is essential. Lower bounds underestimate the travel costs from a node to another node. The general idea is analogous to the concept of the $A^\star$ search for shortest path computation; the tighter the bounds, the better the pruning and goal-direction. The difference is that in multicriteria networks lower bounds for all criteria are required. Although there are path skyline algorithms that do not employ lower bounds (e.g., BRSC in [80]), their use is restricted to small graphs which can be visited entirely, making these algorithms infeasible in most cases. Lower bounds for all criteria are required to ensure efficiency. There exist two approaches to lower bound computation, query-independent and query-dependent bounds.

Query-independent bounds can, for instance, be derived from precomputational methods like ALT [53] or the reference node embedding [80]. In the precomputation phase a number of designated vertices, the landmarks, are picked. Subsequently, the shortest path costs between these landmarks and all other vertices in the graph are computed. At query time, lower bounds may be derived using the triangle inequality (cf. Section 2.3). This yields an approximation of the actual path costs. Query-independent bounds constitute the state of the art of single-criterion optimal path queries. However, when computing the path skyline, the pruning power of query-independent bounds often does not suffice to ensure feasible runtime.

Query-dependent bounds, on the other hand, take start and target of a query into account. In general, query-dependent bounds constitute the actual cost of paths yielding these bounds. The most established approach [148], conducts reverse Dijkstra searches from the target to all other nodes in the network. This provides exact information about the costs of the optimal path from every node to the target. Although the computational effort is large, the optimality of the bounds for the given task often leads to a significant speed-up when performing the actual path computation.

With ParetoPrep, we introduce a new method for the computation of query-dependent bounds in multicriteria networks. It provides optimal lower bounds while reducing the search space. Independent of the number of criteria, the graph only has to be traversed once. In addition, the graph traversal is only partial. We prove that ParetoPrep visits all nodes which are potentially part of any skyline path. This means, ParetoPrep is a valid and highly efficient preprocessing step for any path skyline algorithm, e.g., for the algorithms presented in [80, 158, 141]. Furthermore, we show that ParetoPrep computes the single-criterion cost-optimal path for each of the criteria in a single graph traversal. Therefore, an additional use of ParetoPrep is to compute optimal paths for a given set of criteria or combinations thereof. When introducing combined cost criteria (e.g., 60% `distance`, 30% `toll fees` and 10% `energy consumption`), ParetoPrep provides all cost-optimal paths.



(a) Conventional Skyline (b) Linear Skyline

Figure 4.1: Conventional and linear skylines on an exemplary data set in a two-dimensional cost space.

4.2 Introduction to Linear Skylines

Let us highlight the relation between optimality under monotone combination functions and domination in the path skyline, as in Definitions 2.6 and 2.7. First, a *monotone combination functions (mcf)* is a function $f : \mathbb{R}^d_{\geq 0} \to \mathbb{R}$ which fulfills: For any cost vectors $x, y \in \mathbb{R}^d_{\geq 0}$ where $a_i \leq b_i$ for all $1 \leq i \leq d$, holds that $f(a) \leq f(b)$. Recall that

$$x \prec_{\text{dom}} y \Leftrightarrow x_i \leq y_i \; \forall \; i \wedge \; \exists \; j : x_j < y_j$$

We note the following relations between mcfs and the concept of domination:

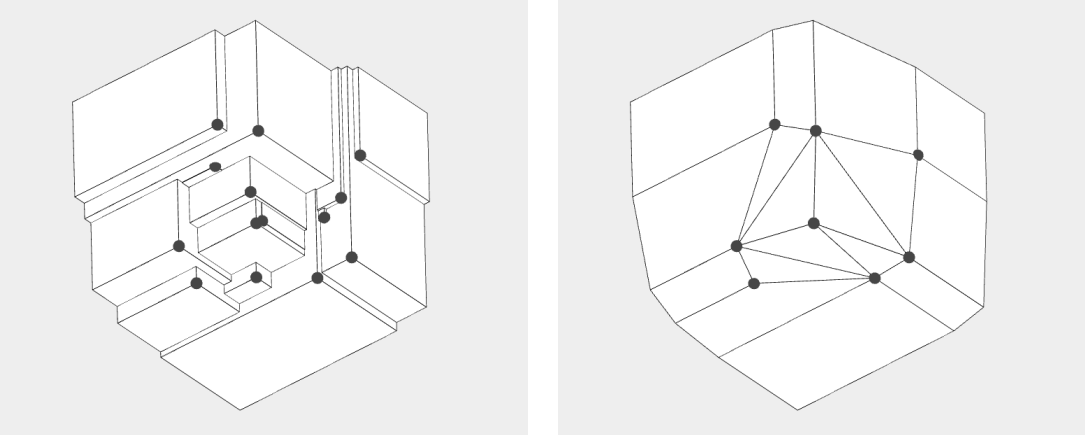
$$x \prec_{\text{dom}} y \Leftrightarrow \forall \; 0 \neq f \text{ mcf} : f(x) < f(y)$$
$$x \not\prec_{\text{dom}} y \Leftrightarrow \exists \; 0 \neq f \text{ mcf} : f(y) \leq f(x)$$

Taking these equivalences into account, the analogy between the two concepts becomes evident. The set of mcf, however, does not necessarily correspond to the intuition of a trade-off between criteria. For instance, the functions $f_p(x) := \sum x_i^p$ are monotone for $p \in \mathbb{N}$ on non-negative vectors x, but they hardly describe a meaningful trade-off. In contrast, the meaning of

$$\texttt{distance}^p + \texttt{toll fees}^p + \texttt{energy consumption}^p$$

is hardly conceivable.

Therefore, we propose to tackle the shortcomings of the conventional path skyline by altering its definition of optimality. Instead of computing the paths optimal under some monotone cost function, we propose to compute the paths optimal under the simplest subclass of monotone functions, the weighted sum. Intuitively, this corresponds to the case where a user inputs a percentage of importance for each cost criterion. We refer to



(a) Conventional Skyline (b) Linear Skyline

Figure 4.2: Conventional and linear skylines on an exemplary data set in a three-dimensional cost space viewed from the origin.

the paths optimal under the weighted sum with some positive weight vector as the *linear path skyline*. The linear path skyline is a subset of the general path skyline and usually has significantly smaller cardinality. This increases interpretability of results and facilitates result choice for the user. In addition, a reduced result set implies the opportunity to accelerate computation. The difference between the conventional skyline and the linear skyline is illustrated in Figures 4.1 and 4.2 for two-dimensional and three-dimensional cost spaces.

From a mathematical point of view, the linear path skyline corresponds to the convex hull in a particular augmented set of cost vectors which will be described in greater detail later on. A naïve approach to computing the linear path skyline is to compute the ordinary path skyline and subsequently compute the convex hull on the augmented set of cost vectors. However, this approach requires computing the conventional path skyline first, which, of course, is inefficient and precisely what we aim to avoid. We present novel algorithms for computing linear path skylines which are considerably faster and more memory efficient than state-of-the-art path skyline algorithms.

First, we present a method for computing linear path skylines in bicriteria networks. Our approach is a label-correcting algorithm which exploits the particular properties of linear path skylines in two-dimensional cost spaces. Second, we extend our solution to arbitrarily dimensional cost spaces, introducing the algorithm LSCH. LSCH constructs the linear path skyline iteratively, adding result paths generated by cost-optimal path searches w.r.t. specific combinations of cost criteria. These combinations of cost criteria depend on the hyperplanes which currently enclose the skyline in its cost space. To execute cost-optimal path searches most efficiently, the query-dependent lower bounds generated by ParetoPrep may be employed.

Although the linear skyline is usually much smaller than the conventional skyline, the

amount of result paths still might exceed the amount of alternative paths that are of use in practice. Thus, we present an approximate variant of our algorithm called ε-LSCH which further restricts the number of results.

To conclude, the key contributions of this part are as follows:

- We present ParetoPrep, an efficient algorithm for the computation of optimal lower bounds in multicriteria networks. We prove the correctness of ParetoPrep as a preprocessing step for the computation of path skylines and show that it computes the cost-optimal paths for all criteria as a byproduct.
- We define and discuss the linear path skyline in two dimensions and introduce efficient means for its computation.
- We extend the definition of the linear path skyline to arbitrarily many dimensions, discussing and proving its properties. Besides, we introduce the further restricted set of linear skyline paths, the ε-LSCH skyline. Finally, we present efficient algorithms for the computation of both sets.
- All of the above approaches are evaluated on real-world data sets to substantiate the practicability of the presented ideas.

Structure of this part: Chapter 5 provides a review of existing research on cost estimations as used by routing algorithms, on methods for skyline computation as well as on works related to the notion of the linear skyline. In Chapter 6, the topic of multicriteria lower bounds is explored in detail, providing computation techniques, proof of validity and an in-depth analysis. With optimal lower bounds at hand, Chapters 7 and 8 introduce, investigate and evaluate linear path skyline computation, first for networks with two cost criteria, then for arbitrary multicriteria networks. This research has been previously published in [129, 131, 126].

Chapter 5

Related Work

In this section, we review research relevant to the topics of lower bounds and (linear) path skylines. First, we explain the existing methods of lower bound computation in greater detail and point out their shortcomings. Then, we turn to state-of-the-art algorithms for (linear) path skylines, stressing that performance crucially depends on tight lower bounds.

As mentioned before, query-independent bounds are often derived by the ALT principle. For given landmarks, the optimal path costs of each node to and from each landmark are precomputed and stored. For a specific query, computing the optimal path from a node v to a target t, the triangle inequality yields a lower bound for $c(v, t)$. Choosing the maximum value among all landmarks ensures the tightest possible bound. In the original paper [53], the cost attribute was distance, and the approach was compared to Euclidean distance bounds which were significantly outperformed. However, the quality of the bounds depends significantly on the choice of landmarks, a relationship which has been analyzed thoroughly [55, 33].

The approach may also be applied to other cost criteria or multiple criteria at once using a so-called reference node embedding [80]. Lower bound cost vectors which approximate the costs of multiple criteria may be used to accelerate path skyline computation. Let us note that analogously upper bounds may be computed [80]. Bounds computed in this manner are query-independent as they are computed during an offline preprocessing phase prior to any query. This has several drawbacks albeit benefiting efficiency at query time. First, memory requirement is relatively high, $\mathcal{O}(Nkd)$ where N is the number of nodes in the network (usually in the millions), k is the number of landmarks (usually in the tens) and d is the number of criteria (usually not higher than five). Second, in dynamic networks – where edge costs vary over time –, these lower bounds may lose their bounding properties and require costly recalculation. And, finally, the quality of the approximation is often insufficient for the majority of queries. Due to the query-independence, the bounds are relatively loose which has great impact in multicriteria networks.

In contrast, query-dependent bounds require a precomputation step before every query. The state-of-the-art method for computing lower bounds is [148] (or [93] in bicriteria networks) which, for a given start and target pair, conducts d reverse Dijkstra searches from the target to all nodes, where d denotes the number of criteria. This means, the whole

graph is visited d times, and the memory requirement is $\mathcal{O}(Nd)$, where N is the number of nodes. However, the memory requirement is not permanent, as the precomputed bounds may be dropped upon answering the query. The information provided by this precomputation is exact, de facto optimal path costs are calculated. This yields enormous acceleration when executing the actual query, e.g., a path skyline computation. In fact, the results of our experiments show that this method clearly outperforms the use of a reference node embedding for computing path skylines with more than two criteria.

Both types of lower bounds find application in path computation algorithms. They enable goal-direction and allow for pruning of branches of the search tree. We now give a brief introduction to the topic of skyline queries, followed by a survey of path skyline algorithms, closing with an overview of methods for linear path skyline computation.

In the database community, the skyline operator was first introduced by [8]. Given a set of positive vectors, the aim was to find the maximal set of non-dominated vectors, where domination was defined as in Definition 2.6. Multiple approaches to compute skylines in database systems on sets of cost vectors followed [142, 79, 110]. Several variations of the skyline problem on vectors have been proposed [11, 87, 112]. As mentioned before, when computing the skyline among cost vectors, the set of vectors is given. This is not the case when computing path skylines where it is infeasible to materialize all possible paths from a given start to a given target node.

Therefore, although related, the task of computing path skylines (or route skylines [80]) bears different algorithmic challenges. The goal is to find the maximum set of non-dominated paths. In Operations Research, the problem of computing path skylines is known as multiobjective shortest path problem, and surveys on existing solutions to this problem can be found in [137, 115, 143, 34]. Early on, [61] showed that the size of the path skyline may increase exponentially with the number of edges between start and target and that the problem is NP-hard. More recently, [103] showed that the number of paths is in practice feasibly low when using strongly correlated cost criteria. The current state of the art of computing path skylines are label-correcting methods [80, 141, 158] relying on lower bounds. In the special case of bicriteria networks, there exist specific optimizations [10, 138]. However, they are not generalizable to more criteria, therefore, we clearly distinguish between bicriteria and multicriteria cases in the rest of this part.

The works mentioned so far aim at computing the set of all pareto-optimal paths, i.e., the whole path skyline. It is our goal to restrict the set of paths returned to the user and only compute the linear path skyline, a subset of conventional skyline. In bicriteria networks there exist some approaches to directly computing the linear path skyline [116, 99] which are all outperformed by the label-correcting approach presented in Chapter 7.

For the task of computing linear path skylines in multicriteria networks (with more than two criteria), there exists no designated algorithm. However, the authors of [108] propose an algorithm called ExA for computing multiobjective linear programs. Since for computing the linear path skyline a multiobjective linear program formulation exists, ExA can be modified to process linear path skyline queries in multicriteria networks. Let d denote the number of cost criteria in the network. ExA searches all sets of cost vectors with d elements, i.e., those sets which specify a hyperplane in the cost space. Given such

a set of cost vectors, ExA solves a linear equation system to determine the normal vector of the respective hyperplane. This hyperplane is in turn used to decide whether further computation is needed. The major drawback of ExA is the combinatorially large number of cost vector sets with d elements. Therefore, we propose a different approach which works on the facets of the convex hull. The difference is that there are far less facets than d-sized cost vector sets. Thus, the amount of linear equation systems to be solved is considerably smaller. Since ExA is the only approach for computing linear path skylines in multicriteria networks so far, we compare to ExA in our experiments.

Let us note that the linear path skyline is a subset of the vertices on the convex hull of cost vectors. We will therefore formally discuss the relationship of the linear skyline and the convex hull in Section 8.2. Our algorithm makes use of multidimensional convex hull computations to prune the search space. More precisely, we employ the Beneath-and-Beyond approach [113] which incrementally adds new points to the convex hull. The method starts with a trivial point set spanning a d-dimensional hyperplane. Given a new point p, it is tested whether p is inside the hull. If so, the next point is examined. If p is not inside the hull, the algorithm removes the facets shaded by p and computes a set of new facets. Although there exist more recent and more sophisticated methods of convex hull computation, Beneath-and-Beyond fits the requirements of our algorithm optimally. We rely on the computation of facets when a new point is added to the hull. More recent convex hull algorithms like quick hull [5] mainly optimize the selection of new points to add to the hull. However, we do not benefit from this kind of improvement as our algorithm asserts that each point that is added is indeed part of the hull.

For the notions of multicriteria networks, paths and their cost vectors, we follow Section 2.4. In general, we do not limit the number of criteria. If results presented in this part pertain to bicriteria networks only, it is mentioned accordingly. Throughout this part, we assume G to be a d-dimensional multicriteria network unless explicitly stated otherwise. s and t refer to start and target nodes, respectively. Recall that when comparing paths, we assume the same start and target. Further definitions regarding the respective chapters will be given later.

Chapter 6

Optimal Lower Bounds for Path Skyline Computation

6.1 Multicriteria Lower Bounds

This section formalizes different pruning concepts which lay the foundation for ParetoPrep.

Definition 6.1 (Lower Bound Costs)**.** *Let c_i be a cost criterion. If for all nodes n and m holds $lb_i(n,m) \leq p_i$ for any path p connecting n and m, then lb_i is called a* lower bound *for criterion c_i. By $lb(n,m)$ we denote a vector consisting of lower bounds for all cost criteria of the given multicriteria network, referred to as* lower bound (cost vector)*.*

A lower bound vector contains a lower bound for each of the cost criteria. Thus, for an arbitrary path p connecting nodes n and m and for any mcf f holds, $f(lb(m,n)) < f(c(p))$. In other words, the image of a lower bound vector under an mcf is also a lower bound for the costs of all paths between m and n. The lower bounding property is invariant under mcfs.

We refer to a lower bound vector $lb(n,t)$ as *optimal* iff for all $1 \leq i \leq d$ there exists a path $p_i = ((n,m_1),\ldots,(m_k,t))$ with $\mathrm{lb}_i(n,t) = p_i$, i.e., if the lower bound value of each criterion is indeed attained by a specific path. The bounds computed by the state-of-the-art method Multi-Dijkstra and the proposed method ParetoPrep are optimal. Furthermore, these bounds are query-specific, i.e., depending on the given target node t. Since any node is a potential start and target node, it is not feasible to precompute these bounds. Both methods compute these bounds at query time. In contrast, the lower bounds provided by a reference node embedding [80] or other ALT methods [53] are not specific to a particular query. Thus, they are usually not optimal. As will be shown in the experiments, for more than two cost criteria and larger distances between start and target, optimal bounds compensate for the additional overhead. In the following, we will explain how lower bounds are employed in algorithms for finding cost-optimal and pareto-optimal paths in multicriteria networks.

Note that computing an optimal lower bounds implies computing the costs of cost-optimal paths from s to t w.r.t. all given cost criteria. Thus, a method for computing

optimal lower bounds can be extended to computing the set of single-criterion cost-optimal paths. Of course, additional criteria may be introduced by combining cost criteria. This can be useful if particular trade-offs (e.g., 40% toll fees, 60% energy consumption) are known in advance. Adding such "hybrid" criteria to the catalogue of original criteria, it is possible to compute the single-criterion cost-optimal paths for the entirety of criteria. If the desired trade-offs or combination functions are not known in advance, computing (linear) path skylines is recommended, in order to provide the user with a set of optimal paths.

For (linear) skyline computation, state-of-the-art algorithms rely on two ways of pruning the search space. We refer to them as (local) domination and global domination. Local domination was introduced in Definition 2.6. The difference of local and global domination will be highlighted shortly. For a path $q = ((n_1, n_2), \ldots, (n_{l-1}, n_l))$, any consecutive subsequence of edges $p = ((n_i, n_{i+1}), \ldots, (n_{k-1}, n_k)), i \geq 1, k \leq l$ is called a subpath of q.

Lemma 6.1. *Any subpath of a (locally) non-dominated path is (locally) non-dominated (w.r.t. its respective start and end nodes).*

Proof. Suppose the statement would not hold. Then there existed a non-dominated path p from some node u to a node v with a dominated subpath q from u to some intermediate node w. Let q' denote the path from u to w which dominates q. Replacing q in p by the subpath q', we obtain a path p' from u to v which dominates p. This is a contradiction to the assumption. Hence, any subpath of a non-dominated path is non-dominated [80]. □

In contrast to Dijkstra's algorithm which only keeps the best value and the predecessor at each visited node, skyline algorithms need to maintain all non-dominated paths ending at each node. [61, 94, 80] keep such local skyline of paths for each visited node n. Lemma 6.1 guarantees that only locally non-dominated paths may be part of the global skyline. As a consequence, any locally dominated path may be pruned. This fact constitutes the first domination check referred to as *local domination* check. Although local domination allows for pruning non-promising paths, it is not capable of restricting the search space by goal-direction. For this, lower bounds are employed.

Definition 6.2. *Given start and target nodes s and t, an arbitrary path p is called globally dominated iff it is a subpath of a dominated path q between s and t. Conversely, any subpath p of a non-dominated path q between s and t is called globally non-dominated.*

Thus, a path p from s to some node n may be excluded from further exploration if its best possible extension (given by a lower bound) of p to the target t is dominated in the global skyline.

Lemma 6.2. *Let p be a path from node n to node m and q be a path from the start node s to the target node t. Let lb be a lower bound. p is* globally dominated *iff*

$$q \prec_{dom} lb(s, n) + p + lb(m, t)$$

Proof. The cost vector $lb(s,n) + c(p) + lb(m,t)$ is a lower bound cost of all paths from s to t via p. If this lower bound cost is dominated by the cost of a path from s to t, there is no non-dominated path from s to t via p. □

Our reference algorithm for path skyline computation [80] employs global domination checks as soon as the first skyline path is known. Recall that by computing query-specific lower bounds we obtain all single-criterion optimal paths. Any of the single-criterion optimal paths is part of the skyline and therefore may serve as an initial comparison partner for global domination checks.

6.2 Multicriteria Lower Bound Computation

Our method for computing multicriteria lower bounds is called *ParetoPrep*. Just as Multi-Dijkstra (MD), ParetoPrep computes query-dependent optimal lower bounds. In contrast to MD which visits the whole graph for each cost criterion, ParetoPrep needs only one partial graph traversal. However, it will be shown that there cannot be a node which is part of a skyline path and which is not visited during this partial graph traversal. Thus, ParetoPrep is skyline-correct in the sense that it visits all nodes required for computing a path skyline.

The pseudocode of ParetoPrep is provided in Algorithm 10, an exemplary output and execution of the algorithm are shown in Figures 6.1 and 6.2, respectively. The benefit of bounds provided by ParetoPrep is illustrated in Figure 6.3 where the visited portion of the graph for a skyline computation (with ARSC [80]) is visualized. On the left no bounds were employed, on the right, ARSC used the lower bounds provided by ParetoPrep, drastically reducing the search space.

ParetoPrep maintains a set of open nodes *open* and a set $\mathcal{S}$ of paths from s to t. It starts traversing the graph at t, following incoming edges. Each visited node n has an entry consisting of two elements $\{lb(n,t),\ succ_i(n)\}$. The cost vectors $lb(n,t) : \mathcal{V} \times \mathcal{V} \to \mathbb{R}^d_{\geq}$ are tentative lower bounds. During processing, the values are relaxed. Upon termination, the $lb(n,t)$ are in fact optimal lower bounds. The edges $succ_i(n) : \mathcal{V} \times \mathbb{N} \to \mathcal{E}$ are the outgoing edges of n along the tentative cost-optimal path from n to t for criterion i. These successor edges are used to reconstruct the cost-optimal paths upon reaching s in the backward traversal. An entry of an unvisited node n is assumed to be $lb_i(n,t) = \infty$, $succ_i(n) = \emptyset$ for all i. $lb(t,t)_i$ is always zero because the lower bound cost of reaching t from t are zero.

Initialization The open set is created, and the target node t is added to the open set.

Node Selection In each iteration, an open node n is selected and removed from the open set. To reduce the number of nodes which have to be visited twice, the nodes closest to t should be visited first. We suggest to choose $\arg\min_{n \in \text{open set}} \sum_i lb_i(n,t)$.

Algorithm 1 Pseudocode of ParetoPrep

in : multicriteria network, start s, target t
out: optimal lower bounds, all single-criterion cost-optimal paths

```
// step 1: initialization
```
$\mathcal{S} \leftarrow \emptyset$ and $open \leftarrow \{t\}$ **while** $open \neq \emptyset$ **do**
```
    // step 2: node selection
```
    select n with minimal lower bound sum from $open$ and remove from set
```
    // step 3: global domination
```
    if $p \prec_{dom} lb(n,t) + lb(s,n)$ *for some* $p \in \mathcal{S}$ **then**
        skip step 4 and 5
    end
```
    // step 4: node expansion
```
    foreach *incoming edge* (m,n) *of* n **do**
        foreach *criterion* i **do**
            if $lb_i(n,t) + (m,n)_i < lb_i(m,t)$ **then**
                $lb_i(m,t) \leftarrow lb_i(n,t) + (m,n)_i$
                $succ_i(m) \leftarrow (m,n)$
                $open \leftarrow open \cup \{m\}$
            end
        end
    end
```
    // step 5: path construction
```
    if $lb(s,t)$ *was modified in step 4* **then**
        foreach *modified component* i *of* $lb(s,t)$ **do**
            $p \leftarrow$ constructpath$(s,t,succ_i,i)$
            $\mathcal{S} \leftarrow \mathcal{S} \cup \{p\}$
            remove paths dominated by p from $\mathcal{S}$
        end
    end
end

Global Domination The third step is a check if the selected node has to be extended. If $lb(s,n) + lb(n,t)$ is dominated by the costs of a known path, step 4 and 5 are skipped. The cost vector $lb(s,n)$ is the lower bound cost of all globally non-dominated paths from s to n. If no such information is available $lb(s,n) = 0$.

Node Extension The fourth step is the expansion of the selected node. The i-th cost of each neighboring node m is set to the minumum of $lb_i(m,t)$ and $lb_i(n,t) + (m,n)_i$, where (m,n) is the edge from m to n. For each criterion i for which $lb_i(m,t)$ is relaxed, the i-th predecessor edge $succ_i(m)$ is set to (m,n). If any entry of $lb(m,t)$ was changed and m is not the start node s, m is added to the set of open nodes.

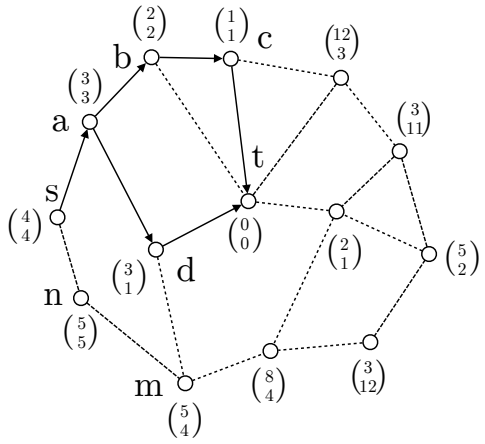


Figure 6.1: Exemplary output of ParetoPrep given a start node s and a target node t. The indicated paths $\{s, a, b, c, t\}$ and $\{s, a, d, t\}$ are the cost-optimal paths for the first and the second criterion, respectively. The vectors next to each node are the computed lower bound costs lb of reaching t from the respective nodes.

Algorithm 2 Pseudocode of ParetoPrep's path construction routine

in : s, t, $succ_i$, i
out: current cost-optimal path from s to t for criterion i

```
1 m ← s  p ← new empty path  while m ≠ t do
2 |   (m, n) ← succ_i(m)
3 |   p ← p extended with (m, n)
4 |   m ← n
5 end
6 return p
```

Path Construction If in the previous step $lb(s, t)$ was modified path construction is called. For each modified cost criterion the currently cost-optimal path from s to t is constructed. The paths are constructed by following $succ_i$ of nodes, similarly to how paths are reconstructed in Dijkstra's algorithm. The pseudocode of the routine is given in Algorithm 2.

Termination The sixth step is termination. If after an iteration there are no more open nodes the algorithm terminates, otherwise the algorithm continues with step 2. Upon termination, $\mathcal{S}$ contains a cost-optimal path s to t for each criterion, and lb maps each node which is possibly part of a non-dominated path from s to t onto its lower bound costs of reaching t.

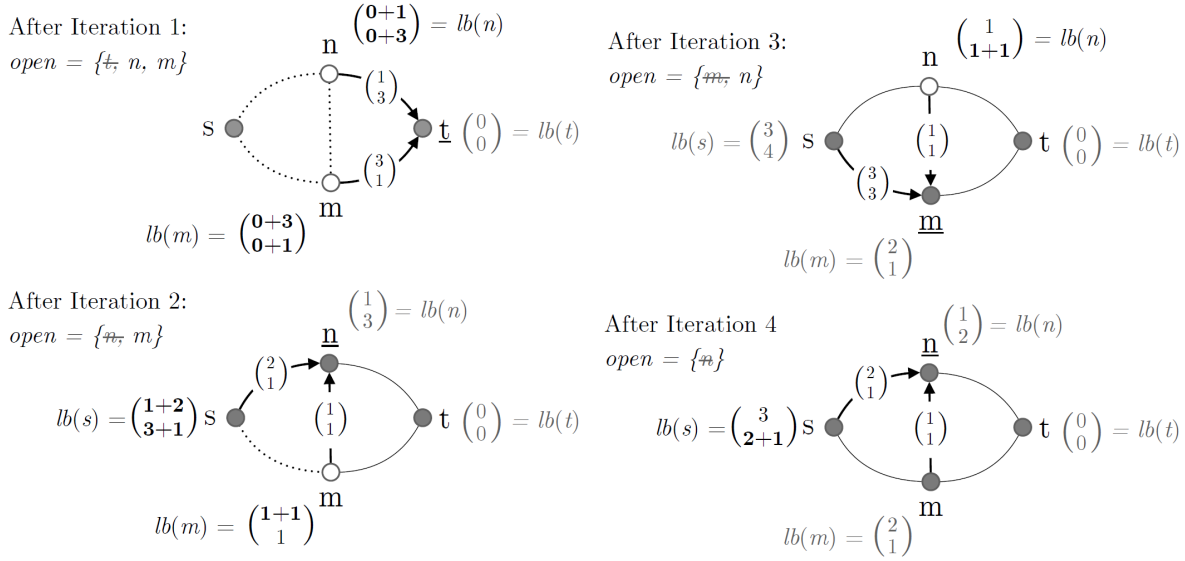


Figure 6.2: Exemplary execution of ParetoPrep. Active node of iteration is underlined. After Iteration 2 path through $[s, n_1, t]$ with costs $[3, 4]$, after iteration 3 path through $[s, n_1, n_2, t]$ with costs $[6, 3]$ is constructed. ParetoPrep terminates after iteration 4.

6.3 Correctness and Termination

In this section, we prove that ParetoPrep does indeed visit the portion of the graph necessary to compute a path skyline. More precisely, we will show that every node which is part of a non-dominated path from start to target is visited.

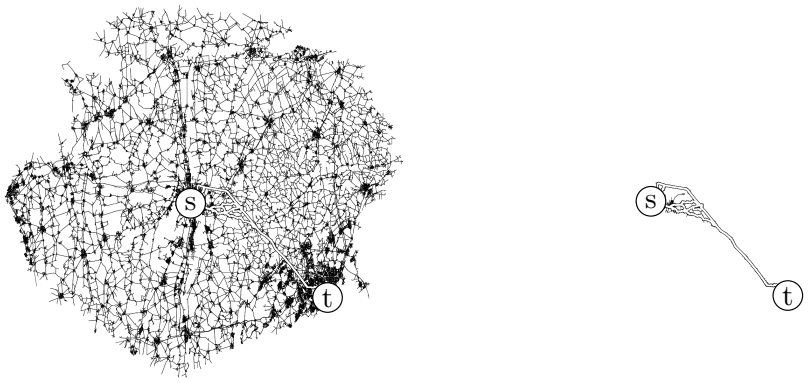
Definition 6.3. *Let $\mathcal{R}(n)$ denote all paths through which ParetoPrep reached a node n at a particular point during the execution (for which we refrain from introducing new notation). For every $n \neq t$, we initialize $\mathcal{R}(n) = \emptyset$. In each iteration, every neighbor m of the selected node n is reached through the edge connecting the two nodes*

$$\mathcal{R}(m) = \mathcal{R}(m) \ \cup \ \{p \text{ extended by } (m, n) \mid p \in \mathcal{R}(n)\}$$

Note that in this case (m, n) becomes the first edge of the path extended with (m, n). This is because ParetoPrep follows incoming edges, moving backward from the target node. For the rest of this section, when we speak of (non-)domination, we mean global (non-)domination.

Lemma 6.3. *At the end of each iteration, $lb(m, t)$ equals the lowest costs of all paths through which m was reached, i.e., $lb_i(m, t) = \min_{p \in \mathcal{R}(m)} p_i$ for all i and m.*

Proof. For an arbitrary criterion i, we prove the statement by induction over k, the number of hops (edges along cost-optimal path w.r.t. criterion i) to the target.
$k = 0$: The statement trivially holds for the target node.



without ParetoPrep with ParetoPrep

Figure 6.3: Comparison of search areas for a routing task from Augsburg (s) to Munich (t) with two cost criteria. The illustration compares the search area of the path skyline algorithm ARSC [80] without lower bounds to that with ParetoPrep bounds. It is easily observed that the search explores almost no dominated paths when using the information provided by ParetoPrep.

$k \to k+1$: Let m be a node which is $k+1$ hops from the target. Suppose

$$lb_i(m,t) = \min\{lb_i(n,t) + (m,n)_i \mid \exists\ (m,n) \in E\} < \min_{p \in \mathcal{R}(m)} p_i$$

In particular, $lb_i(n,t) < \min_{p \in \mathcal{R}(n)} p_i$ which contradicts the assumption that the statement holds for nodes k hops from the target. Hence, the statement holds. □

Lemma 6.4. *If a node is reached by a non-dominated path, it is expanded.*

Proof. ParetoPrep expands every node m with two exceptions: (1) if global domination holds in Step 3, or (2) if the m is not added to the *open* set in Step 4. Of course, if m is reached through a non-dominated path, global domination does not hold. Hence, the node is expanded, unless $lb_i(n,t) + (m,n)_i \geq lb_i(m,t)$ for all of the criteria. If m was previously unvisited, then $lb_i(m) = \infty$ and will therefore always be relaxed. If m was previously visited and reached by a non-dominated path, then $lb_i(n) + (m,n)_i < lb_i(m)$ must hold for some i. This means, m is added to the open set and expanded subsequently. This proves the statement. □

Lemma 6.5. *Upon termination, each node n which is part of a non-dominated path from s to t was reached. Furthermore, the node n is reached through each non-dominated path from n to t.*

Proof. Let p be an arbitrary non-dominated path from n to t. If no such path exists for n, then n is not contained in any non-dominated paths from s to t. Let K be the number of nodes through which p passes. Let $p^{(k)}$ be the k-th node through which p passes. Let

$p^{(j,k)}$ be a subpath of p which starts at $p^{(j)}$ and ends at $p^{(k)}$. If the claim were incorrect, there would exist some k-th node $p^{(k)}, k < K$, which would not be reached. This implies one of the two cases:

(1) $p^{(k+1)}$ was reached by $p^{(k+1,K)}$, but not expanded afterwards

(2) $p^{(k+1)}$ was not reached

Since $p^{(k+1,K)}$ is a subpath of the non-dominated path p and thereby non-dominated itself, it must be expanded by Lemma 6.4 which contradicts (1). (2) is the inductive shifting of the original statement that $p^{(k)}$ was not reached. This implies two cases, as above. The first one is contradicted as before, the second one is again the inductive shifting. Following the chain of induction, we get $p^{(K)} = t$ was not reached which is contradicted by the empty path starting at t. Thus, n is reached by all non-dominated paths from n to t. □

The above lemmas prove that ParetoPrep is sufficient for path skyline computation, i.e., every node that is possibly part of a skyline path is indeed visited. In addition, let us state precisely, what the values of the lower bound costs are and how they are related to single-criterion cost-optimal paths.

Claim 6.1. *Let n be a node contained in a non-dominated path from s to t. The cost vector $lb(n,t)$ equals the lower bound costs of all non-dominated paths from n to t. Furthermore, a cost-optimal path from s to t for each criterion is found.*

Proof. The first statement follows directly from Lemma 6.5 and Lemma 6.3. If n is contained in a non-dominated path from s to t, n is reached through all non-dominated paths from n to t. Hence, $lb(n,t)$ equals the lower bound costs of all non-dominated paths from n to t. Now, let us investigate the special case of the start node s. The single-criterion cost-optimal paths are obviously a subset of the non-dominated paths. Hence, s is reached by all single-criterion cost-optimal paths which are reconstructed in Step 5 of Algorithm 10. □

Finally, let us note that ParetoPrep always terminates if executed on finite graphs. This is due to the fact that each previously visited node n is expanded if at least one criterion of $lb(n,t)$ has been lowered. Once a node is reached by the cost-optimal paths for each criterion, it will not be expanded anymore. From a finite number of nodes follows a finite number of paths between nodes which in turn implies that ParetoPrep performs a finite number of iterations.

This concludes our section on the properties and the correctness of ParetoPrep. In the following, we will explore the efficiency and performance of ParetoPrep.

6.4 Evaluation

We evaluate ParetoPrep (PP) on settings based on the real-world road network of the state of Bavaria, Germany, with 1 023 561 nodes and 2 418 437 edges, extracted from OpenStreetMap[1] (OSM) using the MARiO framework [58]. All experiments were conducted on

[1] www.openstreetmap.org/

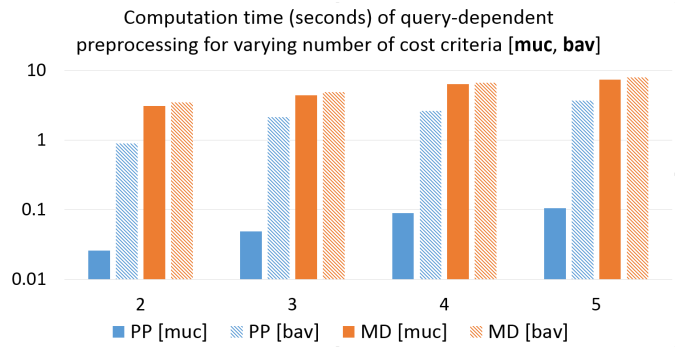
a dedicated machine with an Intel i7 CPU (3.4 GHz) and 32 GB RAM, running Windows 8. Different algorithms are tested on the same randomly generated scenario before comparing results. Runtime evaluations are based on Java's nanotime clock and performed for each algorithm individually.

First, we compare the query-dependent preprocessing steps PP and Multi-Dijkstra (MD) in terms of computation time. Given a start and target pair, we evaluate how long the preprocessing step takes. Second, for PP, MD and the reference node embedding (RNE), we investigate the quality of the respective bounds combined with the preprocessing time, i.e., the actual performance gain for different algorithms. For RNE, we select nine reference nodes on a uniform grid over the map which is a typical choice. Furthermore, we assume that there is no overhead for creating the embedding, as it is a precomputation step during the offline phase. In a first subsetting, we examine how the path skyline algorithm ARSC benefits from the bounds provided by PP, MD and RNE. Given the information of the respective preprocessing step and a start and target pair, we take the runtime of the algorithm plus computation time of the query-dependent bounds as a measure for the quality of the bounds. In a second subsetting, we analyze how the linear path skyline algorithm LSCH, detailed in Chapter 8, benefits from the bounds, again in terms of runtime. Finally, we evaluate PP as a means for single-criterion cost-optimal path computation for given cost criteria. In a multicriteria network with d criteria, PP can be used to compute the cost-optimal paths w.r.t. each of the criteria, as produced by d distinct Dijkstra searches. We compare PP to these multiple single-source single-target Dijkstra searches w.r.t. runtime and visited portion of the graph. Note that in order to compensate runtime effects in the virtual machine, runtime is measured by performing each task five times and taking the minimum of these runs.

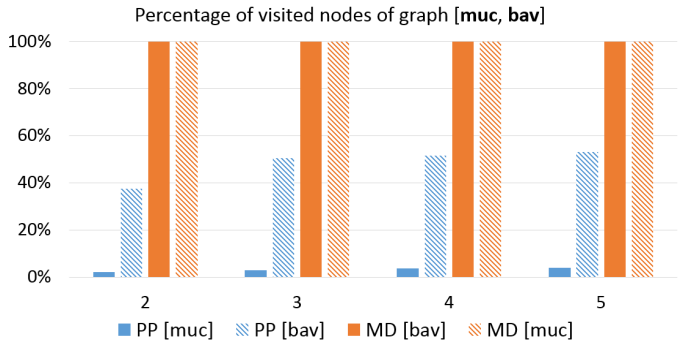
We evaluate the above scenarios on two settings based on the road network graph of Bavaria, Germany. The first one is rather local and set in Munich, capitol of Bavaria, routing from one of the 25 district centers to another, amounting to $\binom{25}{2} = 300$ pairs in total. We refer to this setting as **muc**. The other setting is – relative to the graph – rather global, routing from one of the 5 major cities in Bavaria to all others, amounting to $\binom{5}{2} = 10$ pairs in total. We refer to this setting as **bav**. The cost criteria used are distance (dist), travel time (tt), ascent (asc), penalized travel time (ttpen), and energy expenditure (ener). The basic travel time estimate (tt) assumes travel speeds equal to the speed limits and no delays at crossings. The penalized travel time estimate (ttpen) assumes additional 30 seconds for each traffic light. The energy expenditure estimate (ener) assumes that 0.2 kWh are lost on friction per kilometer, which is a rough estimate derived from typical battery capacities of electric cars and their respective ranges. For each ascended kilometer 4 kWh are added to the energy usage, which is derived from the increase in potential energy from ascending 1 000 meters with a 1 500 kg vehicle.

$$1\,000 \text{ m} \cdot 1\,500 \text{ kg} \cdot 9.81 \frac{\text{m}}{\text{s}^2} \cdot \frac{1 \text{ kWh}}{3.6 \cdot 10^6 \text{ J}} = 4.0875 \text{ kWh}$$

For each descended kilometer 2 kWh are subtracted from the energy loss and negative energy loss values are corrected to zero. The employed formula for the energy loss in



(a) Computation Time



(b) Visited Nodes

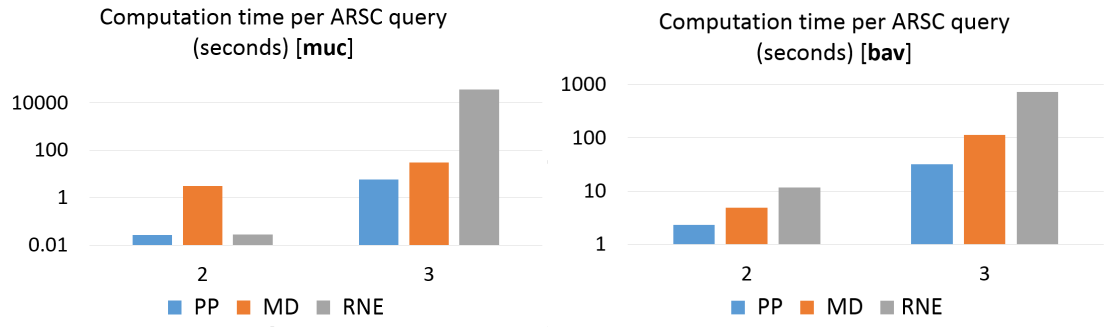
Figure 6.4: Computation time (seconds) and percentage of visited nodes (of all nodes in the graph) of the query-dependent preprocessing steps PP and MD for the settings **muc** and **bav**.

kWh for a road segment from n to m with length $len(n, m)$, ascent $asc(n, m)$ and descent $desc(n, m)$ in kilometers is

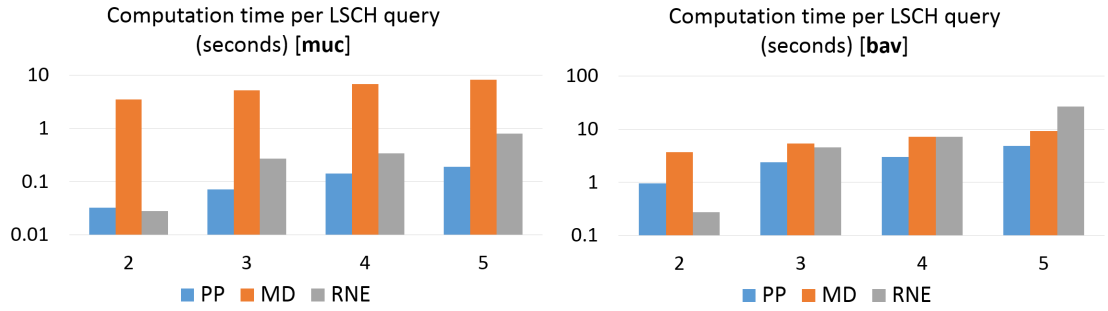
$$\max(0,\ 0.2 \cdot \text{len}(n, m) + 4 \cdot \text{asc}(n, m) - 2 \cdot \text{desc}(n, m))$$

where ascent and descent are derived from OSM data. Let us stress that in order to validate the efficiency of the proposed approach, the cost criteria are not required to be realistically modeled; we do not claim so for tt, ttpen, ener. We performed queries using the following selections of criteria:
2: dist+tt, 3: dist+tt+asc, 4: dist+tt+asc+ttpen, 5: dist+tt+asc+ttpen+ener

Figure 6.4 compares the preprocessing times of the query-dependent methods PP and MD. It shows the computation time in seconds when varying the number of cost criteria for both settings, **muc** and **bav**. We observe that PP always outperforms MD; independent of the number of cost criteria, PP is always around two orders of magnitude faster than MD, hardly ever exceeding 100 milliseconds of computation time. Both methods compute optimal bounds. The main reason for this behavior can be observed in the lower part of Figure 6.4. PP only visits a very restricted part of the graph, yet, the necessary part of



(a) ARSC



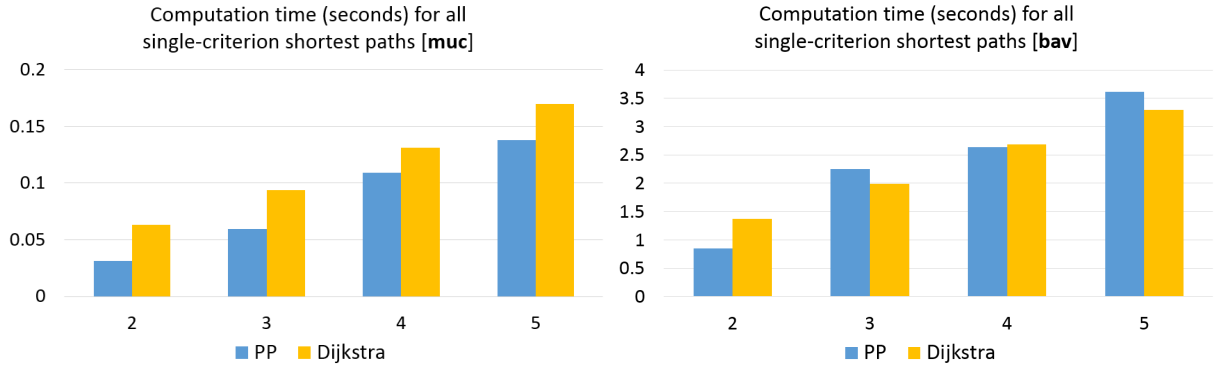
(b) LSCH

Figure 6.5: Average computation time (seconds) of ARSC and LSCH given the precomputed bounds by the respective methods, evaluated on the **bav** scenario.

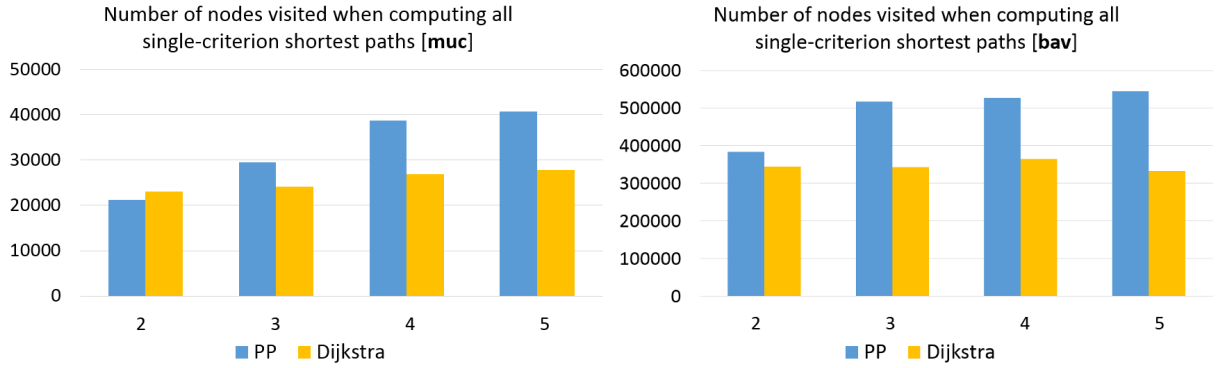
the graph to compute all pareto-optimal paths. For the **muc** setting, it visits at most 6% of the network and even for examples with large distances (**bav**), PP visits only about half of all nodes. As explained before, MD visits the whole graph for each of the criteria.

The impact of the bound quality can be observed in Figures 6.5. The optimal bounds computed by PP and MD accelerate both algorithms significantly. While LSCH is evaluated for the same cost criteria as above, ARSC is evaluated for two and three cost criteria only, as it becomes infeasibly slow for more dimensions. Note that this is a drawback of the algorithm itself, not the bound quality. Of course, the significant discrepancies in runtime are due to algorithmic details and the different result sets (ARSC computes the path skyline, LSCH computes the usually smaller linear path skyline). For both algorithms RNE yields slightly faster runtimes in the two-dimensional case. However, the time for creating the RNE is not factored in. In higher dimensions the suboptimal quality of the query-independent bounds of the RNE becomes evident. Overall, the bounds computed by PP yield an ARSC speed-up between five times and two orders of magnitude for both scenarios. A similar acceleration is achieved for LSCH. Note that even in the complex scenarios with five cost criteria, PP yields fast processing times.

As a byproduct to its bounds, PP computes the single-criterion cost-optimal paths w.r.t. all d cost criteria. In the following, we want to compare this task to d separate



(a) Computation time



(b) Visited nodes

Figure 6.6: Computation time (seconds) and visited nodes when computing all k single-criterion cost-optimal paths with PP and k Dijkstra searches.

single-source single-target Dijkstra searches. Note that this is not the same procedure as MD which performs all-source single-target searches and cannot terminate upon arrival at the source. Figure 6.6 visualizes computation time and number of visited nodes for both approaches and scenarios. Remarkably, although PP visits significantly more nodes, it is faster than the separate Dijkstra searches for the **muc** scenario. This is because PP only requires a single graph traversal. However, when the discrepancy regarding the number of visited nodes becomes too large – as in the **bav** scenario – PP is marginally slower than the Dijkstra approach. Of course, PP visits more nodes than separate Dijkstra searches because it visits all nodes relevant to any skyline path, as proved in Lemma 6.5. In contrast, the nodes visited by the Dijkstra searches will in general not suffice to build the path skyline. Hence, considering the additional information which PP acquires during its single graph traversal, the overall processing time is unrivaled.

Chapter 7

Linear Path Skylines in Bicriteria Networks

The conventional path skyline may contain exponentially many paths. Besides degenerating processing time and memory consumption, the possibly exponential number of paths implies less meaningful results, often hindering interpretation and usability. Therefore in this chapter and the following, we propose to use a different notion of optimality which can lead to faster computation and less results.

7.1 Preliminaries

All skyline paths are optimal w.r.t. some monotone combination function (mcf), as established in Chapter 4. To tackle the shortcomings of the path skyline, we propose to alter the definition of optimality. Instead of computing the paths optimal w.r.t. monotone cost functions, we propose to compute the paths optimal w.r.t. the simplest subclass of monotone functions, the weighted sums (or linear combinations). According to Definitions 2.6 and 2.7 we present the equivalent definition for linear path skylines.

Definition 7.1. *Let s and t be nodes in, and let $\mathcal{P} = \mathcal{P}(s,t)$ be the set of paths connecting s and t. A subset $\mathcal{S}_L \subseteq \mathcal{P}$ is called* linear (vector) skyline*, iff the following two conditions hold:*

(i) Completeness:

$$\forall\ 0 \neq w \in \mathbb{R}^d_{\geq 0}\ \exists\ x \in \mathcal{S}_L \text{ such that: } w^T x = \min w^T y\ \forall\ y \in \mathcal{P}$$

(ii) Minimality:

$$\forall\ x \in \mathcal{S}_L\ \exists\ 0 \neq w \in \mathbb{R}^d_{\geq 0} \text{ such that: } w^T x < w^T y\ \forall\ x \neq y \in \mathcal{P}$$

While the conventional skyline requires its paths (or vectors) to be optimal w.r.t. some mcf, the linear skyline requires the same but for non-negative weight vectors. Since any

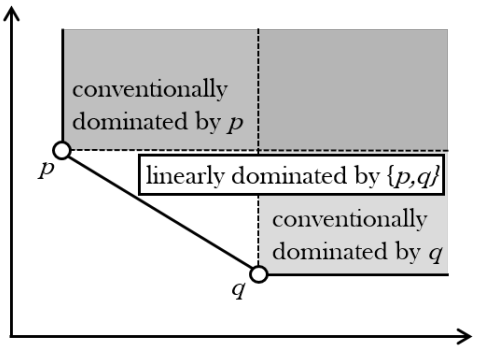


Figure 7.1: Illustration of areas which are dominated conventionally and linearly.

weighted sum with a non-negative weight vector is monotone, the linear skyline is a subset of the conventional skyline. While the conventional skyline is defined by its notion of domination, it is more natural to define the linear skyline by completeness and minimality. Similar to the conventional skyline, however, the linear skyline may be defined as a set of non-dominated paths (or vectors).

Definition 7.2. *Let s, t and $\mathcal{P}$ as before. p is* linearly dominated *by the paths in $\mathcal{P}$, iff*

$$\forall\ 0 \neq w \in \mathbb{R}^d_{\geq 0}\ \exists\ q \in \mathcal{P} \text{ such that: } w^T q < w^T p$$

The maximal set of non-dominated paths is referred to as linear (path) skyline *of $\mathcal{P}$ and denoted by $\mathcal{S}_L(s,t)$ or $\mathcal{S}_L$ for short.*

Note that linear domination (in two dimensions) may hold because of two reasons:

- p' is conventionally dominated by some path p. In this case, the inequality holds for any mcf, particularly for any weighted sum with a non-negative weight vector w.
- p' lies "behind" a line between the cost vectors of two linearly non-dominated paths p, q (viewed from the origin). In this case, $w^T p' > w^T q = w^T p$ for the normal vector w of the hyperplane which is the extension of the line between p and q.

This is illustrated in Figure 7.1. Unless conventional domination is the case, linear domination requires the computation of normal vectors of hyperplanes. Hence, conversely to the linear skyline which is a subset of the conventional skyline, the area of linear domination is a superset of the conventional domination area.

In this chapter, we explore linear skylines in bicriteria networks, relying on properties which only hold in the bicriterion case. Later, in Chapter 8, we present the extension to the case of arbitrary dimensions. As with conventional domination, we have the local domination check (cf. Lemma 6.1).

Lemma 7.1. *Let $p = ((s,n_1),\ldots,(n_i,v),\ldots,(n_j,t)) \in \mathcal{S}_L(s,t)$ be a linear skyline path, then any subpath $q = ((s,n_1),\ldots,(n_i,v))$ is linearly non-dominated in $\mathcal{S}_L(s,v)$.*

Proof. Since path p is linearly non-dominated, we know that $\exists\ w \in \mathbb{R}^d_{\geq 0}$ such that $w^T p$ is minimal in the linear skyline. Assume there is a subpath $q = ((s, n_1), \ldots, (n_i, v))$ of p which is linearly dominated in $\mathcal{S}_L(s, v)$. The argumentation is analogous to Lemma 6.1. If there is a path q' such that $w^T q' < w^T q$, then by q with q', we obtain a full path p' such that $w^T p' < w^T p$. This contradicts the assumption. □

As mentioned in Section 6.1, skyline algorithms need to maintain all non-dominated paths ending at each node. Relying on the property of local domination, this is usually done by keeping a local skyline of non-dominated paths at each visited node n, where only these paths are extended. The same holds here, in the case of linear skylines in bicriteria networks. The key to our performance gain lies in efficiently managing the local linear skylines.

7.2 Managing Bicriteria Linear Skylines

Based on the local domination check, we may discard any linearly dominated paths and their extensions from further exploration. Thus, after extending a path p from s to v, we have to check for linear domination. If p is linearly non-dominated, it is stored in the local linear skyline of v, $\mathcal{S}_L(s, v)$. Subsequently, it has to be checked, if any other $p^i \in \mathcal{S}_L(s, v)$ might be linearly dominated by p, i.e., if there exists $p^j \in \mathcal{S}_L(s, v)$ such that $\{p, p^j\} \prec_{\text{lin}} p^i$.

These two steps are crucial because they are performed frequently and because the complexity increases with the number of elements in $\mathcal{S}_L(s, v)$. A naïve solution to this problem is to compare the new path p to any distinct pair $p^i, p^j \in \mathcal{S}_L(s, v)$. If non-dominated, it has to be checked whether p prunes any existing paths, i.e., if $\{p, p^j\} \prec_{\text{lin}} p^i$ holds for any pair $p^i, p^j \in \mathcal{S}_L(s, v)$. Both steps naïvely have quadratic complexity in the number of elements in $\mathcal{S}_L(s, v)$.

The complexity can be reduced from quadratic to linear by making use of the following observation: Linear skylines in two-dimensional cost spaces that are sorted ascending w.r.t. one criterion are sorted descending w.r.t. the other criterion. Following the notation for two-dimensional vector spaces, we denote the first criterion by subscript x and the second criterion by subscript y. Hence, ordering a linear skyline $\mathcal{S}_L(s, v) = \{p^1, \ldots p^K\}$ such that $\forall\ i < j$ holds $p^i_x < p^j_x$, then $p^i_y > p^j_y$. This follows directly from the definition of conventional domination. Thus, in the following, by *ordered linear skyline* we mean a tuple representing the linear skyline paths ordered ascending w.r.t. the first criterion (w.l.o.g.).

For a given path p in a linear skyline $\mathcal{S}_L(s, v)$, we call those elements $p^k, p^{k+1} \in \mathcal{S}_L(s, v)$ neighbors which are closest to p w.r.t. the first cost criterion. We refer to p^k with $p^k_x \leq p_x$ as left neighbor, and p^{k+1} with $p^{k+1}_x > p_x$ as right neighbor. At least one of both neighbors must exist, unless $\mathcal{S}_L(s, v) = \emptyset$. In order to check for linear domination, it suffices to compare a path to its neighbors. This property is proved in the following theorem.

Theorem 7.1. *Given a path $q = ((s, u), \ldots, (w, v))$ and the ordered local linear skyline $\mathcal{S}_L(s, v) = (p^1, .., p^K)$ at node v, then the following statements hold:*

(i) If $q_x < p^1_x$, then q is linearly non-dominated in $\mathcal{S}_L(s, v)$.

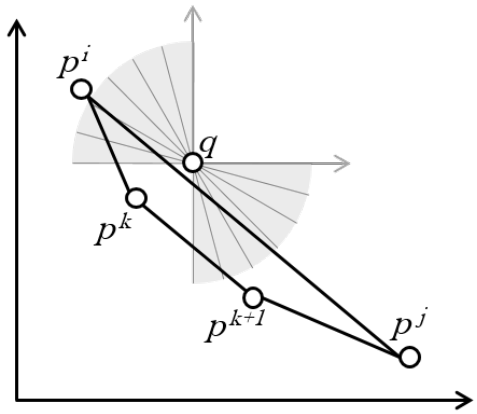


Figure 7.2: Exemplary visualization of vectors p^i, p^j dominating q. If left and right neighbors p^k and p^{k+1} exist, domination also holds for them. This is evident when translating by $-q$. Any non-negative weight vector creates two half-spaces, at least one of the neighbors is on the negative side.

(*ii*) *If* $q_y < p_y^K$*, then* q *is linearly non-dominated in* $\mathcal{S}_L(s, v)$*.*

(*iii*) *If there exist left and right neighbors of* q*, i.e.,* $p_x^k \leq q_x < p_x^{k+1}$*, and there exist* $p^i, p^j \in \mathcal{S}_L(s, v)$ *with* $\{p^i, p^j\} \prec_{lin} p$*, then* $\{p^k, p^{k+1}\} \prec_{lin} p$ *holds as well.*

Proof. We begin with the cases where p is a border vector in the cost space.
(*i*): If $q_x < p_x^1$, where p^1 is the skyline element with the smallest x-value, clearly

$$\begin{pmatrix}1 & 0\end{pmatrix}^T q = q_x < \min \begin{pmatrix}1 & 0\end{pmatrix}^T p^i$$

Hence, q is linearly non-dominated.
(*ii*): Conversely,

$$\begin{pmatrix}0 & \omega\end{pmatrix}^T q \underset{\omega \to \infty}{=} q_y < \min \begin{pmatrix}0 & \omega\end{pmatrix}^T p^i$$

Again, q is linearly non-dominated.
(*iii*): The situation is exemplified in Figure 7.2. The property is evident when translating q into the origin. For any (non-negative) weight vector w, $w^T q = 0$. $w^T p$ is the directed distance to a hyperplane through the origin with normal vector w. Any such weight vector divides the space into two half-spaces. Geometrically, it is obvious that for any such w at least one of the vectors p^k, p^{k+1} (just as for p^i, p^j) is in the negative half-space, i.e.,

$$\{p^k, p^{k+1}\} \cap \{x \in \mathbb{R}^2 : w^T x < 0\} \neq \emptyset$$

This proves the theorem. □

Practically, this observation allows us to reduce the amount of domination checks to a single check for each insertion. Furthermore, determining the neighbors of a path p

in $\mathcal{S}_L(s,v)$ can be performed using a binary search w.r.t. the first criterion. Thus, the complexity of checking whether p has to be inserted into $\mathcal{S}_L(s,v)$ is $\mathcal{O}(\log|\mathcal{S}_L(s,v)|)$.

Upon inserting a path, we have to check whether the insertion leads to the deletion of other elements in $\mathcal{S}_L(s,v)$. The general idea of how to efficiently determine linearly dominated paths is the following: All dominated elements have to be either conventionally dominated by the new path p or linearly dominated by p and some other linear skyline element. To find all linearly dominated paths efficiently (rather than testing all pairs), we propose to start two linear searches. The first search beginning with the left neighbor of p, p^k, and the second search beginning with the right neighbor, p^{k+1}. Using Theorem 7.1, we know a path is dominated, if it is either conventionally dominated or dominated by its neighbors. Thus, in addition to the conventional domination check, it suffices to check whether p^k is dominated by its left neighbor p^{k-1} and p when searching the left side. Conversely, it suffices to check for domination by p^{k+2} and p when searching the right side. Note that it is possible for the left neighbor p^{k-1} to conventionally dominate p.

If p^k indeed is dominated, there might be more dominated objects in the same search direction and the process has to be repeated. On the other hand, if p^k is not linearly dominated, we can stop the search in this direction. Again, this holds conversely for the right side.

Lemma 7.2. *Let $\mathcal{S}_L(s,v) = (p^1,..,p^K)$ be an ordered linear skyline and p be a non-dominated path which is not part of the skyline yet. If the left neighbor of p, p^k, is not linearly dominated by p and p^{k-1}, then no skyline element further left is linearly dominated, i.e., $\nexists\ i \in \{1,..,k-1\} : \{p,p^{i-1}\} \prec_{lin} p^i$. Conversely, if the right neighbor of p, p^{k+1}, is not linearly dominated by p and p^{k+2}, then no skyline element further right is linearly dominated, i.e., $\nexists\ i \in \{k+1,..,K\} : \{p,p^{i+1}\} \prec_{lin} p^i$.*

Proof. Suppose,$\{p^{k-1},p\} \not\prec_{\text{lin}} p^k$ but $\{p^{i-1},p\} \prec_{\text{lin}} p^i$ for some $i \leq k-1$. From Theorem 7.1 we know that if p^i is linearly dominated, it is linearly dominated by its neighbors p^{i-1} and p^{i+1}. This is a contradiction to the assumption that $p^i \in \mathcal{S}_L(s,v)$. Hence, $\{p^{i-1},p\} \not\prec_{\text{lin}} p^i$. The argument holds analogously for the right skyline side of p. □

Thus, when checking if a new path p dominates elements of $\mathcal{S}_L(s,v)$, we have to inspect the neighbors to both sides of p. If a neighbor is not linearly dominated, we may terminate the search in this direction. Updating the linear skyline $\mathcal{S}_L(s,v)$ has a linear worst case complexity $|\mathcal{S}_L(s,v)|$, which is the case if all former elements of $\mathcal{S}_L(s,v)$ have to be deleted. Let us note that the results of the dominance checks are negative in the majority of cases. Typically, it is only necessary to perform the domination test which has a logarithmic time complexity.

7.3 Computing Bicriteria Linear Skylines

After describing the efficient management of linear skylines, we will now specify our complete algorithm for computing linear skylines in bicriteria networks. In general, our algorithm starts constructing linearly non-dominated paths at the start node s by successively

selecting nodes and expanding all linearly non-dominated paths discovered so far. Furthermore, we mark paths which have been already expanded to avoid duplicate expansions.

Algorithm 3 Pseudocode of BLSRC

in : bicriteria network, start s, target t
out: linear skyline $\mathcal{S}_L(s,t)$

```
compute query-dependent lower bounds minimal costs to target t
initialize queue Q of nodes ordered w.r.t. first criterion (asc.)
add start node s to Q
while Q not empty do
    remove first element u of Q
    foreach path p ∈ S_L(s,u) do
        if p has not been processed then
            // global domination check using forward estimation
            if p + u^min not linearly dominated in S_L(s,t) then
                foreach outgoing edge (u,v) of u do
                    q ← extension of p by (u,v)
                    find neighbors p^k, p^{k+1} of q in S_L(s,v) (binary search)
                    if q is not linearly dominated by p^k and p^{k+1} then
                        // prune elements left of q
                        set i ← k
                        while i > 2 ∧ {p^{i-1}, q} ≺_lin p^i do
                            delete p^i from S_L(s,m), decrement i
                        end
                        // prune elements right of q
                        set i ← k + 1
                        while i < (|S_L(s,m)| − 1) ∧ {q, p^{i+1}} ≺_lin p^i do
                            delete p^i in S_L(s,m), increment i
                        end
                        reinsert v into Q
                    end
                end
            end
            flag p as processed
        end
    end
end
return S_L(s,t)
```

Our method employs two pruning rules to exclude paths which cannot be extended into a linearly non-dominated result path. The first rule is the local domination check. A path ending at node v has to be linearly non-dominated in $\mathcal{S}_L(s,v)$. To enforce this rule, we

Table 7.1: Overview of absolute runtime averages. In case of timeouts (>60s) the rounded number of timeouts is denoted next to the † symbol.

cost criteria	time & distance		time & crossings		rand.$_1$ & rand.$_2$	
graph	G_{munich}	G_{bavaria}	G_{munich}	G_{bavaria}	$G_{250\times250}$	$G_{50\times50\times50}$
# of tasks	2450	90	2450	90	4	8
BLRSC	0.2s	4s	0.36s	4.83s	17s	22.49s
Multi-A^*	0.21s	5.13s	0.37s	>8.47s †3	30.4s	>55.4s †4
ARSC	0.25s	>21.06s †18	0.4s	>11.12s †6	>60s †4	>60s †8
Multi-BD	0.61s	>33.05s †32	0.41s	>17.54s †13	30.85s	41.3s

maintain a local linear skyline at all visited nodes v and delete all paths that are linearly dominated. The second pruning rule is the global domination check, i.e., whether a path p might be extended into a linearly non-dominated full path between s and t. If the cost vector of a path $p = ((s, n_1), ..(n_i, v))$ is already linearly dominated in $\mathcal{S}_L(s, t)$, the path may be discarded from further consideration. This pruning method can be considerably improved by employing lower bound cost vectors. As detailed in Section 6.1, we differentiate between query-independent lower bounds (e.g., [80]) and query-dependent lower bounds (e.g., Multi-Dijkstra [148] and ParetoPrep). While query-independent lower bounds are of minor quality but may be precomputed during an offline phase, query-dependent lower bounds are optimal and computed at query time. We propose to employ ParetoPrep for the computation of lower bounds, as it outperforms other approaches.

Another approach to computing optimal lower bounds for bicriteria networks is [93] which is – casually speaking – a variation of Multi-Dijkstra making use of the special properties in bicriteria networks. At query time, two reverse Dijkstra searches from target to start are performed, therefore we refer to this method as Double-Dijkstra. From these searches, two cost-optimal paths are obtained. The cost-optimal path w.r.t. the first cost criterion p^1 serves as an upper bound w.r.t. the second criterion, i.e., p^1_y is the maximum y-value of any (linear) skyline path due to the ordering property of two-dimensional skylines. Conversely, the cost-optimal path w.r.t. the second criterion p^2 serves as an upper bound w.r.t. the first criterion, p^2_x is the maximum x-value of any (linear) skyline path. Upon finding the two cost-optimal paths, the two searches are continued until the respective bounds p^2_x, p^1_y are exceeded. By using the sorting property of bicriteria skylines, [93] need not visit all nodes of the network as Multi-Dijkstra would. In conclusion, any visited node v may be labeled with a cost vector $(v^{\min}_x, v^{\min}_y)$ holding the costs of the optimal paths from v to the target. ParetoPrep, [93] and Multi-Dijkstra generate the same optimal lower bounds. Although ParetoPrep is more efficient (only one traversal instead of two), for reasons of comparability, we implemented the well-established method [93] for the experiments.

Algorithm 3 describes our proposed method BLRSC in pseudocode. Double-Dijkstra ([93]) is employed for the computation of lower bound cost vectors. Subsequently, the actual graph traversal is initialized. We maintain a priority queue of nodes in ascending order w.r.t. the first cost criterion. Upon initialization, the start node s is added to the queue. For each visited node u, a local linear skyline is stored which is managed as an

Table 7.2: Overview of runtime averages relative to runtime averages of our main contribution BLRSC. In case of timeouts (>60s) the rounded number of timeouts is denoted next to the † symbol.

cost criteria	time & distance		time & crossings		rand.$_1$ & rand.$_2$	
graph	G_{munich}	G_{bavaria}	G_{munich}	G_{bavaria}	$G_{250\times250}$	$G_{50\times50\times50}$
# of tasks	2450	90	2450	90	4	8
BLRSC	1	1	1	1	1	1
Multi-A^*	1.05	1.28	1.02	>1.75 †3	1.78	>2.46 †4
ARSC	1.22	>5.25 †18	1.09	>2.29 †6	>3.52 †4	>2.66 †8
Multi-BD	2.98	>8.24 †32	1.12	>3.62 †13	1.81	1.83

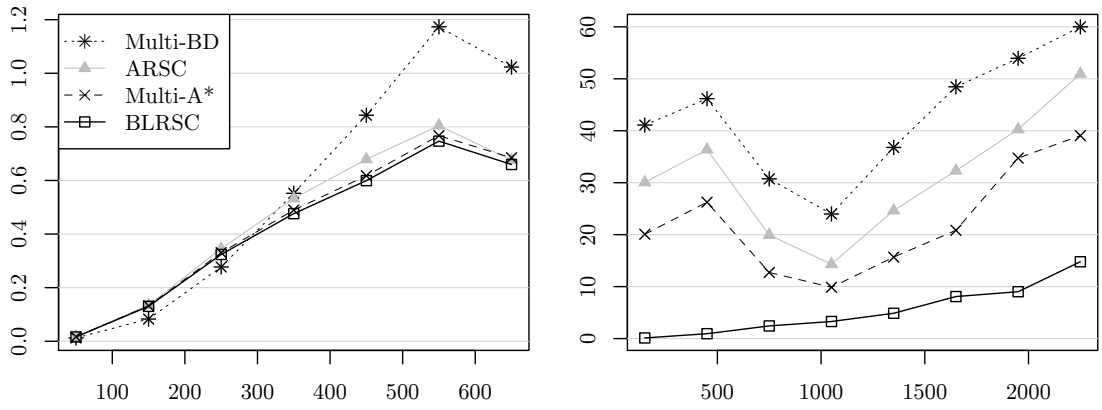
ordered list. The priority of a node corresponds to the smallest x-value of any linearly non-dominated path which has not yet been processed, i.e., extended. Note that sorting the priority queue according to the y-value or a weighted sum of both values is possible but in general yields no speed-up. Already processed paths are kept in the skyline, as they may dominate other paths but are flagged accordingly to prevent multiple extensions. In the main loop, the algorithm removes the top node u. Each unprocessed path $p \in \mathcal{S}_L(s,u)$ is checked for global domination, i.e., if $p+v^{\min}$ is dominated in $\mathcal{S}_L(s,t)$. If p is dominated, it is flagged as processed. Otherwise, p is extended by all outgoing edges of node u. Each new path q ending at node v is checked for local linear domination in the skyline $\mathcal{S}_L(s,v)$. If q is linearly non-dominated in $\mathcal{S}_L(s,v)$, it has to be checked whether q dominates any paths in $\mathcal{S}_L(s,v)$. All linearly dominated results are removed from $\mathcal{S}_L(s,v)$ by successively searching left and right neighbors of q until a linearly non-dominated path is found in either direction. Furthermore, if previously not contained, node v is added to the queue. If contained, the priority of v is updated if possible. The algorithm terminates, if the queue is empty which means that there is no path left which may be extended into a result path.

This concludes the section on linear skyline computation in bicriteria networks. Before linear path skylines in multicriteria networks are explored, we present the results of our experiments within bicriteria networks.

7.4 Evaluation

All experiments are performed on a dedicated machine with an 3.0 Ghz Intel Xeon 5160 processor and 32 GB RAM. The algorithms were implemented in Java 1.7 and do not make use of multiple cores. Each task is consecutively solved by all compared algorithms and the execution order is randomized for each task. If an algorithm is not able to solve a task in less than 60 seconds the computation is aborted and it is counted as a timeout. Three separate runs of all tasks are performed and the average of these three runs is used. This experimental setup is intended to ensure similar evaluation conditions, eliminate possible evaluation order effects and smooth out runtime discrepancies.

There are four different graphs on which routing tasks are performed:



(a) G_{munich} with time & crossings (b) G_{bavaria} with time & crossings

Figure 7.3: Values on the x-axis are numbers of hops between end points and values on the y-axis are average runtimes in seconds.

1. G_{munich} is a OpenStreet Map (OSM) road network of the area around Munich which covers 6 992 km^2, including the neighboring city Augsburg. The derived graph consists of 782 030 nodes and 1 595 261 edges. The 2450 routing tasks on this graph are between 50 city districts and municipalities in and around Munich.

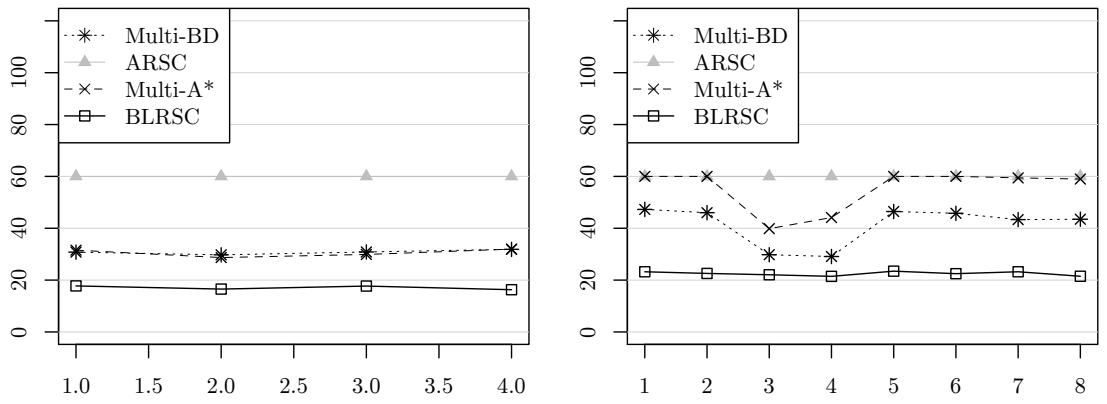
2. G_{bavaria} is a OSM road network of Bavaria which covers 70 549 km^2. The derived graph consists of 4 044 556 nodes and 8 298 017 edges. The 90 routing tasks on this graph are between 10 major Bavarian cities.

3. $G_{250\times250}$ is a two-dimensional grid network with 62 500 nodes and 249 000 edges. The four routing tasks on this graph are between opposite corners of the grid.

4. $G_{50\times50\times50}$ is a three-dimensional grid network with 125 000 nodes and 735 000 links. The eight routing tasks on this graph are between opposite corners of the lattice.

On G_{munich} and G_{bavaria}, the criteria time & distance and time & crossings are used for the routing tasks. On $G_{250\times250}$ and $G_{50\times50\times50}$, the criteria rand.$_1$ & rand.$_2$ are used which have independent pseudo-random cost values.

We compare our proposed method BLRSC to three other algorithms: The first is ARSC [80] which is an algorithm developed to compute conventional path skylines. It has been modified to use the same precomputation step as proposed in [93] to achieve better comparability to BLRSC. Multi-BD represents the state-of-the-art method for computing supported solutions as proposed in [116]. For Multi-BD, the single-criterion cost-optimal paths are computed using bidirectional Dijkstra searches. However, the lower bounds as proposed in [93] (or in Section 6) may be combined with the approach in [116]. This allows to run $A^\star$-searches instead of bidirectional Dijkstra searches. We will refer to this modification as Multi-A^*. Comparing to Multi-A^*, we may evaluate how much of the performance gain is caused by the optimal lower bounds.



(a) $G_{250\times250}$ with rand.$_1$ & rand.$_2$ (b) $G_{50\times50\times50}$ with rand.$_1$ & rand.$_2$

Figure 7.4: Values on the x-axis are ids of the tasks and values on the y-axis are runtimes of three runs in seconds. Any run which took longer than 60 seconds was counted as timeout and is marked with "60s".

As Table 7.1 shows, BLRSC has the lowest runtime average in all experimental conditions, and it is the only algorithm in the experiments which solves all tasks in less than 60 seconds. This is the case throughout different task difficulties, i.e. for varying distances, measured in the number of hops between start and target (see Figure 7.3). The only exception occurs in Figure 7.4(a), where Multi-BD is faster than BLRSC for less than 300 hops. BLRSC is also clearly faster for grid networks, as depicted in Figure 7.4. BLRSC is on average significantly faster than Multi-BD. As can be seen in Table 7.2, on G_{munich} with criteria time & distance it is about three times faster, and on G_{bavaria} it is more than three to eight times faster.

ARSC performs very poorly on $G_{250\times250}$ and $G_{50\times50\times50}$, where it cannot solve a single task in less than 60 seconds which is also the worst performance out of all tested algorithms. ARSC is more than five times slower than BLRSC on G_{bavaria} with time & distance. Multi-A^* also performs poorly on $G_{250\times250}$ and $G_{50\times50\times50}$, where it is even slower than Multi-BD and much slower than BLRSC. It is at least about twice as slow as BLRSC on G_{bavaria} with time & crossings. The experiments show that BLRSC clearly outperforms all other tested approaches and that its performance gain is not only caused by the use of a forward estimation which is also employed by ARSC and $A^\star$.

Chapter 8

Linear Path Skylines in Multicriteria Networks

Computing linear path skylines in multicriteria networks is a complex task. In multicriteria networks, the sorting property ($p_x^i < p_x^j \Rightarrow p_y^i > p_y^j$ for two linear skyline paths p^i, p^j) does not hold. There exists no notion of neighboring paths. Consequently, there is no intuition on how to maintain local and global skylines. Also, it is not straightforward which paths may dominate other paths (as Lemma 7.1 does not hold). Furthermore, in a d-dimensional multicriteria network, a path is possibly dominated by d other paths. In a skyline with n elements, naïvely $\binom{n}{d}$ domination checks would have to be conducted. Finally, forward estimations are more costly in multicriteria networks than in bicriteria networks.

Given these obstructions, BLRSC, the algorithm for bicriteria networks, cannot be extended to the arbitrarily dimensional case. We therefore propose a different approach, related to the computation of convex hulls. Additionally, we introduce an extension of the linear skyline which further restricts the result set. Before we may go into detail, we first have to prove some properties and introduce new definitions.

8.1 Preliminaries

In the definition of the linear skyline 7.1 it suffices to require $w \in \mathbb{R}^d_{>0}$ instead of $w \in \mathbb{R}^d_{\geq 0}$. Since this property is of importance to later results, it is proved in the following lemma.

Lemma 8.1. *Let $\mathcal{P}$ denote a finite set of distinct points in a d-dimensional vector space. Let $0 \neq w \in \mathbb{R}^d_{\geq 0}$ be an arbitrary weight vector, and let $x \in \mathcal{P}$ such that $w^T x < w^T y$ for any $x \neq y \in \mathcal{P}$. Then there exists $\hat{w} \in \mathbb{R}^d_{>0}$ for which holds: $\hat{w}^T x < \hat{w}^T y$ for any $x \neq y \in \mathcal{P}$.*

Proof. Let $I \subseteq \{1, \ldots, d\}$ be such that $w_i = 0$ for all $i \in I$ and $w_j > 0$ for all $j \notin I$. Let $k = |I|$ denote the number of dimensions in which w is zero. If $k = 0$, then $w \in \mathbb{R}^d_{>0}$ is already the case. Therefore, we assume $k > 0$, and by $w \neq 0$, we have $k < d$. Furthermore, let $M := \max\{y_i \mid y \in \mathcal{P}, 1 \leq i \leq d\}$ denote the maximal component of all points in $\mathcal{P}$. By assumption, we have $w^T x < w^T y$ for all $x \neq y \in \mathcal{P}$. Let $m := \min\{w^T y - w^T x \mid$

$x \neq y \in \mathcal{P}\} > 0$. For w with k components being zero, we now construct $\hat{w} \in \mathbb{R}^d_{>0}$ for which the required condition holds. Let $w'_i := w_i$ for all $i \notin I$, and let $w'_j := w_j + \epsilon$ for all $j \in I$ where $\epsilon := m/(2kM) > 0$. Hence, $\hat{w} \in \mathbb{R}^d_{>0}$. Therefore, it remains to show that indeed $\hat{w}^T x < \hat{w}^T y$ for any $x \neq y \in \mathcal{P}$. To prove this, we use the following inequalities: $\hat{w}^T x \leq w^T x + \epsilon kM$ and $\hat{w}^T y > w^T y$.

$$\begin{aligned} \hat{w}^T y - \hat{w}^T x &> w^T y - \hat{w}^T x \\ &\geq w^T y - w^T x - \epsilon kM \\ &\geq m - \frac{m}{2kM} kM = \frac{m}{2} > 0 \end{aligned}$$

This proves the lemma. □

From the above lemma, we may derive the following equivalence in the definition of the linear skyline.

Remark 8.1. *In Definition 7.1, requiring the weight vectors w in condition (i) and (ii) to be strictly positive, i.e. $w \in \mathbb{R}^d_{>0}$, yields the same result set.*

Proof. Let $\mathcal{S}_L^{\geq}$ denote the linear skyline as defined by Definition 7.1, and let $\mathcal{S}_L^{>}$ denote the set of points which are described by the definition if $w \in \mathbb{R}^d_{>0}$ is required. Furthermore, let $\mathcal{P}$ denote set of d-dimensional cost vectors. We show: $\mathcal{S}_L^{\geq} = \mathcal{S}_L^{>}$. Obviously, $\mathcal{S}_L^{>} \subseteq \mathcal{S}_L^{\geq}$, because for $x \in \mathcal{S}_L^{>}$, there exists $0 \neq w \in \mathbb{R}^d_{>0} \subset \mathbb{R}^d_{\geq 0}$ such that $w^T x < w^T y$ for all $x \neq y \in \mathcal{P}$. Now, let $x \in \mathcal{S}_L^{\geq}$. By definition, there exists $0 \neq w \in \mathbb{R}^d_{\geq 0}$ such that $w^T x < w^T y$ for all $x \neq y \in \mathcal{P}$. Lemma 8.1 states that it suffices to require $0 \neq w \in \mathbb{R}^d_{>0}$. Therefore, $x \in \mathcal{S}_L^{>}$ which proves the statement. □

By introducing the notion of the linear skyline, we aim to accelerate computation while limiting the result set. $\mathcal{S}_L$ generally holds less elements than $\mathcal{S}_C$, however, we may trim the result set even further. For this purpose we introduce the notion of an ε-linear skyline.

Definition 8.1. *Let s and t be nodes in a multicriteria network, and let $\mathcal{P}$ be the set of paths between s and t. Furthermore, let $\varepsilon > 0$. A subset of paths $\mathcal{S}_\varepsilon \subseteq \mathcal{P}$ is the* ε-linear path skyline *(w.r.t. s and t), iff the following two conditions hold:*

(i) Completeness:

$$\forall\ 0 \neq w \in \mathbb{R}^d_{\geq 0}\ \exists\ x \in \mathcal{S}_\varepsilon \text{ such that: } w^T x \leq \frac{1}{1+\varepsilon} \min w^T y\ \forall\ y \in \mathcal{P}$$

(ii) Minimality:

$$\forall\ x \in \mathcal{S}_\varepsilon\ \exists\ 0 \neq w \in \mathbb{R}^d_{\geq 0} \text{ such that: } w^T x < w^T y\ \forall\ x \neq y \in \mathcal{P}$$

By the tightened condition (i), we get $\mathcal{S}_\varepsilon \subseteq \mathcal{S}_L$. Therefore, we have the following chain of inclusion: $\mathcal{S}_\varepsilon \subseteq \mathcal{S}_L \subseteq \mathcal{S}_C$. Note that equality in the chain is improbable and only occurs for very small skyline sizes or pathological examples. According to these notions of a skyline, we have three different queries. Given a multicriteria network, a start and a target node therein, we may compute the *(conventional) path skyline*, the *linear path skyline* or the *ε-linear path skyline* for an additional parameter $\varepsilon > 0$.

8.2 Convex Hulls

In the following, convex hulls are defined which are inherently similar to linear skylines. This similarity will in turn be emphasized in a series of statements. As a foundation for what will follow, let us formalize the notion of convex combinations. Convex combinations are weighted averages of point sets. Let $\{\alpha^1, \ldots, \alpha^k\}$ be non-negative weights such that $\alpha^1 + \ldots + \alpha^k = 1$. Any linear combination $\alpha^1 p^1 + \ldots + \alpha^k p^k$ is a convex combinations of the set of points $\{p^1, \ldots, p^k\}$. We now introduce convex hulls in a similar way as linear skylines are defined.

Definition 8.2. *Let $\mathcal{P} \subset \mathbb{R}^d_{\geq 0}$ be a finite set of points, e.g., cost vectors of paths between the same pair of nodes in a multicriteria network. The convex hull of $\mathcal{P}$ is the set of all convex combinations of the points in $\mathcal{P}$, denoted by $Conv(\mathcal{P})$. Each point $p \in \mathcal{P}$ that is not in the convex hull of all other points, i.e., $p \notin Conv(\mathcal{P} \setminus \{x\})$, is called a vertex of $Conv(\mathcal{P})$. We denote the set of vertices of $Conv(\mathcal{P})$ by $\mathcal{V}(\mathcal{P})$. $\mathcal{V}(\mathcal{P})$ is defined by the following two conditions:*

(i) Completeness:

$$\forall\ 0 \neq w \in \mathbb{R}^d\ \exists\ x \in \mathcal{V}(\mathcal{P})\ \textit{such that:}\ w^T x = \min_{y \in \mathcal{P}} w^T y$$

(ii) Minimality:

$$\forall\ x \in \mathcal{V}(\mathcal{P})\ \exists\ 0 \neq w \in \mathbb{R}^d\ \textit{such that:}\ w^T x < w^T y\ \forall\ x \neq y \in \mathcal{P}$$

The notion of a convex hull is illustrated in Figure 8.1(a). Note that the concept of linear skylines only differs from that of convex hulls in the requirement of the weight vectors being non-negative (or strictly positive by Lemma 8.1). This is also evident in Figure 8.1(b). Another important geometric notion is needed, the notion of facets of the convex hull.

Definition 8.3. *Let $\mathcal{P} \subset \mathbb{R}^d_{\geq 0}$ be a finite set of points. Let $\mathcal{V}(\mathcal{P})$ be the set of convex hull vertices of $\mathcal{P}$.*

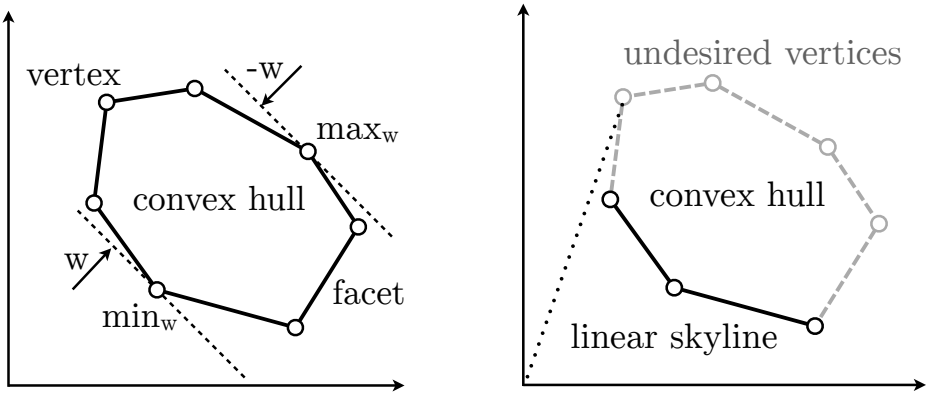
(i) All vertices $\{v_1, \ldots v_k\}$ of a facet lie on the same hyperplane with the normal vector $0 \neq w \in \mathbb{R}^d$, such that $w^T v_1 = \ldots = w^T v_k$.

(ii) $w^T v_1 = \min_{x \in \mathcal{V}(\mathcal{P})} w^T x$

As we rely on a rather untypical definition of convex hulls, we shall now prove its equivalence to the conventional definition. Before the actual theorem, we first prove the following lemma.

Lemma 8.2. *Let $0 \neq x \in \mathbb{R}^d$. The following two statements are equivalent:*

(i) $\exists\ 0 \neq w \in \mathbb{R}^d$ such that $w^T x < w^T y$ for all $x \neq y \in \mathcal{P}$



(a) Illustration of the definition of convex hulls.

(b) Illustration of the part of the convex hull which forms the linear skyline.

Figure 8.1: Linear Skylines and Convex Hulls

(ii) x is no convex combination of $\mathcal{P} \setminus \{x\}$

Proof. $(i) \Rightarrow (ii)$: Suppose, x was a convex combination of $\mathcal{P} \setminus \{x\}$, i.e., there exist non-negative $\alpha^1, \ldots, \alpha^k$ with $\sum_{i=1}^{k} \alpha^i = 1$ such that $x = \sum_{i=1}^{k} \alpha^i p^i$. Let $0 \neq w \in \mathbb{R}^d$ as in (i).

$$\begin{aligned} w^T x &= w^T \big(\sum_{i=1}^{k} \alpha^i p^i\big) \\ &= \sum_{i=1}^{k} \alpha^i w^T p^i \\ &\geq \sum_{i=1}^{k} \alpha^i \min_j w^T p^j \\ &= \big(\sum_{i=1}^{k} \alpha^i\big) \min_j w^T p^j = \min_j w^T p^j \end{aligned}$$

Which is a contradiction to (i), i.e., $w^T x < w^T y$ for all $x \neq y \in \mathcal{P} \setminus \{x\}$. Therefore, x must be a convex combination.
$(ii) \Rightarrow (i)$: The value of $w^T x$ is the distance from a hyperplane through the origin with normal vector w. If x is no convex combination of $\mathcal{P} \setminus \{x\}$, then it is, by definition, outside of the convex hull $Conv(\mathcal{P})$. For any point x outside of $Conv(\mathcal{P})$ it is possible to find $0 \neq w \in \mathbb{R}^d$ such that $w^T x < w^T y$ for any $y \in Conv(\mathcal{P})$. This can be seen by translating the vector space with $-x$, i.e., shifting x into the origin. Since $x - x = 0 \notin Conv(\mathcal{P})$, there must exist $0 \neq w \in \mathbb{R}^d$ such that for all $y \in Conv(\mathcal{P})$ holds $w^T y > 0 = w^T x$, which proves the statement. □

Theorem 8.1. *Definition 8.2 and the conventional definition of the convex hull vertices coincide, i.e., $\mathcal{V}(\mathcal{P})$ is the minimal set of points such that all points in $\mathcal{P}$ are convex combinations of $\mathcal{V}(\mathcal{P})$ (or equivalently, the intersection of all sets of points with this property).*

Proof. We first show completeness (all points in $\mathcal{P}$ are convex combinations of $\mathcal{V}(\mathcal{P})$), then we show minimality ($\mathcal{V}(\mathcal{P})$ is the smallest set which fulfills this property).
Let $x \in \mathcal{P}$. If $x \in \mathcal{V}(\mathcal{P}) \subseteq \mathcal{P}$, the statement holds trivially. Let $x \in \mathcal{P} \setminus \mathcal{V}(\mathcal{P})$. Suppose, x was no convex combination of $\mathcal{V}(\mathcal{P})$. By Lemma 8.2 there exists $0 \neq w \in \mathbb{R}^d$ such that $w^T x < w^T y$ for all $y \in \mathcal{V}(\mathcal{P})$. Hence, by Definition 8.2, $x \in \mathcal{V}(\mathcal{P})$. This contradicts the assumption, therefore x must be a convex combination of $\mathcal{V}(\mathcal{P})$.
Suppose there was some set $V \subsetneq \mathcal{V}(\mathcal{P}) \subseteq \mathcal{P}$ such that all points in $\mathcal{P}$ are convex combinations of V. Let $x \in \mathcal{V}(\mathcal{P}) \setminus V$. By Definition 8.2 (for $\mathcal{V}(\mathcal{P})$), there exists $0 \neq w \in \mathbb{R}^d$ such that $w^T x < w^T y$ for all $x \neq y \in \mathcal{P}$. Particularly, this holds for all $y \in V \subseteq \mathcal{P}$. By Lemma 8.2, this implies that x is no convex combination of V. This contradicts the assumption that all points in $\mathcal{P}$ are convex combinations of V. Therefore, $\mathcal{V}(\mathcal{P})$ is the minimal set fulfilling this property. □

8.3 Linear Skyline Convex Hulls

As depicted in Figure 8.1(b), the linear skyline is a subset of the convex hull. One may think that the linear skyline is the subset of the convex hull that is visible from the origin, but the dotted line in the figure disproves this notion. Geometrically, the linear skyline contains all points that minimize the distance to some hyperplane through the origin, but convex hull vertices can also maximize that distance. In order to eliminate the undesired vertices, the convex hull can be computed in the augmented cost vector space $\mathcal{P} \cup \mathrm{INF}$ instead of $\mathcal{P}$. We now define what we refer to as infinity points.

Definition 8.4. *The set of infinity points INF is defined as follows.*

$$\{\lim_{\omega\to\infty}(\omega, 0, \ldots, 0]^T, \ldots, \lim_{\omega\to\infty}(0, 0, \ldots, \omega)^T\}$$

The i-th criterion's infinity point $INF^i \in INF$ lies on the i-th criterion's axis at infinity, i.e.,

$$INF^i_j = \lim_{\omega\to\infty} \begin{cases} \omega, & j = i \\ 0, & j \neq i \end{cases}$$

The convex hull of $\mathcal{P} \cup \mathrm{INF}$ has two facets which have exclusively infinity points as vertices, the front-infinity-facet and the back-infinity-facet. The front-infinity facet's normal vector has only positive components and the back-infinity-facet has only negative components. The back-infinity-facet is part of any convex hull in $\mathcal{P} \cup \mathrm{INF}$.

We call the convex hull of the augmented cost vector space $\mathcal{P} \cup \mathrm{INF}$ the *linear skyline convex hull of* $\mathcal{P}$. We prove that the vertices of the linear skyline convex hull are simply the elements of the linear skyline in $\mathcal{P}$ plus the infinity points.

Lemma 8.3. *Let $\mathcal{P} \subset \mathbb{R}^d_{\geq 0}$ be a finite set of points.*

$$\mathcal{S}_L(\mathcal{P}) = \mathcal{V}(\mathcal{P} \cup \mathit{INF}) \setminus \mathit{INF}$$

Proof. "$\subseteq$": This inclusion follows directly from $\mathbb{R}^d_{\geq 0} \subset \mathbb{R}^d$.

"$\supseteq$": We show two properties from which we may deduce the inclusion. (i): For any $0 \neq w \in \mathbb{R}^d \setminus \mathbb{R}^d_{\geq 0}$, an infinity point constitutes the minimal dot product. (ii): For any $0 \neq w \in \mathbb{R}^d_{\geq 0}$, a point in $\mathcal{P} \setminus \mathrm{INF}$ constitutes the minimal dot product. Hence, for positive weight vectors, we obtain the linear skyline points, and for weight vectors with at least one negative component, we obtain infinity points.
(i): Consider $0 \neq w \in \mathbb{R}^d \setminus \mathbb{R}^d_{\geq 0}$, i.e., a weight vector with at least one negative component $1 \leq i \leq d$. Obviously, $w^T \mathrm{INF}^i < w^T y$ for all $y \in \mathcal{P} \setminus \mathrm{INF}$. Therefore, the infinity points minimize dot products with non-positive weight vectors.
(ii): From Lemma 8.1 follows that if for $x \in \mathcal{P}$ holds $w^T x < w^T y$ for all $y \in \mathcal{P} \setminus \{x\}$ and some $w \in \mathbb{R}^d_{\geq 0}$, then there exists $w \in \mathbb{R}^d_{>0}$ such that $w^T x < w^T y$ for all $y \in \mathcal{P} \setminus \{x\}$. This means, it suffices to require $0 \neq w \in \mathbb{R}^d_{>0}$. But for any such weight vector, the dot product with an infinity point diverges. Therefore, the set of vectors with minimal dot products is the same for $\mathcal{P}$ and $\mathcal{P} \cup \mathrm{INF}$. □

8.4 Incremental Computation of Linear Skylines

In the previous section, it was shown that the linear skyline of $\mathcal{P}$ can be obtained by computing the convex hull in $\mathcal{P} \cup \mathrm{INF}$. Now, we show how this can be used to compute linear path skylines. In general, for two nodes in a networks, there are too many connecting paths to compute all. The complete linear skyline convex hull can be obtained by creating paths whose cost vectors are contained in $\mathcal{V}(\mathcal{P} \cup \mathrm{INF})$. The idea is to find new solutions in each iteration, determine the convex hull in $\mathcal{P} \cup \mathrm{INF}$ of the known solutions and search for a new solution outside of the convex hull by minimizing the dot product with the normal vectors of the hull's facets. In order to produce a new path whose cost vector minimizes the dot product, an $A^\star$-search is performed using the normal vector. The computation terminates when no more cost vectors outside of the convex hull can be found. Any solutions that are known during the computation are elements of the final result set and can therefore immediately be returned to the user.

The proposed algorithm to compute the linear skyline is outlined in Algorithm 4. An exemplary run of the algorithm for two cost criteria is shown in Figure 8.2 Note that angles and distances are depicted incorrectly, in order to display the infinity points. At first the algorithm begins with the front-infinity-facet and back-infinity-facet which are both depicted as lines connecting the infinity points. Open facets are depicted as dashed lines and closed facets as solid lines. The first search minimizes the normal vector of the front-infinity-facet and finds a new solution which leads to the removal of the front-infinity-facet and the addition of two new open facets. The second and third search minimize the normal vectors of these two open facets. The second search finds a previously discovered solution and the corresponding facet is closed. The third search finds a previously undiscovered

solution which leads to the removal of the facet and the addition of two new open facets. One of these new facets has the same normal vector as a previously processed facet and is immediately closed. The fourth search finds a previously undiscovered solution and the corresponding facet is closed. This concludes the computation. It can be seen that instead of computing the complete convex hull of all cost vectors in $\mathcal{P}$, it only computes the linear skyline of $\mathcal{P}$.

Initialization The algorithm starts with a convex hull of INF which only contains the front- and back-infinity-facet. The front-infinity-facet is added to the set of open facets and the back-infinity-facet is added to the set of closed facets. We propose to choose the normal vector of the front-infinity-facet as $[1, 1, \ldots, 1]^T$ but this is not essential for correctness of the algorithm.

The algorithm maintains a set of open facets, a set of closed facets and a set of linear skyline elements $P \subseteq \mathcal{S}_L(\mathcal{P})$. Facets are represented as a sorted integer array and vertices are represented as list indices. Infinity points can then be given negative vertex indices from $-d$ to -1 and are not part of the list, because they are not part of the linear skyline in $\mathcal{P}$.

Find solutions outside current Convex Hull In the second step, the algorithm searches for further solutions outside of the current convex hull. A facet's hyperplane divides the space in two halves. One half contains the origin, the other does not. For all hyperplanes, any cost vectors inside the convex hull lie in the half that does not contain the origin. To find solutions that are outside of the convex hull, it is necessary to find cost vectors that lie on the same side as the origin for some facet's hyperplane. If such a cost vector does not exist for a facet's hyperplane, the facet is a *closed facet*, i.e., $\min_{x \in V} w^T x = min_{y \in \mathcal{P}} w^T y$ where V is the set of vertices of the facet and w is its normal vector.

First, an open facet F with the vertices V and the normal vector $0 \neq w \in \mathbb{R}^d_{\geq 0}$ is selected. How normal vectors are determined is outlined in the third step of the algorithm. Second, the cost vector x such that $w^T x = \min_{x \in \mathcal{P}} w^T x$ and its associated solution is computed. Third, if $w^T x$ is smaller than the minimal dot product with the facet's vertices $\min_{x \in V} w^T x$, then x lies outside the convex hull. If x lies outside the convex hull, the facet F is simply removed from the set of open facets and x is added to the list of linear skyline elements. If x does not lie outside the convex hull, the facet F is moved from the set of open facets to the set of closed facets.

Open facets of convex hulls in $\mathcal{P} \cup \text{INF}$ always have normal vectors with non-negative components. By definition, vertices of facets have to lie on the same hyperplane and cost vectors in $\mathcal{P}$ always have larger dot products than cost vectors in INF for any normal vectors that contain negative components. Therefore only facets which consist solely of infinity points can have normal vectors $w \notin \mathbb{R}^d_{\geq 0}$ and the back-infinity-facet is trivially never an open facet. Minimizing the dot product with such vectors is therefore simply a search with scalarized cost objectives. For linear path skylines, scalarized searches to

Algorithm 4 LSCH (Linear Skyline Convex Hull)

```
begin
    initialize P = INF, open = ∅, closed = ∅
    add front-infinity-facet to open and back-infinity-facet to closed
    while open ≠ ∅ do
        remove an open facet F with normal vector 0 ≠ w ∈ ℝ^d_≥0 and vertices V ⊂ P from
        open
        find x such that w^T x = min_{y∈P} w^T y
        if w^T x < min_{y∈V} w^T y then
            add x to P
        end
        if w^T x ≥ min_{y∈V} w^T y then
            add F to closed
            if open = ∅ then
                return P \ INF
            end
        end
        determine facets of convex hull and their normal vectors, add new facets to open
    end
end
```

find cost-optimal paths between two query locations, can be performed by shortest path algorithms like Dijkstra, Bidirectional Dijkstra and $A^\star$-search. For a weight vector w and an edge's cost vector e the scalarized edge cost during shortest path search is simply $w^T e$. If a search was previously conducted with the same weight vector w, the search would yield the same result. The normal vectors of previous results are therefore stored to prevent redundant searches.

Update convex hull and determine normal vectors of facets Finding the convex hull facets of a set of vertices is a well-known problem called facet enumeration and can be solved by most existing convex hull algorithms for an arbitrary number of dimensions. Incremental approaches like the Beneath-And-Beyond approach have the advantage, that they do not need to recompute all facets and only determine new facets. If the algorithm or its implementation does not support infinitely large numbers, it can be approximated by a very large number.

Normal vectors of facets can be determined by solving a linear equation system $Ax = b$. The matrix A has the vertices of the facets as row vectors, the vector b is any vector that has the same positive value for all components and the vector x is the resulting normal vector. When infinity points are vertices of the facet, any components where infinity points have infinitely large components have to be zero for the normal vector. The infinity points and their infinity components therefore can be excluded from the linear equation system only a $d - k \times d - k$ linear equation system has to be solved if there are k infinity point

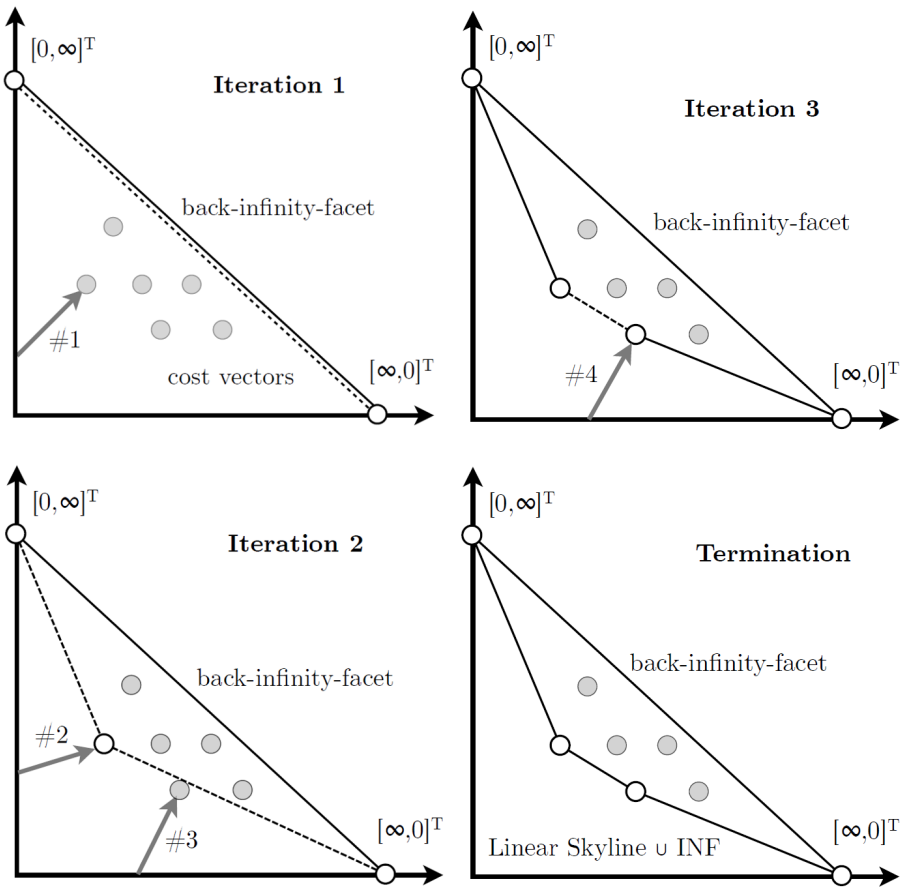


Figure 8.2: Exemplary linear skyline convex hull computation which obtains a linear skyline with two cost vectors in four scalarized searches. Coordinates are schematic, otherwise it would not be possible to display the infinity points.

vertices.

Termination If the set of open facets is not empty, the algorithm returns to the second step and tries to find cost vectors outside the new convex hull.

If the set of open facets is empty, the hyperplanes of the closed facets define the convex hull of the augmented space $\mathcal{P} \cup \text{INF}$ and the algorithm terminates. After termination, the algorithm's list of linear skyline solutions P is equal to the complete linear skyline $\mathcal{S}_L(\mathcal{P})$.

8.5 Linear Path Skyline Computation

In the previous section, it was shown how the linear path skyline computation can be reduced to scalarized searches, i.e., $A^\star$-searches according to the weighting of normal vectors. We will now describe the details about how these scalarized searches are performed.

Algorithm 5 Linear Path Skyline Computation

begin
compute shortest path costs from all nodes to the target for subsequent $A^\star$-searches
compute linear path skyline with LSCH using $A^\star$ for scalarized searches
end

The linear path skyline computation consists of the two steps outlined in Algorithm 5. In the first step the shortest path distances between all nodes and the target node are computed for all cost criteria. This yields cost vectors which can be scalarized for each of the searches. The purpose of these shortest path distances is goal-direction and acceleration of the multiple searches between the start and target. We propose to employ ParetoPrep but for reasons of comparability, we implemented the established method [148] for the experiments. As before, we refer to [148] as Multi-Dijkstra, because for each of the d cost criteria a Dijkstra search with reversed edges starting at the target (to every node in the network) is performed. In the second step, the algorithm, as outlined in Algorithm 4, performs scalarized searches using $A^\star$ employing the scalarized shortest path distances from the first step as lower bounds.

8.6 Correctness and Termination

In the following, we prove correctness of LSCH, the proposed algorithm for linear path skyline computation. Furthermore, we prove that every scalarized search in LSCH either closes a facet or generates a new linear skyline element. From this statement, we may deduce an upper bound for the average number of scalarized searches which have to be performed in order to compute the linear path skyline.

Theorem 8.2. *Upon termination of the algorithm, P is the complete linear skyline of $\mathcal{P}$.*

Proof. A facet with the vertices V and the normal vector w is closed in $\mathcal{P}$ iff $\min_{x \in V} w^T x = \min_{y \in \mathcal{P}} w^T y$.
The algorithm only terminates when all facets of the current convex hull $\mathcal{V}(P \cup \mathrm{INF})$ of $P \subseteq \mathcal{P}$ are closed facets. As implied by Lemma 8.2, a cost vector $x \in \mathcal{P}$ lies outside the convex hull of $P \subseteq \mathcal{P}$ iff the convex hull $Conv(\mathcal{P})$ has a facet with vertices V and normal vector $0 \neq w \in \mathbb{R}^d$ such that $w^T x < \min_{y \in V} w^T y$. If all facets are closed, no such cost vector exists. Therefore, all vectors are contained in the convex hull. The rest follows from the equality of $\mathcal{V}(\mathcal{P} \cup \mathrm{INF}) \setminus \mathrm{INF}$ and $\mathcal{S}_L$ as shown Lemma 8.3. □

Theorem 8.3. *Let $\mathcal{P} \subset \mathbb{R}^d_{\geq 0}$ be a finite set of points. Each scalarized search either yields a previously undiscovered closed facet in $\mathcal{P}$ or a previously undiscovered linear skyline element.*

Proof. Let $x \in \mathcal{P}$ be the solution of the scalarized search, i.e., $w^T x \leq \min_{y \in \mathcal{P}} w^T y$. The weight $0 \neq w \in \mathbb{R}^d_{\geq 0}$ of the search is the normal vector of an open facet by definition of

	$k = 10$	$k = 50$	$k = 100$	$k = 250$
Maximum number of stages when running ExA[108]				
$d = 2$	20	100	200	500
$d = 4$	130	19650	161800	2573250
$d = 6$	262	2118810	75287620	7817031550
Maximum number of facets when running LSCH				
$d = 2$	20	100	200	500
$d = 4$	45	1225	4950	31125
$d = 6$	60	17300	152100	2511500

Table 8.1: Comparison of known upper bounds of LSCH and known upper bounds of ExA.

our algorithm. Let V be the vertices of this facet. If $w^T x = \min_{y \in V} w^T y$, then the facet is a closed facet by definition. Otherwise, if $w^T x < \min_{y \in V} w^T y \leq \min_{y \in \mathcal{V}(P \cup \text{INF})} w^T y$, x is a new solution. □

Table 8.1 displays the amount of facets leading to path searches in LSCH in comparison to the amount of stages examined in ExA [108]. It can be observed that the theoretical worst case amount of facets is considerably smaller than the worst case amount of the number of stages in ExA.

Theorem 8.4. *The maximal number of scalarized searches and convex hull updates for a linear path skyline with k solutions using the proposed algorithm is $k + \binom{k-\lceil d/2 \rceil}{k-d} + \binom{k-\lfloor d/2 \rfloor}{k-d}$. The number of scalarized searches and convex hull updates using the proposed algorithm is k solutions $O(k^{\lfloor \frac{d}{2} \rfloor})$.*

Proof. The algorithm performs at most one convex hull update per scalarized search. Each scalarized search yields either a new solution or a closed facet. The maximal number of searches and convex hull updates is therefore the number of solutions plus the number of closed facets. Let us note that the vertices of the closed facets are ignored in this case and facets are only regarded as halfspace representations. It is therefore irrelevant if there are multiple vertex combinations that could be used to represent one of the convex hull's intersecting halfspaces. The Upper Bound Theorem [97] states that the number of convex hull facets as a function of the number of vertices is maximal for cyclical polytopes which have $\binom{k-\lceil d/2 \rceil}{k-d} + \binom{k-\lfloor d/2 \rfloor}{k-d}$ facets for k vertices. The asymptotic version of the Upper Bound theorem [124] states that the number of convex hull facets as a function of the number of vertices is $O(k^{\lfloor \frac{d}{2} \rfloor})$ for k vertices. □

8.7 Computing ε-Linear Skylines

The proposed algorithm to compute ε-Linear Skylines is outlined in Algorithm 6. The basic idea of this approach is to check whether a new solution x would lie inside the convex

Algorithm 6 LSCH for Computation of $\mathcal{S}_\varepsilon$

```
begin
    initialize P = INF, open = ∅, closed = ∅
    add front-infinity-facet to open and back-infinity-facet to closed
    while open ≠ ∅ do
        remove an open facet F with normal vector 0 ≠ w ∈ ℝ^d_≥0 and vertices V ⊂ P from open
        find x such that w^T x = min_{y∈P} w^T y
        if w^T x < 1/(1+ε) min_{y∈V} w^T y then
            add x to P
        end
        if w^T x ≥ 1/(1+ε) min_{y∈V} w^T y then
            add F to closed
            if open = ∅ then
                return P \ INF
            end
        end
        determine facets of convex hull and their normal vectors, add new facets to open
    end
end
```

hull of the previous solutions multiplied by $\frac{1}{1+\varepsilon}$. If it does, previous solutions P already contain a cost vector $y \in P$ such that for all $0 \neq w \in \mathbb{R}^d_{\geq 0}$ holds $w^T y \leq (1+\varepsilon) w^T x$.

To simplify the proof of correctness, the convex hull multiplied by $\frac{1}{1+\varepsilon}$ is defined as the ε-convex hull.

Definition 8.5. *Let $\mathcal{P} \subset \mathbb{R}^d_{\geq 0}$ be a finite set of points. The ε-convex hull of $\mathcal{P}$ consists of the vertices of the convex hull of $\mathcal{P}$ multiplied by $\frac{1}{1+\varepsilon}$.*

$$\mathcal{V}_\varepsilon(\mathcal{P}) = \left\{ \frac{1}{1+\varepsilon} x \,|\, x \in Conv(\mathcal{P}) \right\} \tag{8.1}$$

Multiplying a set of points $\mathcal{P}$ by $\frac{1}{1+\varepsilon}$ and then determining the convex hull leads to the same result as first computing the convex hull of the set of points and then multiplying all convex hull vertices by $\frac{1}{1+\varepsilon}$. As can be seen in Figure 8.3, multiplying a set of points with a scalar simply scales all dot products by the same amount and therefore does not affect angles or geometric relationships. The minima and maxima for dot products with any $0 \neq w \in \mathbb{R}^d$ are just scaled. Intuitively this is comparable to changing from coordinates in meters to coordinates in kilometers, which clearly would not affect the number of facets, which vertices belong to which facets and the facets' normal vectors.

Theorem 8.5. *Upon termination of the algorithm, P is the complete ε-linear skyline of $\mathcal{P}$.*

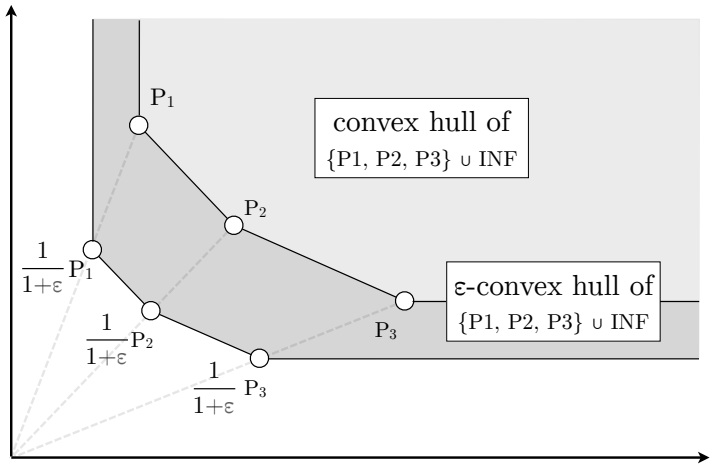


Figure 8.3: Comparison of ε-convex hull with conventional convex hull in $\mathcal{P} \cup \text{INF}$.

Proof. The algorithm only terminates when all facets of the ε-convex hull are closed facets. Analogously to the proof for the conventional LSCH algorithm in Theorem 8.2, all cost vectors in $\mathcal{P}$ are contained in the ε-convex hull of $\mathcal{P} \cup \text{INF}$ upon termination.

The equivalence follows essentially from the above intuition that scaling has no influence on geometric relationships and from the linearity of the scalar product. For any $0 \neq w \in \mathbb{R}^d$, the ε-convex hull of $P \subseteq \mathcal{P}$ contains the vertex $x = \min_{y \in \mathcal{P}} \frac{1}{1+\varepsilon} w^T y$. From this follows that the convex hull contains $(1+\varepsilon)x = \min_{y \in P} w^T y$. If a cost vector $x \in \mathcal{P}$ is part of the ε-convex hull of $P \subseteq \mathcal{P}$, then for each $0 \neq w \in \mathbb{R}^d$ there exists a vertex $y \in P$ of the convex hull of $P \subseteq \mathcal{P}$ such that $w^T x \leq \frac{1}{1+\varepsilon} w^T y$. It follows that if all solutions are inside the ε-convex hull of $\mathcal{P} \cup \text{INF}$, the vertices of the convex hull of $P \cup \text{INF}$ contain the ε-linear skyline. □

8.8 Evaluation

All experiments are performed on a dedicated machine with an 2.2 GHz Opteron Dual Core processor and 64 GB RAM. All algorithms were implemented in Java 1.7 and a single core was used for each experiment. If an algorithm is not able to solve a task in less than 360 seconds, the computation is aborted counted as a timeout.

We tested our algorithms on two types of networks. The first type of network is the road network of Munich as derived from OpenStreetMap. The derived graph consists of 220K nodes and 520K edges. The cost criteria utilized in the experiments are travel time, path length, number of crossings, penalized travel time and energy loss. The criterion travel time assumes travel speeds to equal the speed limits whereas the penalized travel time assumes additional 30 and 15 seconds for each crossing with and without traffic lights, respectively. Energy loss is modeled as in Chapter 6.4, roughly derived from typical battery capacities of electric cars and their respective ranges. It incorporates altitude differences in the following

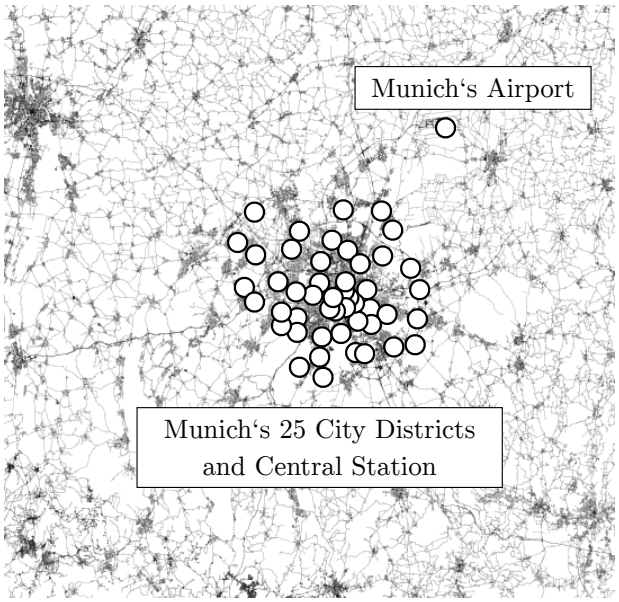


Figure 8.4: Map of Munich with all selected start/end locations.

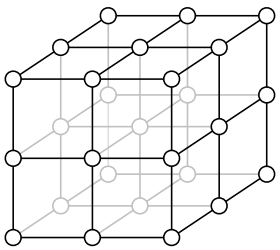
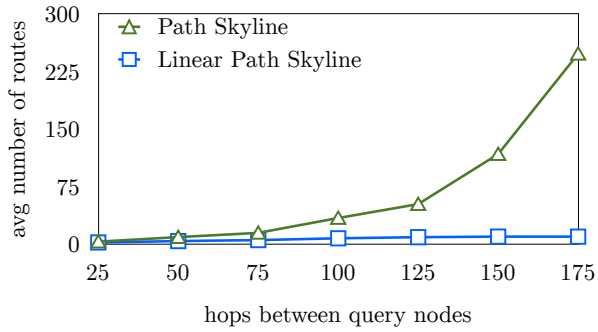


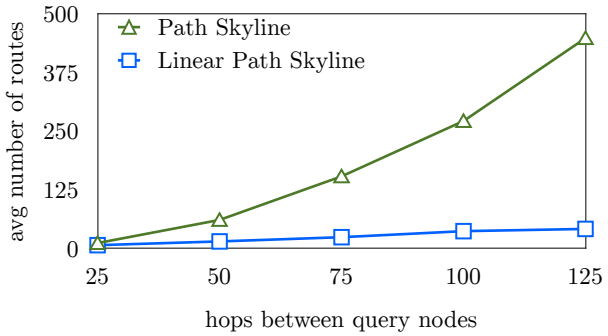
Figure 8.5: $3 \times 3 \times 3$ lattice graph, routing tasks are performed between opposing corners.

sense: ascent increases the energy consumption by the gained potential energy and descent reduces the energy consumption (while negative values are corrected to zero). As before, the authenticity of the cost models used has no influence on the computational benefits of the proposed algorithms. For experiments using two criteria, we employ travel time and path length. For the three criteria setting, we add the number of crossings. Finally for the five criteria settings, all five criteria are used. As sample queries we select the centers of the 25 city districts of Munich plus central train station and airport. Based on these start and target locations, we conduct $\binom{27}{2} \cdot 2 = 702$ sample queries. Figure 8.4 depicts the 27 locations on the map.

The second type of graphs are three dimensional lattice graphs of varying sizes (cf. Figure 8.5). In other words, we generate lattices having $n \times n \times n$ nodes for $n \in \{2, 4, 6, 8\}$. Thus, the graph for $n = 6$ has $6 \cdot 6 \cdot 6 = 216$ nodes. As attributes, we artificially generate five cost values for each edge. For each graph, each pair of diagonal edges is used as a



(a) Number of skyline paths in Munich network with three criteria.



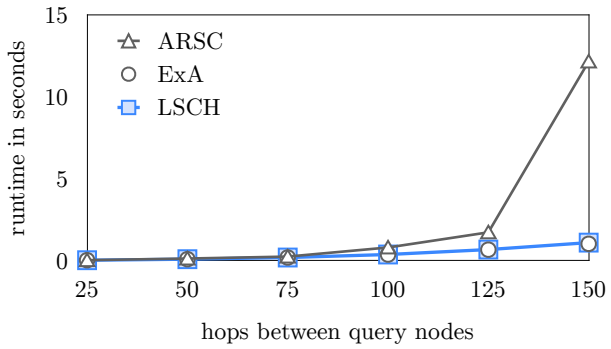
(b) Number of skyline paths in Munich network with three criteria.

Figure 8.6: Comparison of the number of skyline paths for conventional and linear path skylines.

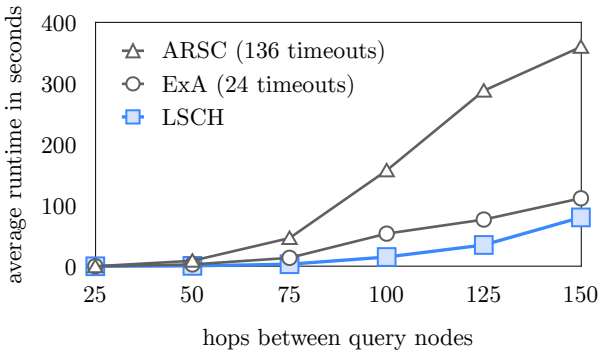
query. Thus, for each graph, we test a total of eight queries.

We compare our new algorithms LSCH and ε-LSCH with the ExA algorithm proposed in [108]. Since ExA is designed for multiobjective linear programs instead of linear path skyline queries, we incorporated the $A^\star$ search for computing shortest paths w.r.t. a linear combination, as in our own algorithm. To have a state-of-the-art comparison partner for computing the ordinary skyline, we compare to ARSC [80]. All tested algorithms start by precomputing the query-specific lower bounds using Multi-Dijkstra [148]. As mentioned before, we use Multi-Dijkstra for reasons of comparability. Employing PareteProp instead would benefit any of the algorithms equally while bound computation would be accelerated.

In a first set of experiment, we compare the number of result paths between conventional and linear path skylines. The set of linearly optimal paths is usually much smaller than the set of all pareto-optimal paths. Furthermore, the increase of result paths with an increasing path length and an increasing number of criteria is far less extreme and thus, stays manageable in many applications. Figure 8.6 displays the size of the path skyline and the linear path skyline in the Munich setting for three and five cost criteria. To measure the distance between start and target, we determine the minimum number of hops (edges)



(a) Runtime in Munich network with three criteria.



(b) Runtime in Munich network with three criteria.

Figure 8.7: Runtime comparison between ARSC, ExA and LSCH in the Munich setting.

between start and target. The plots display the size of the skylines as a function of the number of hops between the locations. It can be seen that the amount of result paths in conventional path skylines increases superlinear with the number of hops. In contrast, the size of the linear skyline displays a weak linear increase. Thus, the size of linear path skylines remains manageable even when the query points are far away from each other. Furthermore, we observe that the increase w.r.t. the number of criteria, is far less dramatic than it is the case for the conventional skyline. While the average amount of paths for five criteria in conventional skylines increases from 25 for three criteria to 145 for five criteria, the linear skyline increases from 5 paths for three criteria to 22 paths for five criteria. To conclude, the increasing number of result paths in linear skylines is significantly slower than for conventional skylines. This holds equally for increasing number of criteria and for increasing distance between start and target.

In the next experiment, we compare the runtime of LSCH to ARSC and ExA in the Munich setting for three and five cost criteria (see Figure 8.7). It can be seen that LSCH and ExA outperform ARSC when computing the linear skyline. This is not surprising, as LSCH and ExA have been developed for linear path skyline computation and are, thus, able to exploit the fact that there are less result paths to compute. When comparing ExA to LSCH, the effort needed to compute a linear path skyline is almost identical for

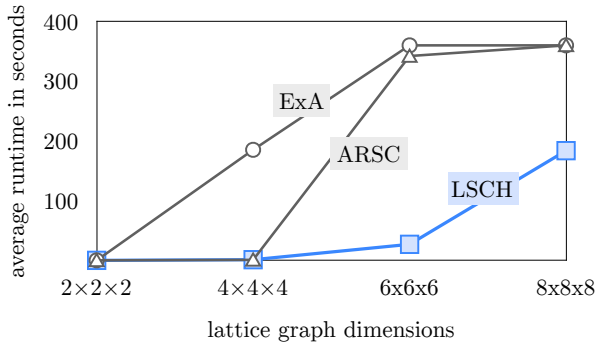


Figure 8.8: Runtime comparison of ExA, ARSC and LSCH in the lattice graphs.

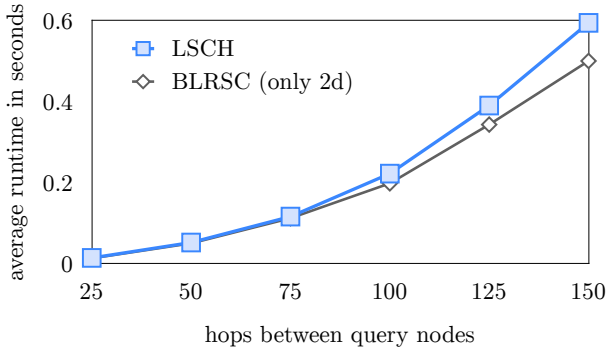
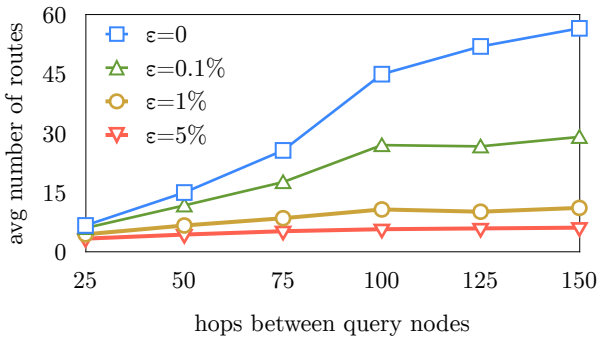


Figure 8.9: Runtime comparison of LSCH and BLRSC (only applicable to bicriteria networks).

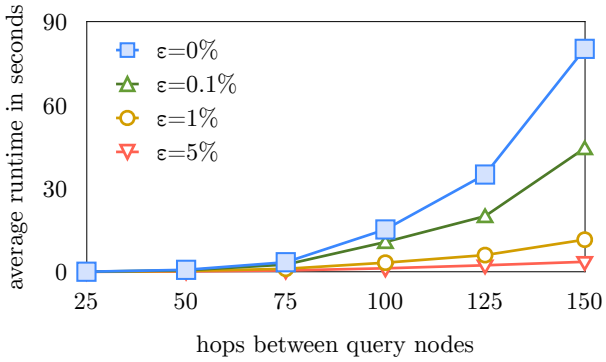
three criteria. The reason for the similar behavior is that both algorithms require about the same amount of $A^\star$-searches in the graph and the number of solved linear equations is comparable. For five cost criteria, we observe that LSCH performs better than ExA, especially for greater distances. ExA has to solve over 8 times more linear equation systems than LSCH, and the number of linear equation systems in LSCH is still rather small with about 22 updates of the convex hull per query. Additionally, for this setting, ExA starts to encounter queries where the time limit of 360 seconds does not suffice to finish the computation. LSCH, on the other hand, finishes all queries within the time limit.

We additionally test the runtime of all three algorithms on the aforementioned lattice graphs. The results are depicted in Figure 8.8. In this setting, ExA performs very badly compared to both ARSC and LSCH, even in rather small graphs. The reason for the large query times of ExA is the large number of stages. This results in a dramatic increase of linear equations which have to be solved. LSCH can avoid this increase by using the concept of the convex hull which prunes large amounts of possible search directions.

In a last LSCH experiment, it is compared to BLRSC (see Chapter 7). BLRSC was developed to compute linear path skylines in bicriteria networks and makes use of particular qualities that only hold in such networks. The results are depicted in Figure 8.9. For smaller



(a) Number of result paths for different ε-values and distances.



(b) Runtime for different ε-values and distances.

Figure 8.10: Impact of varying ε-values, evaluated on the five criteria setting in Munich.

distances, both algorithms display a rather similar runtime behavior. Even for larger distances, the advantage of BLRSC is rather small. Thus, the computational overhead of using LSCH compared to BLRSC barely justifies the overhead to implement a specialized solution for the special case of bicriteria networks.

In a final set of experiments, we compare the performance of our approximative ε-linear path skyline algorithms ε-LSCH with the the exact version. We tested our method with $\varepsilon \in \{5.0\%, 1.0\%, 0.1\%\}$ to show the effect of different ε-values. Figure 8.10(a) shows the impact of the ε-parameter on the size of the result set. The number of results paths is plotted for varying ε-values, from the full linear skyline to the 5 % further restricted skyline, as well as for varying distances. Even a rather small value of 0.1 % allows to roughly drop about half the results. For the maximum value of 5 %, the number of result paths no higher than six paths, even for queries with more than 150 hops between start and target. Thus, ε-LSCH allows to compute small but representative result sets which can be handled more easily by users.

After showing that ε-LSCH is suitable to control the amount of result paths by tuning the ε-parameter, we now want to investigate whether ε-LSCH can exploit the reduced result set size to speed up query processing. Figure 8.10(b) illustrates the corresponding

runtime in the five criteria Munich setting. ε-LSCH reduces the processing times roughly by the same amount as the number of results. Hence, ε-LSCH allows to compute reduced, representative sets of alternative paths in very efficient time. By tuning the ε-parameter, the number of result paths can be reduced in a controlled way.

Chapter 9

Summary

This part of the thesis investigated multicriteria networks and optimal paths in such networks. The problem is interesting from a (graph) theoretic point of view but also has practical implications. Computing the standard set of optimal paths, the path skyline, in such networks is a costly task and has two major drawbacks. First, the number of skyline paths may be exponential in the number of criteria of the network. Second, with increasing number, the diversity and therefore the significance of the result paths decreases. This part tackled the aspect of efficiency as well as the increasing result set size.

In Chapter 6, ParetoPrep was introduced. ParetoPrep is a means for computing optimal lower bound vectors. In general, algorithms for computing alternative paths in large multicriteria networks employ lower bound estimations of the cost for reaching the target from a given node for each of the d cost criteria. To generate these bounds, the established method [148] runs an single-source all-target Dijkstra search for each criterion. This requires d separate searches, each visiting the whole network. These lower bounds serve various purposes. First, they directly imply the single-criterion shortest paths. Second, they may be employed as lower bound cost approximations in different algorithms, for instance, to compute the path skyline. ParetoPrep yields significant speed-up compared to the state-of-the-art method [148]. Information which is usually generated in multiple graph traversals, is generated in a single graph traversal. ParetoPrep achieves this by instantly using the attained information for excluding non-promising graph regions. This limits the search space while guaranteeing optimal bounds. Furthermore, it has been proved that ParetoPrep visits all nodes which are potentially part of any skyline path. Employing ParetoPrep as a precomputation step for path skyline algorithms, it is possible to compute skylines in networks and for settings where previous methods reached their limits. Using ParetoPrep as a means for single-criterion cost-optimal path computation yields respectable results. This could be used to simultaneously compute multiple paths according to pre-defined trade-offs (e.g., as in the application in Chapter 16).

In Chapters 7 and 8, an alternative definition of the set of optimal paths is proposed, the linear path skyline. In general, it contains less elements than the conventional path skyline. First, in Chapter 7, the problem of computing the linear path skylines is solved for networks with two cost criteria where particular properties can be exploited. We propose

an efficient algorithm which is based on custom techniques for managing and updating linear skylines in bicriteria networks. In Chapter 8, the problem is solved for arbitrary multicriteria networks. The approach is based on the relation between linear path skylines and convex hull. The normal vectors of the facets of an augmented convex hull serve as weights for multiple $A^\star$-searches. Every search either returns a new skyline path or closes a facet, i.e., a part of the cost space. Hence, the linear path skyline is computed iteratively based on geometric properties of the cost space. To cope with cases where there has to be a limited amount of representative results, we propose the ε-linear path skyline query which computes a subset of the linear path skyline. Linear path skylines represent a relevant subset of the conventional path skyline which in general contains less and more diverse results. In conclusion, linear path skylines facilitate handling of results and accelerate computation.

Part III

Network Enrichment and Paths in Enriched Networks

Chapter 10

Introduction

Nearly all routing algorithms, commercial as well as scientific, minimize cost functions. Usually single criteria are minimized, most popularly distance, sometimes, as in Part II, multiple criteria are minimized simultaneously. However, minimizing cost does not necessarily maximize quality. For instance, on the weekend a driver might prefer a more scenic path over a fast one. Or a tourist might wish for an attractive path from their hotel to a restaurant rather than a short one. This kind of routing, where optimality is not necessarily equal to cost-optimality but defined by alternative criteria, has only been studied for particular use cases and hardly finds application in commercial systems. This is due to several problems. First, it is unclear how to make quality measurable as it is inherently subjective and infinitely diverse. Second, for reasonable application, the quality measure would have to be derived automatically at appropriate scale. Third, generating routing suggestions which purely rely on quality is not feasible. Any quality-optimal path would be of infinite length. Therefore the quality measure has to be traded-off against some cost function. In this part, we make an advance towards solutions to these problems.

In Chapter 11, we mine crowdsourced data for qualitative information reflecting some notion of popularity. This information is in turn used to enrich the underlying road network with the knowledge of the crowd, in order for it to be applied in routing algorithms. One of the rare examples where quality is measured in commercial solutions has been developed by navigation system producer TomTom. Some devices provide routes optimized for motorcycle drivers where particularly steep, winding and/or mountainous roads are suggested. In this case, the content is curated by experts and the routing suggestions correspond to one particular scope of application, i.e., routing for motorcyclists. In contrast, we propose to mine this information from crowdsourced data which allows for scalability. We survey heterogeneous sources of spatial, spatio-temporal and textual data reflecting different notions of popularity. We explore the intricate process of data categorization, knowledge extraction and network enrichment. The proposed methods are evaluated on multiple real-world data sets. We show that qualitative information, as captured by crowdsourced data, can indeed be integrated into road networks to improve the subjective quality of result paths. Chapter 12 describes a demonstration framework which partially employs the proposed methods for the use case of tourist route recommendation.

Chapter 13 proposes a novel graph routing problem where such quality measures may be applied as value functions. The Twofold Time-Dependent Arc Orienteering Problem (2TD-AOP) extends the family of Orienteering Problems (OPs). In its original definition, the NP-hard OP asks to find a path from a source to a destination maximizing the accumulated value while not exceeding a time budget. Variations of the OP and AOP are frequently used for tourist route recommendation where the most valuable sights and attractions are to be visited in limited time. Other applications include mobile crowdsourcing tasks (e.g., repairing and maintenance or dispatching field workers) or logistics problems (e.g., crowd control or controlling wildfires). In the proposed extension 2TD-AOP, travel times and value functions are assumed to be time-dependent. The dynamic values model, for instance, the scenicness of spots which may change over the course of a day. We discuss this novel problem, show the benefit of time-dependence empirically and present an efficient approximative solution, optimized for fast response systems. The presented approach is the first time-dependent variant of the AOP to be evaluated on a large scale, fine-grained, real-world road network. We show that optimal solutions are infeasible and solutions to the static version of the problem are often invalid in time-dependent networks. We propose an approximate dynamic programming solution which produces valid paths and is orders of magnitude faster than any optimal solution.

This research has been previously published in [68], preliminary results have been published in [66, 136, 135]. The work in Chapters 11 and 13 is currently undergoing peer review.

Chapter 11

Crowdsourced Data and Knowledge Enrichment

11.1 Introduction

Crowdsourced data has benefited many scientific disciplines by providing a wealth of new data. Technological progress, especially smartphones and GPS receivers, has facilitated contributing to the plethora of available information. Nowadays, a large share of crowdsourced data (often also: user-generated content/information) contains spatial information, often referred to as volunteered geographic information (VGI). The term, however, is very generic and refers to various different types of content. VGI may refer to geo-tags which have been explicitly or implicitly added to a tweet, picture or status update. Also, a check-in at a registered location or a review for a restaurant's menu in a social network can be considered VGI. Other examples include the shared record of a user's favorite cycling route or, in the broader sense, a textual description of a museum in a blog entry.

In this work, we explore how different sources of user-generated data may be used to enrich road networks in order to better represent the human way of thinking and liking. It is our hypothesis that crowdsourced data reflects the "mind of the crowd", that it conveys semantic knowledge and that it expresses sentiment. Algorithms as used by navigation systems, however, rely on purely quantitative measures, on "hard" metrics. We argue that routing has a great cognitive aspect which is ignored by plain turn-by-turn instructions derived from absolute measures. Using crowdsourced data as a proxy, we aim to bring routing algorithm in line with users' preferences and cognition. For example, under the assumption that Flickr users take photos of particularly appealing places, a routing algorithm which is "steered in the direction" of areas where photos are dense, generates paths which are likely to be more appealing. Therefore, our goal is to integrate the wealth of crowdsourced spatial data into road networks, such that existing routing algorithms can be applied to find routes that better reflect human perception.

The aforementioned diversity of spatial crowdsourced data constitutes the main challenge of this work. Some data sources are noisier than others, and some have greater depth

of information than others. For example, check-ins at curated locations are more reliable than geo-tagged tweets which are dependent on the quality of the user's GPS signal and the density of possible Points of Interest (POI) in their environment. A plain textual mention of a "Tibetian Yoga Studio", for instance, may be ambiguous and might not be mapped to a unique location. As textual sources are particularly noisy, we develop specific methods to handle their inherent ambiguity. Aside from uncertainty in geo-spatial locations, sentimental information expressed in crowdsourced data may also be highly uncertain: While reviews in location-based services often represent distinct and thorough opinions, tweets usually contain extremely condensed sentiment. In contrast, travel blog entries are typically more focused on the entirety of a trip. Because of this diversity, there is no absolute truth on what to extract from which source. We do not claim to extract all possible information, instead, we want to give a broad overview on how to generate knowledge from a variety of crowdsourced data sources which can in turn be used for routing applications.

Most data sources reviewed in this chapter provide information about particular geographical locations or specific POIs. Depending on the type of data, these POIs might represent different categories. For example, mentions of POIs in travel blogs predominantly cover sights while check-ins in location-based services happen mostly at restaurants, bars or clubs. This effect can be used to enhance the semantic information being extracted. Another phenomenon which we try to trace in the process of knowledge extraction are particular relations between a number of POIs. Especially for the application of routing, it makes sense to not only consider singular POIs but multiple POIs representing a particularly related set or a specific sequence of POIs. When considering the sequence of check-ins of one user during a single day, the sequence may indicate a recommendable itinerary. When considering POIs mentioned in travel blogs, it might make sense to recommend those which are perceived positively, or it might make sense to recommend those POIs together that have a strong spatial connectivity. In contrast, when considering spots where notably many photos are taken, each particular spot might represent a great lookout point on its own.

As an example, consider the routing scenario in Figure 11.1 which is set in the city of Paris, France. The continuous line represents the conventional shortest path from starting point "Gare du Nord" to the target at "Quai de la Rapée" while the dot dashed and dotted lines represent alternative paths computed in enriched networks as proposed in this chapter. The triangles in this example mark POIs as extracted from travel blogs, in this case mostly landmarks and sights. For instance, the dot dashed path on the bottom right passing recognizable locations such as "Place de la République", "Cirque d'hiver" and "la Bastille" reflects the knowledge of the data source, i.e., it satisfies the requirement of being touristically appealing. Compared to the conventional shortest path, it will yield greater value for travelers.

Upon extraction of the crowdsourced information, we proceed to enrich the underlying network. It is our goal to merge the inherent metrics of the network (like distance and travel time) with the semantic knowledge filtered from the data sources. Ideally, this leads to networks with cost criteria which reflect the user-generated information, allowing for alternative routing algorithms that do not only take quantitative measures but also

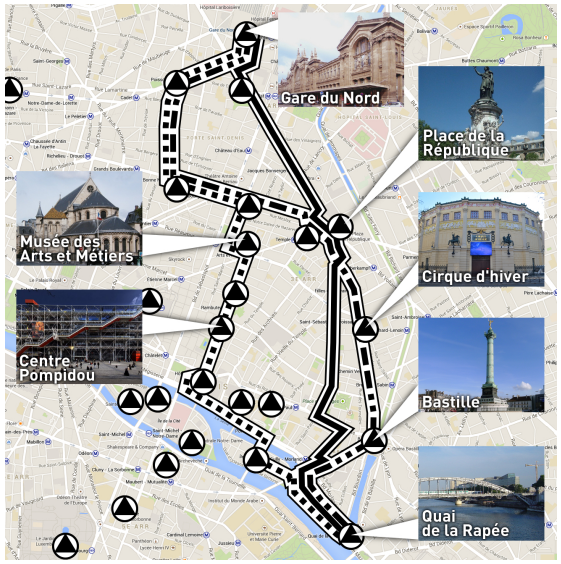


Figure 11.1: Shortest (continuous) and alternative paths (dot dashed and dotted) along POIs in Paris, France. This result is output of the methods presented in this chapter.

qualitative knowledge into account. We propose and investigate different methods of enrichment of the underlying road network, and, additionally, we introduce a meta-network whose nodes correspond to POIs and whose edges correspond to shortest paths connecting them. Our evaluation is based on various real-world crowdsourced data sets. In the process of knowledge extraction different methods are applied, according to different points of view, as described above.

The methodology presented in this chapter is subdivided into three sections. First we give an extensive overview of different data sources, categorize them and mention advantages as well as possible drawbacks. We tackle the aspects of precision, reliability and information content. Second, we explore how crowdsourced knowledge can be extracted from different data sources. We present different approaches for singular, pairwise or sequential occurrences of POIs, and stress that diverse knowledge can be mined, depending on the approach. Third, we investigate how the knowledge may be integrated into the underlying road network in order to be implemented in routing algorithms. We propose to directly enrich the network or, alternatively, to build a meta-network. In our experimental evaluation, we show that routing algorithms indeed benefit from incorporating crowdsourced knowledge. More precisely, at the cost of marginal increase in path length, we may generate paths which yield significantly higher value according to the data source which has been integrated.

This work extends the research presented in [135], which focused on spatial relations between pairwise occurrences of POIs. Incorporating ideas published in [66], we additionally present means for enrichment based on singular occurrences of POIs. Previously unregarded were sequences of POIs (triples or longer), which are included in this chapter, theoretically as well as experimentally. In addition, further data sets were integrated. The

previously published content has been restructured and amended to give a comprehensive survey of the means and methods for the enrichment of road networks with knowledge extracted from crowdsourced data.

Structure of this chapter: In Section 11.2 we summarize research from different areas which is related to this work. Section 11.3 classifies and distinguishes data types and mentions examples and sources. How and which particular knowledge may be extracted from these sources is detailed in Section 11.4. Section 11.5 introduces several methods for the enrichment of existing road networks with the knowledge extracted in the step before. The whole procedure, from data processing over knowledge extraction to network enrichment, is evaluated experimentally in Section 11.6.

11.2 Related Work

In this section, we give an overview regarding the research relevant to our work which we divide into the following categories:

1. trajectories: mining semantic information and trajectory enrichment
2. qualitative routing
3. (crowdsourced) touristic trip planning
4. the arc-orienteering problem

11.2.1 Semantic Mining and Enrichment of Trajectories

The discovery of semantic places through the analysis of raw trajectory data has been investigated thoroughly over the course of the last years. The objective of this field of research is to analyze user trajectories and, in combination with POI databases, to extract semantically relevant places based on the spatio-temporal patterns (number of times POIs are visited and time spent there). The authors in [91, 155, 109] provide solutions for the semantic place recognition problem and categorize the extracted POIs into pre-defined types such as “home”, “work”, “education” and “shopping”. Moreover, the concept of “semantic behavior” has recently been introduced. This refers to the use of semantic abstractions of the raw mobility data, including not only geometric patterns but also knowledge extracted jointly from the mobility data and the underlying geographic and application domains in order to understand the actual behaviour of users in movement. Several approaches like [1, 111, 140, 156, 140, 157] have been introduced in the last decade. The core contribution of these works is in the development of a semantic approach that progressively transforms the raw mobility data into semantic trajectories enriched with POIs, segmentation and annotations. Finally, a recent work, [37], extracts and transforms the aforementioned semantic information into a text description in the form of a diary. The difference to what is presented in this work is that these approaches do not integrate the extracted semantic information into the road network. Instead, they combine trajectories

with POI databases to extract semantic information on POIs and possibly enrich other trajectories (e.g., paths computed by an arbitrary algorithm) with semantic information. For instance, a sequence of timestamped geo-coordinates might be mapped to the semantic sequence: `home` → `work` → `kindergarden` → `supermarket` → `home` → `restaurant`.

11.2.2 Qualitative Routing

The term qualitative routing is not well-defined. We use it to describe two approaches to routing which do not solely rely on absolute measures and are therefore more "qualitative" rather than purely quantitative. First, the computation of routes which are easier to memorize, describe and follow. Second, the computation of routes which are particularly scenic, interesting or popular.

The first problem has been tackled from various angles and even research disciplines. Following a rather cognitive line of argument, the authors of [31] minimize the complexity of a route description. This is done by finding a trade-off between distance and weights which describe the complexity of the routing description at every intersection. Different cognitive models for the complexity can be employed. Later works explore the role of landmarks in route descriptions [32] and strategies on how to create compact route descriptions [118]. In [154], the authors explore the concept of route descriptions in detail by defining and evaluating agent models and deriving an agent-centric qualitative representation of the graph. A less cognitive and more spatial approach is chosen by the authors of [119]. They introduce cost criteria that allow for a trade-off between distance and complexity based on network properties. It is possible that the cognitive agent models could benefit from the extraction process presented in this work, however, this goes beyond the scope of the work. Our methods are based on crowdsourced knowledge such that ideally the crowd becomes its own agent, possibly making complex models obsolete.

The second problem, is most prominently examined in [114]. This work proposes a method for computing beautiful, quiet and happy paths, as the authors phrase it. In order to quantify these three qualities, the authors rely on explicit statements about the locations, obtained from a platform which asked users to specifically rate photos of locations according to the three categories. Obviously, this is a valid approach, however, it does not scale well. This is the reason why we propose to mine this kind of information from existing crowdsourced data, in order to avoid acquiring ratings in the manner of mechanical turks on a global scale. Another way to generate interesting paths is to stitch previously recorded trajectories together, obtaining trajectories where every subtrajectory has previously been traveled by users and is thus "popular" following the definition in [18]. However, this only reflects a notion of common usage not taking into account, why a specific subtrajectory has been favored. For instance, when mining trajectories of commuters, fast paths are most likely to be chosen by most users, but when mining trajectories of runners, green paths will likely be preferred. Connecting parts from both sets, the obtained trajectories might fit into neither class.

11.2.3 Touristic Trip Planning

There are numerous variations of touristic trip planning problems. Most are related to the original trip planning query (TPQ) [84]. The input of a TPQ are start and target locations in a weighted graph as well as a set of categories (of POIs). The output is the cost-optimal path from start to target visiting exactly one instance of each category. The NP-hard Traveling Salesman Problem can easily be reduced to the TPQ by assuming that every POI belongs to its own category. Hence, efficient solutions to the TPQ or further constrained modifications are heuristic. However, in real-world scenarios the candidate sets of POIs may often be narrowed down drastically (e.g., by spatial pruning or additional constraints), allowing for a full enumeration of all solutions within seconds.

Early variations of the TPQ, [74, 15], allowed for a particular order of some of the categories and for further constraints regarding the entities of the categories. Further constrained modifications of the query often focus on the use case of touristic trip planning, occasionally also referred to as itinerary planning, largely summarized in [46]. For example, the query objective may be to maximize the subset of a set of predefined POIs which can be visited in a tour with a certain time-constraint [43]. Younger works in this area exploit crowdsourced data for POI categorization [13], for POI-popularity estimation [64], for POI-recommender systems [9], for the determination of opening hours or recommended visiting times [63], for deriving the average duration of stay at each POI [23] or for combinations of the above. In contrast to these works, we do not focus solely on the purpose of itinerary planning but follow a more general approach. We emphasize the process of knowledge extraction with regard to the diversity of the crowdsourced data, its quality and properties. Also, we investigate various different data types and sources and give insight on the aspects of related POIs and sequences of POIs. It is our belief that itinerary planning methods like the above could be refined using the contributions of this work. However, we choose not to adapt the rather rigid structure of the itinerary planning queries.

11.2.4 (Arc) Orienteering Problems

Another relevant family of works adresses the NP-hard orienteering problem (OP) and the arc-orienteering problem (AOP) and variants thereof. The input of both is a weighted graph as well as start and target locations and a time budget. Additionally, there is a non-negative value assigned to the nodes of the graph of the OP, while for the AOP there is a value assigned to the edges of the graph (or arcs, hence the name). The output of both queries is the path which has the greatest accumulated value among all paths from start to target not exceeding the time budget. These problems often find application in recreational path planning and tourist route recommendation. The contributions of this chapter have great relevance for the OP and AOP, as the extracted knowledge can be conceived as a value associated with nodes or edges. Using the techniques proposed in this chapter, crowdsourced information may be integrated into networks, adding other notions of "value" to the OP and AOP. In Chapter 13, a novel extension of the AOP is presented. This extension allows for time-dependence in both, travel time and value. While this

chapter focuses on knowledge extraction and network enrichment, Chapter 13 focuses on path computation according to the constraints of the novel AOP extension. For related work on the OP and AOP, we refer the reader to Section 13.2.

11.3 Data Types and Processing

In order to enrich a road network, we mine data from different sources. Each type reflects a certain notion of value or popularity. Data may include geo-tagged multimedia data, check-in data as recorded by location-based (social) networks, GPS-trajectory data and even purely textual data. For each of these data sources, we propose one or more methods of knowledge extraction. We derive numerical values which can in turn be mapped onto the underlying road network and, thus, we enrich the road network with condensed crowdsourced information. As a first step, this section describes the data types and their sources as well as some preprocessing methods.

11.3.1 Spatially Enriched Data

By spatially enriched data we mean data with attached or inherent geo-tags, e.g., check-ins at pre-defined locations, geo-tagged tweets or geo-tagged images. The great benefit of basic spatial point data is its wide availability and large coverage. Most services relying on spatially enriched data provide APIs or at least exemplary data sets, such as Foursquare[1], Yelp[2], Twitter[3], Flickr[4] or the inoperative but oft-cited Gowalla. Check-ins at pre-defined locations, as recorded by location-based networks, provide precise information concerning the whereabouts of users (if not assuming deliberate misuse). This kind of reliability does not hold for all geo-tagged data, as the GPS sensors which typically provide the locations are never exact but, on the contrary, often imprecise. Other reasons for unreliability may be data-dependent. Consider for example photos of a landmark like the Eiffel Tower. Because of its height, it is visible from far away, hence geo-locations of photographs may vary greatly. Therefore, it often makes sense to also consider dedicated meta-information such as Flickr tags or Twitter hashtags. How to incorporate this kind of information for probabilistic validation is explained in Subsection 11.4.1.

Of course, the spatial information is just an additional component to the actual content of the service, e.g., Twitter's tweets, Yelp's reviews or Flickr's photos. Depending on the content, different notions of information, value or popularity are reflected. For example, the number of Flickr photos in the vicinity of a particular POI may be interpreted as a measure for its appeal or, more generally, its popularity (as in [89, 66, 135, 68]). The assumption is that people tend to take photos at particularly appealing places, of scenic spots, of nice architecture. This might not be the case for all photo-sharing services

[1] www.foursquare.com
[2] www.yelp.com
[3] www.twitter.com
[4] www.flickr.com

(e.g., Snapchat), but whoever has scrolled through Flickr pictures will most likely agree to the assumption. Similarly, the number of check-ins in location-based networks like Foursquare and Yelp may also be considered a measure for trendiness or popularity (as in [9, 13, 63]). While Flickr photos tend to describe aesthetic appeal, accumulated check-ins rather reflect the popularity of restaurants, bars or clubs (according to the users of the particular service). This underlines the diversity of information provided by different data sources. A particularly ambiguous example are tweets which range from advertising over news and personal opinions to comical as well as serious content. Nevertheless, we choose not to disregard Twitter, as it is a rich source of crowdsourced information with millions tweets per day. How we cope with the ambiguities and extract knowledge from the data is also explained in Subsection 11.4.1.

11.3.2 Spatio-Temporal Data

We define spatio-temporal data as sequences of timestamped locations, possibly enriched with additional data, such as textual descriptions of a trip, of locations visited along the trip or meta-information like the vehicle used for the trip or the weather on that particular day. The prime example of spatio-temporal data are trajectories as collected by contributors to the open source map service OpenStreetMap[5], as contributed by users of shared mobility services such as Capital Bikeshare[6] or as recorded and uploaded by runners, cyclists or others to platforms like Endomondo[7]. However, any temporally ordered sequence of geo-locations can be classified spatio-temporal data, like consecutive check-ins of a particular user of a location-based network. When hand-ling trajectory data, usually the movement pattern is of interest, i.e., the actual path chosen by the user. Of course, this information is not available when mining consecutive check-ins. Although the actual path cannot be determined, a sequence of visited locations still might provide valuable information. For instance, if a significant number of users has visited the same locations within the time frame of a day (according to the timestamps), one may recommend visiting these locations as part of a day's trip (cf. [64], [9], [23]). Like in this case, spatio-temporal data may often be reduced to ordered sequences of spatial point data. Let us note that spatio-temporal data clearly suffers from the same measurement errors as spatial point data, but employing map-matching approaches (which often benefit from taking the timestamps into account) measurement errors are more easily rectified.

11.3.3 Textual Data

While spatio-temporal and spatially enriched data contain explicit spatial information, we now turn to implicitly spatial data. Pure textual narrative such as (non-geo-tagged) tweets, blog entries or other text corpora, often contains mentions of POIs. Obviously, a narrative is inherently noisy. Aside from the lack of geo-locations, the ambiguity of

[5] www.openstreetmap.org
[6] www.capitalbikeshare.com
[7] www.endomondo.com

language and, more specifically, duplicate identifiers raise more difficulties. For instance, "Chinatown" is not a unique identifier, neither is "Gelateria Bella Italia" (it does not even relate locally), and a "Tibetian Yoga Studio" is most likely not in Tibet. Nevertheless, we want to stress the importance of textual data as it is a particularly rich source of crowdsourced information. Although authoring explicit spatial data has been facilitated by technical progress, it sometimes still requires special applications and/or special knowledge, e.g., when contributing to OpenStreetMap. Hence, many users are more comfortable using narrative when generating content. Especially when evaluating visited places, users often generate narrative using qualitative adjectives such as "beautiful", "interesting" and "cool". Similarly with movement patterns, a lot of users describe their motion using toponyms (landmarks) and spatial relations ("near to", "next to", "close by", etc.) rather than using geo-coordinates. Hence, there is a largely unused abundance of crowdsourced knowledge in the form of blog entries or other narratives (freely) available on the internet. In the following, we describe how we mine toponyms and in turn geo-locations from travel blogs as an example for crowdsourced textual data. This is a prerequisite for the knowledge extraction methods presented in Subsections 11.4.1 and 11.4.3.

In order to gather travel blog entries, we use standard web-crawling techniques and compile a database consisting of 250,000 blog entries from 20 different travel blogs[8] as presented in [136, 134]. Extracting qualitative information from text requires the detection of toponyms, i.e., placenames within the raw text. By geoparsing, candidate phrases containing references to POIs are identified. Let us note that geoparsing is a method well-proved in the field of Natural Language Processing, and we used the Natural Language Toolkit [88] in our implementation. Subsequently, we geocode these POIs, i.e., we map the POIs onto geo-locations. This is done using the geographical gazetteer database GeoNames[9] which contains over ten million POI names, their synonyms and their coordinates worldwide. Of the 500,000 POIs mined from the text corpus, we were able to geocode 480,000. For further details, we refer the reader to [133] and [134].

It remains to conclude that many crowdsourced data sets are composed of more than one data type. For instance, a location-based social network like Yelp provides curated locations, numerical ratings as well as often extensive written reviews. It also provides a social qualities like following and befriending. Mining data from broadband crowdsourced content is rarely straightforward and decisions are always application-dependent.

11.4 Knowledge Extraction

This section in detail describes the knowledge extraction process of qualitative information from crowdsourced data with increasing complexity. First, singular occurrences of POIs in the data are described, then the importance of relationships and sequences is stressed. Particular attention is paid to textual data due to its ambiguity and noise.

[8] www.travelblog.com, www.traveljournal.com, www.travelpod.com
[9] www.geonames.org

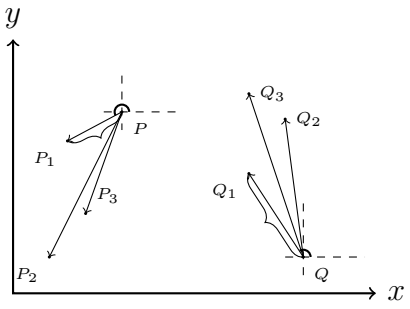
11.4.1 Single Occurrences

First, we describe the simplest approach to knowledge extraction from crowdsourced data sources. Accumulation of separate occurrences is a basic but effective method for quantifying spatial information resulting in simple numeric scores which reflect some notion of quality. As mentioned before, the number of photos taken in the vicinity of a POI or the number of check-ins at particular locations have both been employed as a measure for popularity in research. When considering check-ins, the basic approach is to sum up the number of check-ins at each location and directly use this score as a measure (normalized by the maximum number of check-ins). A straightforward extension is to factor in the location-specific rating provided by the corresponding location-based social network, e.g., the ten-point-rating of Foursquare or the five-star-rating of Yelp.

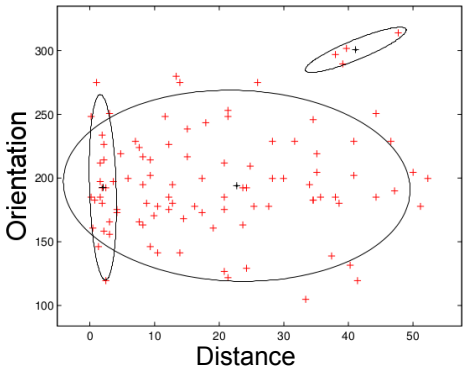
In contrast, geo-tagged photos are not necessarily taken exactly at the POI they depict but in a more or less strict vicinity. The basic approach is to consult a POI database (like GeoNames or appropriately tagged OpenStreetMap nodes), to define a distance threshold ε and to sum up the number of all photos in the ε-range of each POI. However, depending on the threshold value, photos might be assigned to more than one POI. This can be avoided setting $\varepsilon := d_{\min}/2$, where $d_{\min}$ is the minimum distance between any pair of POIs in the considered network. Still, photos might be assigned to the wrong POI, as large landmarks or buildings on spacious esplanades, for instance, are visible from a greater distance than smaller ones. Another possibility is to employ the Reverse Nearest Neighbor query predicate, which for each POI finds the photos which have this particular POI as their Nearest Neighbor. Further refined results may be achieved when additionally considering the Flickr tags (or similar features on other platforms). Using a gazetteer database such as GeoNames which provides synonyms (e.g., "Cologne Cathedral", "Kölner Dom", "Hohe Domkirche St. Petrus"), the text tags of all photos in a greater vicinity of a POI may be scanned for word matches and assigned to that particular POI. The number of photos assigned to a POI again gives an estimate for its popularity. Normalized by the maximum number of photos of one POI, this gives a score in the interval $[0, 1]$.

Not all photos are text-tagged properly. Therefore, we propose a probabilistic modeling approach for non- or insufficiently text-tagged photos. Consider the following scenario: For a specific POI, for example the "Eiffel Tower", let there exist n geo-tagged photos $\{p_1, \ldots, p_n\}$ which are also properly text-tagged, i.e., as "Eiffel Tower", "Tour Eiffel" or another valid synonym. We may use these to train a probabilistic model in order to classify the geo-tagged but not text-tagged photos in the vicinity, as proposed in [133]. From the POI's actual location (as stored in the database) and the photos' geo-tags, we may compute a two-dimensional spatial feature vector $v = (r, \phi)$ for each photo, containing the Euclidean distance and the orientation w.r.t. the counterclockwise rotation of the x-axis (centered at the actual location). These n feature vectors may be used to train a probabilistic model. In Figure 11.2(a) two POI locations P and Q are illustrated. Each POI is assigned three photos (for one of which distance and orientation are visualized).

We propose to train a Gaussian Mixture Model (GMM) which is a well-proved and extensively studied method for many supervised and unsupervised learning problems [7].



(a) Illustration of spatial feature vectors between two POIs P and Q and their respective photos P_i and Q_i.



(b) 3-component GMM trained on two-dimensional point data set visualized upon convergence of Expectation Maximization.

A GMM is a weighted sum of a fixed number M of Gaussian component densities

$$\mathcal{P}_\theta(v) = \sum_{i=1}^{M} w_i g(v; \mu_i, \Sigma_i)$$

where v is an l-dimensional vector, w_i are the mixture weights (summing to 1) and

$$g(v; \mu_i, \Sigma_i) = \frac{1}{\left((2\pi)^l \det(\Sigma_i)\right)^{1/2}} \exp\left(-\frac{1}{2}(v-\mu_i)^T \Sigma_i (v-\mu_i)\right)$$

is an l-variate Gaussian density function with mean vector $\mu_i \in \mathbb{R}^l$ and covariance matrix $\Sigma_i \in \mathbb{R}^{l \times l}$. The model is fully characterized by the weights, mean vectors and covariance matrices, collectively represented in $\theta = \{w_i, \mu_i, \Sigma_i\}, i = 1, \ldots, M$. In our case, $l = 2$, the dimensionality of the spatial feature vectors v. Figure 11.2(b) shows a 3-component GMM trained on two-dimensional point data.

For the parameter estimation of each Gaussian component, Expectation Maximization ([28]) is the state-of-the-art technique which updates the parameters of the components iteratively w.r.t. a given (feature) vector set until a convergence threshold is reached. Thus, using all geo-tagged and properly text-tagged photos of a particular POI POI, we obtain a probabilistic model $\mathcal{P}(\cdot \mid \theta)$ where θ is specific to POI. Applying the model to the spatial feature vector of a non-text-tagged photo, we obtain a probabilistic estimation of the photo's affiliation to that POI. If this probability is sufficiently high, i.e., exceeding an application-dependent threshold, the photo may be assigned to the POI. Consider Figure 11.2(a), with a GMM trained on the given data without photo P_i, we could infer (with high probability) that P_i indeed depicts POI P and not POI Q.

Besides check-ins and photos, we propose another measure for popularity based on single occurrences of POIs in crowdsourced data. While photos tend to measure appeal and check-ins indicate popularity, we also exploit a combination of two crowdsourced data

sets to infer sentiment according to Twitter. As described in Subsection 11.3.3, we have access to the names and geo-locations of touristically relevant POIs. Crawling Twitter's public feed for mentions of each POI's synonymous toponyms, it is possible to obtain a significant number of tweets in the area of interest. Finally, to each tweet we apply the sentiment analysis feature of the Natural Language Toolkit [88] and aggregate the output score for each POI separately. This gives us another notion of popularity in terms of a numeric score for each POI reflecting the sentiment of Twitter users in regard to that POI.

11.4.2 Pairwise Occurrences

So far, we have mentioned how knowledge can be extracted from single occurrences of POIs. It is our goal to propose methods of knowledge extraction for routing purposes. Thus, we now investigate how to consider sequences rather than separate POIs. By considering pairs of POIs, for example, it is possible to model connections between POIs. Depending on the nature of the pairings, different knowledge will be reflected in the connections. For example, if a significant number of users checked in at the same two locations, it might make sense to recommend the pair of locations rather than each location separately when planning an itinerary or a route. Check-ins at pre-defined locations are not subject to spatial variation as the set of locations is curated. Therefore, it suffices to simply sum and normalize occurrences of pairs of check-ins in order to score their value when visited in combination. As before, additional information such as network-specific ratings may easily be taken into account. More importantly, when considering pairs of POIs visited in conjunction, spatial connectivity will become a factor. We explore this aspect in the following using purely textual data to show that even intricate data sets can be mined for knowledge about pairwise occurrences, such as spatially close pairs of POIs.

We make use of the text corpus described in Subsection 11.3.3. More precisely, we now consider toponyms (mapped to POIs) which are linked by spatial relations describing closeness, such as "nearby", "next to", "at", "in" or "in front of". We refer to these spatial relations as closeness relations. If a pair of POIs is mentioned significantly often linked by a closeness relation, one may deduce that the two POIs are in fact close to each other. In the following, we present a probabilistic approach to mining pairs of POIs which – according to the text corpus of travel blogs – stand in close spatial relation to each other. From the text corpus of 250,000 travel blog entries, we were able to mine 660,000 triplets of the form $(P_i, \textit{closeness relation}, P_j)$. Here and in the following, we denote POIs by P_k with varying index. A sample of POIs in London, UK, as well as New York City, US, and their respective closeness relations are visualized in Figures 11.2.

As in Subsection 11.4.1 (and as presented in [135]), we rely on a probabilistic model to counter the ambiguity inherent in the data source. Similar to before, for each occurrence of a particular closeness relation, we create a two-dimensional spatial feature vector $v = (r, \phi)$ where r denotes the distance and ϕ denote the orientation. Now, however, distance and orientation are not relative to a fixed POI but relative to the two POIs which stand in relation to each other. For instance, assume $(P_i, \text{"next to"}, P_j)$ is a triplet mined from the text corpus, then v_{ij} describes Euclidean distance between P_i and P_j and the orientation

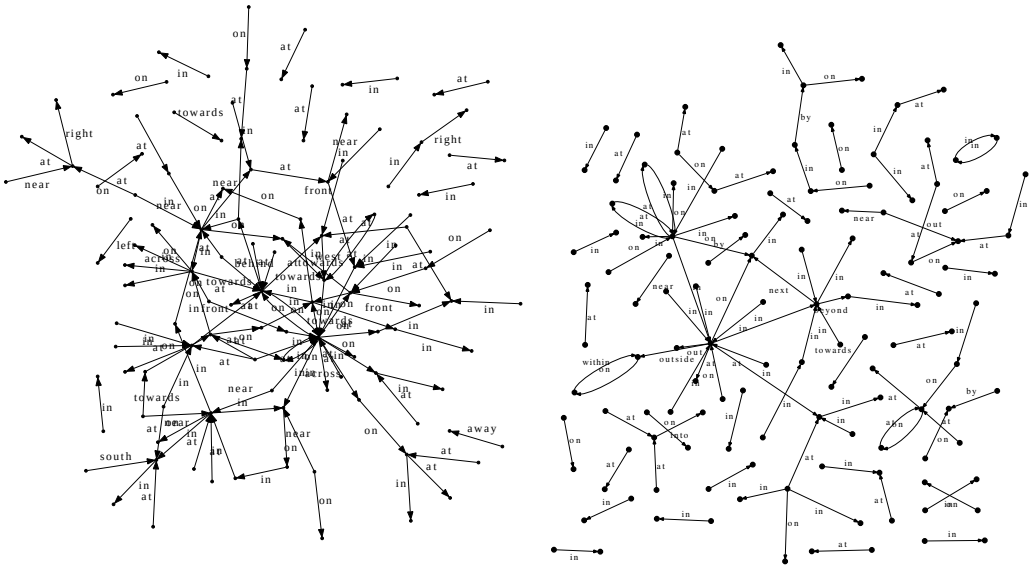


Figure 11.2: POIs (nodes) and their relations (edges) extracted from travel blogs. Visualized are two samples from London, UK, (left) and New York City, US.

as the counterclockwise rotation of the x-axis, centered at P_j, to P_i.[10] Again, we obtain a set of two-dimensional spatial feature vectors V_{REL^k} for each of the closeness relations REL^k. These will in turn be used to train two-dimensional Gaussian Mixture Models, yielding a set of probabilistic models $\mathcal{P}(\cdot \mid \theta^k)$, one for each closeness relation $REL^k \in \{REL^1, \ldots, REL^m\}$, the set of all closeness relations we take into account. For a pair of POIs P_i and P_j with spatial feature vector v_{ij}, $\mathcal{P}_{\theta^k}(v_{ij})$ is the probability that P_i and P_j stand in spatial relation REL^k to each other. Based on this information, we now derive a closeness score for pairs of POIs by Bayesian inference.

For two distinct POIs, let REL_{ij} denote the set of all closeness relations existing between P_i and P_j. Note that REL^k denotes an abstract relation, while REL_{ij} denotes the set of occurrences of relations between a pair of POIs. Furthermore, let v_{ij} denote the spatial feature vector of P_i and P_j. In order to determine a numeric score describing the closeness of the POIs, we estimate the posterior probability of all closeness relations REL^k and accumulate them. The posterior probability of relation REL^k is given by

$$\mathcal{P}(REL^k \mid v_{ij}) = \frac{\mathcal{P}(v_{ij} \mid \theta^k)\ \mathcal{P}(\theta^k)}{\sum_{l=1}^{m} \mathcal{P}(v_{ij} \mid \theta^l)\ \mathcal{P}(\theta^l)}$$

where $\mathcal{P}(\theta^k) = \mathcal{P}(REL^k)$ denotes the prior probability of relation REL^k given by the trained model represented by θ^k.

[10]Note that we rely on relations which are mostly conceived to be symmetric. Handling asymmetric relations such as "north of" is possible but would require separate handling.

In a traditional classification problem the task would be to choose the closeness relation with the highest posterior and to assign the pair of POIs to this class. We, in contrast, consider each posterior probability $\mathcal{P}(REL^k \mid v_{ij})$ as a measure of confidence of the existence of REL^k between P_i and P_j. Stressing that all considered relations represent spatial closeness, we combine all posteriors into one measure which we call *closeness score* cs_{ij} of the pair of POIs P_i and P_j:

$$cs_{ij} = \frac{1}{m} \sum_{k=1}^{m} \frac{\mathcal{P}(REL^k \mid v_{ij})}{\max\{\mathcal{P}(REL^k \mid v_{ij}) \mid \forall\, i \neq j\}}$$

From pure textual data in the form of travel blogs, we first mined triplets of toponyms linked by spatial closeness relations. The toponyms were mapped onto POIs (i.e., their locations and synonyms). For each pair of related POIs, we computed a spatial feature vector, and the entirety of these vectors was used to train GMMs, one for each relation. Finally, we computed a closeness score from the posterior probabilities of each relation given spatial feature vectors. This score reflects a confidence in the notion of closeness as reflected by the spatial relations and by the underlying data. Hence, this method can be used to evaluate occurrences of pairs of POIs in textual data w.r.t. their closeness. In the following, when speaking of a pairwise occurrence (of POIs), we mean a mined pair of POIs with non-zero or sufficiently significant score (greater than a given threshold).

11.4.3 Sequential Occurrences

Extending the above introduced concept, we now consider sequential occurrences of POIs having more than two elements. By the same logic as above, general sequences of POIs might bear additional information compared to single or pairwise occurrences. In particular, itineraries might be refined incorporating triples, quadruples or even longer sequences of POIs. In the use case of itinerary planning, combining two separate pairs of occurrences might yield unwanted categorical duplicates. Consider, for instance, one frequently visited pair of POIs where one POI is a restaurant and another frequently visited pair which also contains a restaurant. In this case, concatenating the two pairs will guide the user to two restaurants, possibly within a short time-span. When mining longer sequences of POIs, we gain valuable information about combinations of POIs which the crowd found to be valid. For example, longer sequences might also contain two restaurants. However, when mining such a sequence frequently, it implies a validation by the crowd, e.g., one restaurant might be a great lunch spot, the other a nice dinner place. Accumulating the significant sequences, we obtain a measure for the relevance of a certain combination of POIs. Clearly, this also holds for similar sequences in other types of crowdsourced data, like, for instance, Flickr photos. Mining sequential photos of Flickr users to extract knowledge about their movement patterns is an established approach in the research community [23], [9].

From a theoretical point of view, the process of knowledge extraction is similar to that described above. Consider, for example, consecutive Foursquare check-ins by one user during a day. If the user checks in at two places, the combination forms a pair, if they

check in at three places, a triple is formed. In this interpretation of the data, the two pairs contained in a triple need not necessarily form a pair themselves. Another way of looking at the data might be to imply that every relevant triple also generates two pairs. Besides consecutive Foursquare check-ins and Flickr pictures, one may also extract sequences of POIs from textual sources, for example from travel blog entries, following the approach introduced above.

The challenge is now, how to enrich the network with the obtained scores. Consider the following example: Assume you have mined and quantified the score of two POI pairs (P_i, P_j) and (P_j, P_k) and the score of their concatenation (P_i, P_j, P_k). The score of the triple might surpass the score of the pairs. This might be because relevant triples are not considered to generate pairs or simply because triples are scored higher than pairs . The question on how to use the quantified knowledge in order to enrich the underlying road network is tackled in the next section. As before, when speaking of a sequential occurrence (of POIs), we mean a mined sequence of POIs with non-zero or sufficiently significant score (greater than a given threshold).

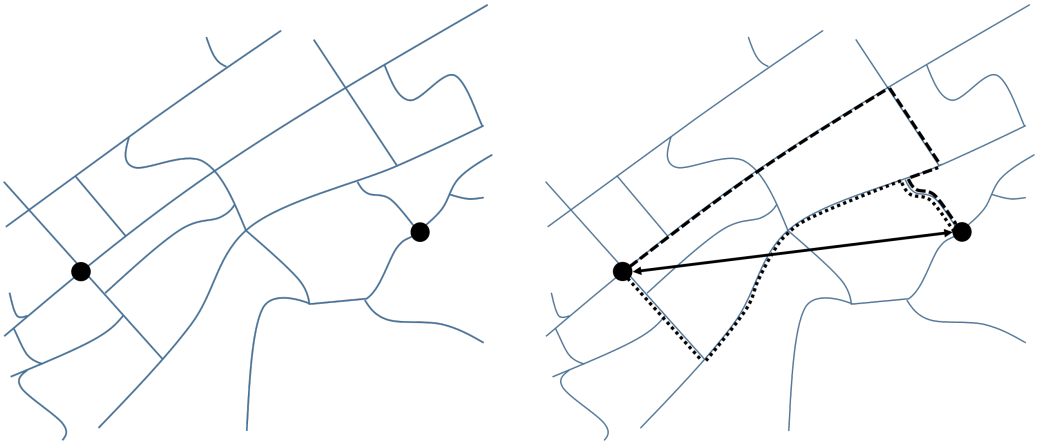
11.5 Enrichment

In this section, we present our approach to map the numerical score derived in Section 11.4 to the underlying road network. This section will show how the popularity scores are made applicable to routing algorithms.

We explore two different approaches: For the first approach, presented in Subsection 11.5.1, we pursue the idea that a gain in score corresponds to a reduction in cost, i.e., gain and cost are assumed to be reciprocal. Conventional routing algorithms expand paths with promising edges, i.e., edges with low cost values. By reducing the cost of edges with non-zero score, conventional routing algorithms favor these edges, steering the exploration in the direction of areas with non-zero scores. For the second approach, presented in Subsection 11.5.2, we introduce a meta-network where nodes correspond to POIs and edges correspond to shortest paths between them. By requiring subpaths to follow the connections between POIs, this allows for a greater restriction of the result paths, a stronger binding to the network layer containing the POIs.

A third approach is examined in Chapter 13 where a variation of the Arc Orienteering Problem (AOP) is presented. Given a travel time budget, the solution of an AOP is the path with maximum value among all paths not exceeding the budget. Hence, values are not perceived as reciprocal cost, making the problem NP-hard. Due to the different nature of the AOP, it is not discussed in this section. Let us note, however, that score functions, as derived in the previous section, may be employed as value functions in AOP instances.

The two approaches presented in this section utilize shortest path algorithms, including Dijkstra's algorithm [29], path skyline algorithms [80] and similar approaches, which we refer to as *conventional routing algorithms*.



(a) Part of a road network with two POIs (black circles).

(b) Illustration of two possible skyline paths connecting a pair of related POIs.

Figure 11.3: Example Road Network

11.5.1 Enriching the Road Network

We model any road network as a directed and weighted graph, based on OpenStreetMap[11] data, as illustrated in Figure 11.3(a). We denote the graph by $G = (V, E, c)$, where the vertices (or nodes) $v \in V$ correspond to crossroads, dead ends etc., and the edges $e \in E \subseteq V \times V$ represent streets connecting vertices. Furthermore, let $c : E \to \mathbb{R}_0^+$ denote the function which maps every edge onto its cost criterion. We introduce the following notations: $e_{uv} = (u, v)$ and $c_{uv} = c(e_{uv})$. If not stated otherwise, the cost criterion is distance. Other possible criteria are, for instance, travel time or energy consumption. If multiple cost criteria are used, they are denoted $c_1, \ldots, c_k$. Furthermore, let $\mathcal{P}$ denote the set of POIs. We assume every POI to be a node in the graph, i.e., $\mathcal{P} \subseteq V$. This is only a minor constraint as we can easily map each POI to the nearest node of the graph or introduce pseudo-nodes. Finally, a set of consecutive edges is referred to as path. Clearly, the notion of a cost criterion c extends naturally to any path p by defining $c(p)$ as the summed cost of its edges.

Single Occurrences

In Subsection 11.4.1 we describe the knowledge extraction process considering single occurrences of POIs in crowdsourced data. The output of this process is a normalized, node-associated score, i.e., POIs are assigned a numeric score in the range $[0, 1]$, describing some notion of value or popularity. 0 corresponds to POIs with no score, and 1 corresponds to the highest possible score. In order for this score to be used in conventional routing algorithms, it has to be offset against the underlying cost criterion. Otherwise, the optimal

[11] www.openstreetmap.org

path would be the solution to a traveling salesman problem visiting all POIs to acquire the maximum possible score. Hence, as mentioned before, we consider the score to be a reduction in cost. More precisely, in the following, we convert the node-associated score into an edge-associated cost.

Let $s(v)$ denote the score we associate with a node $v \in V$. Note that for $v \in V \setminus \mathcal{P}$: $s(v) = 0$, and for $P \in \mathcal{P} : s(P) \geq 0$. For each edge $(u, v) = e \in E$, we define the *score-discounted cost (sdc)* $sdc(e)$ as

$$sdc(e) := c(e) \cdot \phi^{\kappa(s(u)+s(v))} \tag{11.1}$$

where κ denotes a scaling parameter dependent on the data source and $\phi \in (0, 1)$ is a scaling factor for the influence of the respective score, similar to the approaches presented in [66, 68]. Intuitively, if e connects two vertices u and v with score 0, $sdc(e)$ equals $c(e)$. As the total score $s(u)+s(v)$ of nodes u and v increases, the score-discounted cost of e decreases exponentially. Thus, for an exceptionally large score of u and v, the adjusted cost $sdc(e)$ of edge e approaches zero. The parameters ϕ and κ control how quickly the score-discounted cost $sdc(e)$ converges to zero for an increasing aggregate score $s(u)+s(v)$ of the two vertices connected by e. For a larger parameter κ, the discount $sdc(e) - cost(e)$ increases more quickly, making the discounted edge e more attractive for conventional routing algorithms. A smaller parameter ϕ yields the same effect. We use κ to normalize the popularity score across different data sources, ϕ in contrast is a global scaling parameter.

If a given road network graph G is enriched with a score function *sdc* reflecting value or popularity of single occurrences of POIs, we refer to $G^1 = (V, E, sdc)$ as the *enriched (road network) graph*. Since each edge in G^1 is given the score-discounted cost $sdc(e)$, the notion of optimal paths will change given the new cost measure. This way, we incorporate the notion of crowdsourced popularity or value using traditional shortest path algorithms.

Pairwise Occurrences

Now, let us consider pairwise occurrences of POIs such as consecutive check-ins of a user of a location-based network. As mentioned before, the actual route the user chose between two check-ins is rarely recorded. Although there is a (spatial) connection between the check-in locations, it is unclear which part of the network should be enriched. Thus, we make the assumption that users generally prefer cost-optimal paths, i.e., paths which are optimal w.r.t. the given cost criterion or criteria, such as the shortest path, fastest path or paths in the path skyline. We propose to discount all edges along these paths, such that routing algorithms will prefer parts of the network which have been favored by the crowd according to the underlying data set. Consider Figure 11.3(b) where our approach to enriching the network with pairwise occurrences of POIs is exemplified. The set of optimal paths with respect to possibly multiple cost criteria is highlighted. Our approach decreases the cost of all edges located on any of the highlighted paths connecting the POIs. This approach is described in more detail in the following. First, we describe methods for selecting the paths to be enriched, then we describe our approach to enrich this set of paths in the road network.

If the underlying road network uses one cost criterion only, usually distance or travel time, we enrich the cost-optimal path connecting the pair of POIs. In multicriteria road networks, there are several possibilities of enrichment. A straightforward approach is to discount the cost-optimal paths according to each criterion, i.e., enriching up to d paths with crowdsourced knowledge when the network has d criteria. However, in order to increase the density of enrichment, we propose to enrich the path skyline or linear path skyline [80, 130, 127]. Depending on the number of cost criteria and on the distance between start and target node, the number of paths in the conventional skyline quickly deteriorates. Thus, if the number of cost criteria is relatively high (more than three) and/or the average extent between query end points is relatively large (> 100 km), it is recommendable to use the linear path skyline approach to restrict the influence of the enrichment. Once we have selected the set of paths to enrich, we proceed as follows.

Let (P_i, P_j) denote a pair of POIs mined from the data. For instance, two frequent consecutive check-in locations or two POIs which are connected by spatial closeness relations. According to one of the definitions above, we compute the set of (linear) skyline paths, denoted by $S_{i,j}$. Although the paths contained in $S_{i,j}$ differ from one another, they often share edges. We discount the cost of an edge only once, even if it occurs in more than one skyline path. Let $E_{i,j} \subset E$ denote the set of all distinct edges which are part of at least one (linear) skyline path from P_i to P_j. Then we define the score-discounted cost of cost criterion c_i of an edge as

$$sdc_i(e) = c_i(e) \cdot \prod_{e \in E_{i,j}} (1 - \phi^{s_{ij}})$$

As before, $\phi \in (0, 1)$ is a scaling factor. s_{ij} denotes the data set-specific score of the pair of POIs. For example, s_{ij} might be the number of consecutive check-ins at P_i and P_j normalized by the maximum number of check-ins at a pair of POIs. Or, $s_{ij} = cs_{ij}$ might be the closeness score defined in Subsection 11.4.2, an aggregate for spatial closeness relations extracted from text. Analogous to before, given a road network graph G, we define the enriched (road network) graph for a score s_{ij} reflecting connections between pairs of POIs as $\mathrm{G}^2 = (V, E, sdc_1, \ldots, sdc_k)$.

Discussion

All of the approaches of Subsection 11.5.1 modify the costs in the underlying road network. The cost of an edge is reduced if a relevant amount of qualitative information regarding that edge (or its immediate vicinity) has been found. By employing conventional routing algorithms which construct cost-optimal paths w.r.t. one or more cost criteria, cheaper edges will be favored over more expensive ones. More precisely, if edges e and f have the same cost but e has higher score or than f, then $sdc(e) < sdc(f)$. When mining single occurrences of POIs from crowdsourced data, this adequately reflects the local influence of the score of each POI on the cost of the adjacent edges. When considering pairs of POIs the cost of those edges is reduced which are part of some cost-optimal path connecting the pair. This, however, does not "force" routing algorithms to compute a path which actually visits both these POIs.

Algorithm 7 Using the meta-network for path computation

Input: G, G^2_{POI}, start s, target t
Output: Path connecting s and t

```
1 begin
2     P_entry = closest POI to s in G^2_POI
3     P_exit = closest POI to t in G^2_POI
4     (s, ..., P_entry) = cost-optimal path in G from s to P_entry
5     (P_exit, ..., t) = cost-optimal path in G from P_exit to t
6     (P_entry, ..., P_exit) = cost-optimal path in G^2_POI from P_entry to P_exit
7     (P_entry, ..., P_exit)' = mapping of (P_entry, ..., P_exit) onto the road network G
8     return concatenation of (s, ..., P_entry), (P_entry, ..., P_exit)', (P_exit, ..., t)
9 end
```

11.5.2 Creating a Meta-Network

To ensure that in such cases both POIs are indeed visited, we now introduce a meta-network. This meta-network also allows to consider sequential occurrences of POIs mined from the data. We refer to this meta-network as *POI graph* because the nodes of the graph correspond to POIs and the edges to paths connecting these POIs in the underlying road network.

Enrichment of Pairwise Occurrences

In the following, we build a POI-graph G^2_{POI} using pairwise occurrences. Each POI mentioned in any occurrence (or in significantly many) forms a vertex in G^2_{POI}. For any pairwise occurrence between two POIs P_i and P_j, a corresponding edge e_{ij} is added to G^2_{POI}. This edge holds a reference to the cost-optimal path (and its accumulated cost) connecting the pair. If the underlying road network is a multicriteria network, it is possible to precompute the cost-optimal path for each criterion. Figure 11.3(b) depicts an example: Here, a new edge is added as a shortcut between the two depicted POIs. The shortcut edge is not present in the underlying road network shown in Figure 11.3(a).

When issuing a query, the user inputs start and target nodes. Of course, these are not necessarily POIs, i.e., part of G^2_{POI}. Thus, we propose using both graphs, G and G^2_{POI}, in combination with a slight modification of Dijkstra's algorithm, as pseudo-coded in Algorithm 7. The idea is to find entry and exit POIs which are close to the start and target node, respectively. Subsequently, two paths are computed in G: the cost-optimal path from s to the entry POI and the cost-optimal path from the exit POI to t. Between entry and exit POI, routing is executed in G^2_{POI}, i.e., always following cost-optimal paths between pairs of POIs which are related according the crowdsourced information.

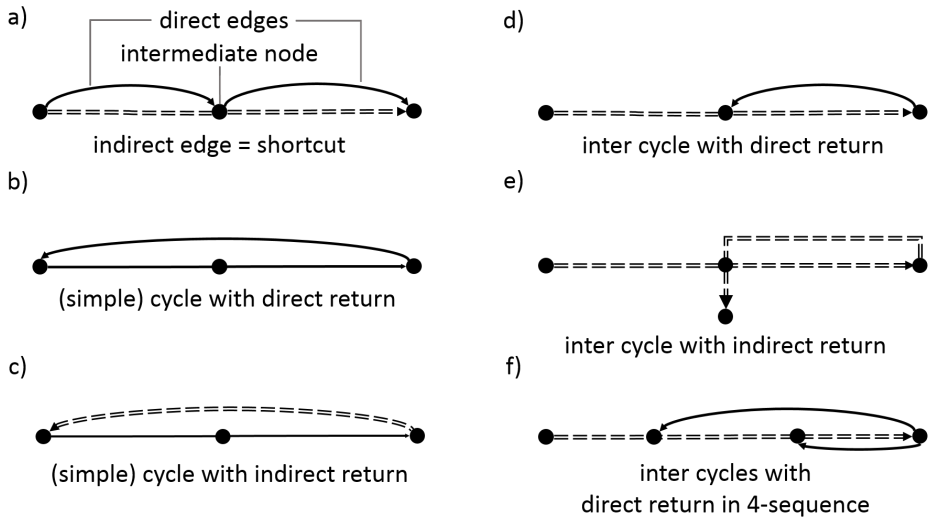


Figure 11.4: Illustration of the introduced terminology a), cycles with direct and indirect return b) and c) and of inter cycles with direct and indirect return d), e) and f).

Enrichment of Sequential Occurrences

So far, we only discussed the enrichment of single and pairwise occurrences. The reason is that in order to enrich the network with crowdsourced knowledge from sequential occurrences of POIs, a meta-network is needed. Assume that in one scenario only the two POI pairs (P_i, P_j) and (P_j, P_k) were mined from the data, and in another scenario the triple (P_i, P_j, P_k) was mined from the data. Thus, in the former case, users frequently travel from P_i to P_j, and from P_j to P_k, but users do not take the full journey from P_i to P_k (via P_j). In the latter case, the full journey was mined as a frequent sequence of POIs, and thus, P_j might be a simple stopover in-between two popular POIs P_i and P_k. By only enriching pairs of POIs, the distinction between these two scenarios is not possible. The information that visiting all three POIs in sequence is popular according to the crowd, is lost. Therefore, we extend the concept of the POI graph, first to triples of occurrences, then to arbitrary sequences.

We denote the extension of a POI graph $\mathrm{G}^2_{\mathrm{POI}}$ to triples of POIs as $\mathrm{G}^3_{\mathrm{POI}}$. Before, POIs which occurred in sufficiently many pairs according to the data formed a node. In $\mathrm{G}^3_{\mathrm{POI}}$, each POI forms a node which occurs in sufficiently many pairs or sufficiently many triples with greater score than its two consecutive pairs. Therefore, we only introduce edges for triples which bear greater score when visited in sequence than visiting its two pairs separately, i.e., $s(P_i, P_j, P_k) > s(P_i, P_j) + s(P_j, P_k)$. As before, each pair of POIs mined from the data is connected by an edge, also called *direct edge* for distinction. Additionally, for any triple (for which holds $s(P_i, P_j, P_k) > s(P_i, P_j) + s(P_j, P_k)$), we introduce a *indirect edge (or shortcut)* in $\mathrm{G}^3_{\mathrm{POI}}$ from P_i to P_k holding a reference to the concatenated cost-optimal path from P_i to P_j and from P_j to P_k. This is illustrated in Figure 11.4 where

the doubled lines represent shortcuts visiting triples of POIs and the single lines represent paths between POIs. We refer to the middle node of a shortcut, as seen in Figure 11.4a) as an *intermediate node*. In the following, we denote the direct edge from P_i to P_j by e_{ij} and the indirect edge (P_i, P_j, P_k) by e_{ijk}.

The idea of introducing shortcuts to model particularly valuable sequences of POIs is appealing. However, conventional routing algorithms may generate result paths with cycles. This is due to two reasons. First, intermediate nodes are not explicitly visited by a routing algorithm and can therefore not be flagged appropriately. Second, in order to promote the usage of sequences of POIs, we discount shortcuts which may lead to violations of the triangle inequality ($c_{ijk} < c_{ij} + c_{jk}$). Before we decide which cycles to avoid and how, we introduce definitions to distinguish different types of cycles.

Definition 11.1. *In a directed graph, a cycle is a path where no node is visited twice except for the start/end node. We distinguish between cycles where the start/end node is a conventional node or an intermediate node (of an indirect edge). We refer to the former as* (simple) cycles *and to the latter as* inter (intermediate) cycles. *Furthermore, we distinguish between cycles where the last edge, i.e., the "returning" edge, is a direct edge and where it is an indirect edge. We refer to these cycles as having* direct return *or* indirect return*, respectively.*

The possible occurrences of these cycles are depicted in Figure 11.4: Figures 11.4 b) and c) show simple cycles with direct an indirect return, respectively. Figures 11.4 d), e) and f) show inter cycles, where the cycle is closed at a node which is not explicitly visited by the routing algorithm, as it is "hidden" by a shortcut.

In the following, we examine which type of cycle may be part of a result path generated by a cost-minimizing routing algorithm and how this affects the result. We first consider simple cycles which require no handling at all, stated by the following lemma.

Lemma 11.1. *Simple cycles cannot occur in result paths.*

Proof. This lemma is a direct consequence of the Dijkstra property: When visited, every node is reached through the cost-optimal path. Formally, the cost of any path $p = (\dots, e_{i,j}, \dots, e_{i,k}, \dots)$ will never be less than that of $p' = (\dots, e_{i,k}, \dots)$ as all edge costs are non-negative, i.e., $c_{i,k} \leq c_{i,j} + \dots + c_{i,k}$. □

Next, consider inter cycles with direct and indirect return, as illustrated in Figures 11.4 d), e) and f). Let us first consider inter cycles with indirect return. Clearly, such cycles imply a detour. Yet, this detour may be compensated by the increased popularity incurred by a valuable sequence of POIs. The gain of visiting the sequence (P_i, P_j, P_k) may justify revisiting P_j, reflected in a sufficiently significant score-discounted cost. However, this is only valid for inter cycles with indirect return, not for those with direct return. This is because inter cycles with direct return offer no additional gain for revisiting a POI. Figuratively speaking, if the gain of the sequence has been collected, returning to a previously visited POI bears no gain. Hence, we want to ensure that inter cycles with direct return cannot occur when employing a routing algorithm on a POI graph with triples $\mathrm{G}^3_{\mathrm{POI}}$. If

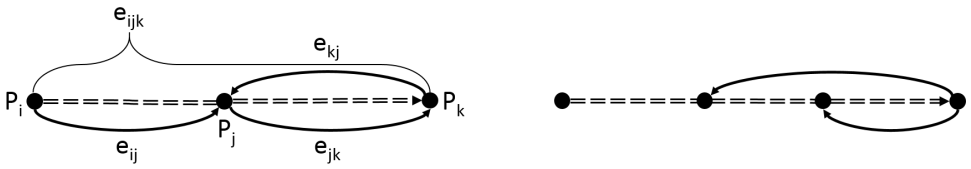


Figure 11.5: Illustration of an inter cycle with direct return in a triple of POIs (left). Visualization of both possible inter cycles with direct return in a 4-sequence of POIs.

$\mathrm{G}^3_{\mathrm{POI}}$ fulfills a specific requirement, we can prove this property. The left side of Figure 11.5 exemplifies the case and illustrates notation.

Lemma 11.2. *Let G^3_{POI} be a POI graph for pairs and triples of POIs. If for every indirect edge (P_i, P_j, P_k) holds:*

$$\delta_{ijk} < c_{jk} + c_{kj} \qquad (\star)$$

then no result path has inter cycles with direct return. δ denotes the additional discount of the POI triple over the concatenation of the two pairs, i.e.,

$$\delta_{ijk} := c_{ij} + c_{jk} - c_{ijk}$$

Proof. We first prove the statement for cycles of the form $(P_i, P_j, P_k), (P_k, P_j)$. Assume, there was an inter cycle with direct return:

$$c_{ijk} + c_{kj} \leq c_{ij}$$

By the property $(\star)$ we have:

$$\begin{aligned} c_{ij} &= c_{ij} + c_{jk} + c_{kj} - c_{jk} - c_{kj} \\ &\underset{(\star)}{<} c_{ij} + c_{jk} + c_{kj} - \delta_{ijk} \\ &= c_{ijk} + c_{kj} \end{aligned}$$

which is a contradiction to the assumption. Hence, there cannot exist an inter cycle with direct return in a result path. For any longer cycles

$$(P_i, P_j, P_k), \ldots, (P_k, P_j)$$

the statement holds because edge-costs are non-negative. □

Note that if one of the POI pairs is not connected, the property is fulfilled trivially. If (P_i, P_j) are not connected, the property is always fulfilled, if (P_k, P_j) are not connected, then no direct return to an intermediate node is possible. Let us note that in the data set for our experiments with triples of POIs, the property was rarely violated, and enforcing it bears minor computational overhead.

In the following, we extend the lemma to sequences of POIs. The problem with sequences of n POIs is the possible direct return to any of its $n-2$ intermediate node. For a sequence of four POIs this is illustrated on the right of Figure 11.5. Lemma 11.2 forbids the direct return to the last intermediate node, i.e., the $n-1$-st POI. In order to also forbid the other direct returns, we have to formulate similar conditions for all other intermediate nodes. With increasing length of sequences, however, it becomes less likely that cases occur where these conditions have to be enforced. Also, the conditions can be checked when constructing the POI graph, i.e., only once during a preprocessing phase of graph extraction.

For this purpose, we introduce the following new notation. Sequences of POIs will be indexed by numbers instead of letters. Additionally, by $\delta_{r,s},\ r > s$, we denote the difference in cost between the s-prefix sequence of an r-sequence. For instance, for a sequence of four POIs (P_1, P_2, P_3, P_4), $\delta_{4,2} = c_{12} - c_{1234}$ or $\delta_{4,3} = c_{123} - c_{1234}$. Generally, $\delta_{r,s} := c_{1,\dots,s} - c_{1,\dots,r}$. Note that these values are not necessarily positive. If they are, this implies a high discount of the full sequence compared to the prefix sequence. Obviously, since all edges, direct or indirect, have a non-negative cost, the cost of visiting additional POIs increases monotonically – despite any discount. Hence, the greater the interval between s and r, the less likely the value is positive. Bearing this in mind, we now extend the above statement inductively. This means, for any POI graph with sequences of POIs up to length r, we assume the statement for sequences up to length $r-1$ holds.

Lemma 11.3. *Let G^r_{POI} be a POI graph for sequences of POIs up to length r. If for every indirect edge $(P_1, \dots, P_r)$ the following statements hold*

$$\begin{aligned} \delta_{r,r-1} &< c_{r,r-1} \\ \delta_{r,r-2} &< c_{r,r-2} \\ &\vdots \\ \delta_{r,3} &< c_{r,3} \\ \delta_{r,2} &< c_{r,2} \end{aligned}$$

then no result path has inter cycles with direct return.

Proof. As before, it suffices to prove the statement for cycles of the form

$$(P_1, \dots, P_s, \dots, P_r), (P_r, P_s)$$

which directly follows from the above:

$$c_{1,\dots,r} + c_{r,s} > c_{1,\dots,s}$$

Any inter cycle with direct return to POI P_s has greater cost than the shortcut of length s, $e_{1,\dots,s}$. Like before, the cycle cannot be part of a result path as it has greater cost than the cycle-free counterpart. The argument holds analogously for longer cycles. □

In the following, we assume POI graphs for triples and longer sequences to fulfill the above properties. Therefore, it is ensured that we do not produce result paths with cycles (with direct return). For routing purposes, we may use Algorithm 7, replacing G^2_{POI} by G^3_{POI} or G^r_{POI} dependent on the POI graph at hand. For a given user query, i.e., start and target nodes, close entry and exit POIs are retrieved. From start to entry POI and from exit POI to target, the cost-optimal paths in the underlying road network are computed. Between entry and exit POIs, routing is executed in G^3_{POI} or G^r_{POI}, i.e., hopping along pairs, triples or longer sequences of POIs. Finally, the three subpaths are concatenated yielding a path from start to target.

11.6 Experimental Evaluation

In this section, we investigate the effect and impact of the network enrichment with crowdsourced data. We compare different data sources as described in Section 11.3, different means of extracting knowledge as described in Section 11.4 as well as different methods for enrichment as described in Section 11.5. To be able to draw comparisons between the different setups, we locate all our experiments in the city of Paris, France, which has high data density for all our sources. For road network data, we use OpenStreetMap. The road network extracted from the raw data has about 1M nodes and around 1.8M edges. All language processing was implemented in Python, the modeling of pairwise occurrences was implemented in Matlab. The tasks of network enrichment and path computation were conducted in the Java-based framework MARiO [58] on an Intel(R) Core(TM) i7-3770 CPU at 3.40GHz and 32 GB RAM running Linux (64 bit).

First, in Subsection 11.6.1, we explore the varying effects when using different data sources. For this, shortest paths in networks enriched with different data mined from single and pairwise occurrences are examined. More precisely, we compare shortest paths in different G^1 and G^2 graphs with each other. Second, in Subsection 11.6.2, we examine the value of routing in a meta-network compared to routing in an enriched road network graph. For this, paths generated in an enriched network G^2 are compared to those generated within a POI graph G^2_{POI}. Third, in Subsection 11.6.3, we substantiate the value of sequences of POIs over pairs of POIs. For this, paths generated in a POI graph for pairs G^2_{POI} are compared to those in a POI graph for triples G^3_{POI}. The roadmap for our experiments in summarized in Table 11.1.

For all experiments, we rely on three data sources from which we extract different notions of popularity. First, a Flickr data set, provided by authors of [101], consisting of 14M photos worldwide and over 40K photos in the metropolitan area of Paris, France. Second, a Foursquare data set, provided by the authors of [159, 160], consisting of 33M check-ins by over 250K users at more than 3.5M venues worldwide, thereof 7.6K venues in the area of Paris. Third, an extract from the aforementioned textual data set extracted from travel blogs. This extract contains 200 significant POIs and 2K occurrences of closeness relations (see Subsection 11.4.2) in the area of Paris. Using Twitter's public feed, we obtain around 200K tweets regarding these POIs. Which kind of knowledge is extracted

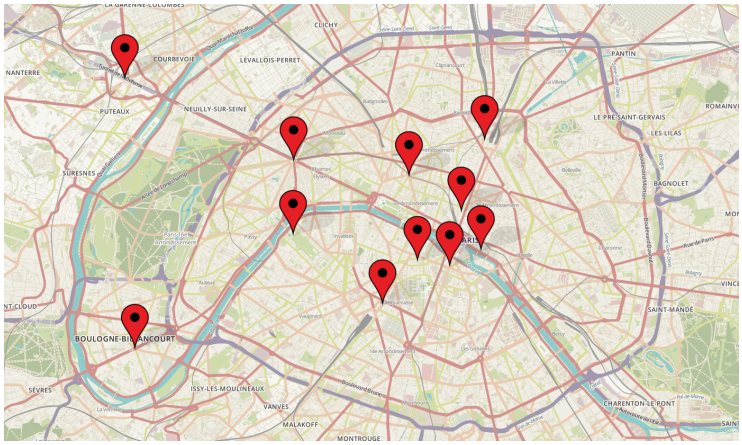


Figure 11.6: Visualization of the eleven hotspot centers in Paris, France.

and how the network is enriched with this knowledge, is described in the respective sections. Table 11.2 gives an overview which type of (POI) graph is derived from the different sources. Note that this selection is not exhaustive, for instance, one could use consecutive photos of Flickr users to derive scenic spots which are spatially connected.

Evidently, the density of the data sets varies considerably. Therefore we restrict ourselves to single, pairwise and triple occurrences of POIs and omit longer sequences. In order to obtain meaningful results, we empirically determine eleven hotspot regions in Paris where all data sets are particularly dense. A hotspot is a circular region with a 500 meter radius centered such that it contains significantly many data points of all sources. The hotspot centers are visualized in Figure 11.6. Covered by the entirety of the eleven hotspots are over 5K nodes of the underlying road network. For a query, two random nodes are drawn from this set as start and target. The results of each setting are grouped in two distance brackets corresponding to the Euclidean distance between start and target, 0 to 4 and 4 to 8 kilometers. For each setting, 6K runs are executed.

In the following, we compute paths in different graphs and compare them w.r.t. the respective scores they attain. We define three scores which are related to the scores used for enrichment in Subsection 11.5.1 but not identical. For a given path p, the absolute Flickr score S_{FLR} corresponds to the summed number of Flickr photos within a 40 meter radius of the course of p. Analogously, the absolute Foursquare score S_{FSQ} is the total number of check-ins at POIs within a 40 meter of the course of p. Finally, the absolute textual sentiment score S_{TXT+SA} is the total number positive mentions of all POIs within a 40 meter radius of the route of p. Again, Table 11.1 gives an overview of the measures used in the experiments.

	Data Sources Subsection 11.6.1		Networks Subsection 11.6.2		Sequences Subsection 11.6.3
	Single	Pair	Foursquare	Textual Closeness	
Graphs	G^1(FLR) G^1(FSQ) G^1(TXT+SA)	G^2(FSQ) G^2(TXT+CR)	G^2(FSQ) G^2_{POI}(FSQ)	G^2(TXT+CR) G^2_{POI}(TXT+CR)	G^2_{POI}(FSQ) G^3_{POI}(FSQ)
Measures	PL S_{FLR} S_{FSQ}	PL S_{TXT+SA} S_{FSQ}	PL S_{FSQ}	PL S_{TXT+SA}	PL S_{FSQ}

Table 11.1: Overview of the experiments conducted, the graphs used and the measures employed. PL stands for path length. Lengths and score values are always relative to those of the conventional shortest path (in G).

Graph type Number of occurrences	G^1 enriched network single	G^2 enriched network pairs	G^2_{POI} meta-network pairs	G^3_{POI} meta-network triples
Flickr (FLR)	✓	✗	✗	✗
Foursquare (FSQ)	✓	✓	✓	✓
Travel Blogs (TXT)	✗	✓	✓	✗

Table 11.2: This table shows which types of graph and POI graph are derived from the different sources used in this work.

11.6.1 Comparing Data Sources

In this part of our experimental evaluation, we illuminate the effects of enrichment with different data sources. We focus on data sources which reflect some notion of crowdsourced popularity with differing connotations. Enriching the road network with this kind of information, we aim to generate a graph and paths therein which better reflect the qualitative "mind of the crowd". Finding measures for evaluation of this enrichment, however, is not straightforward. Because, if there existed an absolute measure for popularity, quality or value, this work would be obsolete. Thus, we have to rely on the measure provided by the enrichment with one data source in order to evaluate the enrichment with other data sources. For instance, consider a single-criterion road network enriched with two notions of popularity, e.g., aesthetic appeal, as for example provided by the crowdsourced knowledge of Flickr photos, and (nightlife or culinary) trendiness, as for example provided by the crowdsourced knowledge of Foursquare check-ins. Given start and target, we may now compute three different cost-optimal paths. The cost-optimal path w.r.t. the network cost criterion and the two cost-optimal paths w.r.t. the score-discounted cost functions as described in Subsection 11.5.1. In lack of an objective measure for the qualitative information of our enriched network, we evaluate each path w.r.t. the score provided by the other data

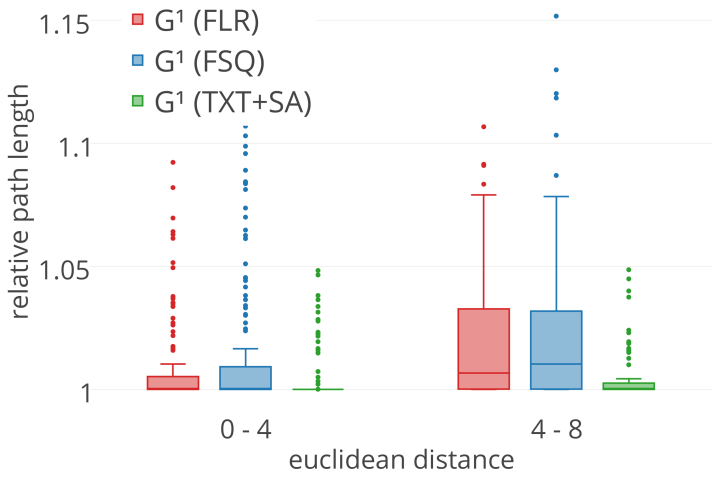


Figure 11.7: Path lenghts of optimal paths in graphs G^1(FLR), G^1(FSQ) and G^1(TXT+SA) relative to path lengths of shortest paths (in G).

source. For the optimal path reflecting aesthetic appeal, we compute the score reflecting trendiness and vice versa. We hope to observe an increase of popularity scores for any path computed within an enriched network, compared across different enrichment sources and, particularly, compared to the conventional cost-optimal path w.r.t. network criterion. We compare paths computed in enriched graphs G^1 and G^2.

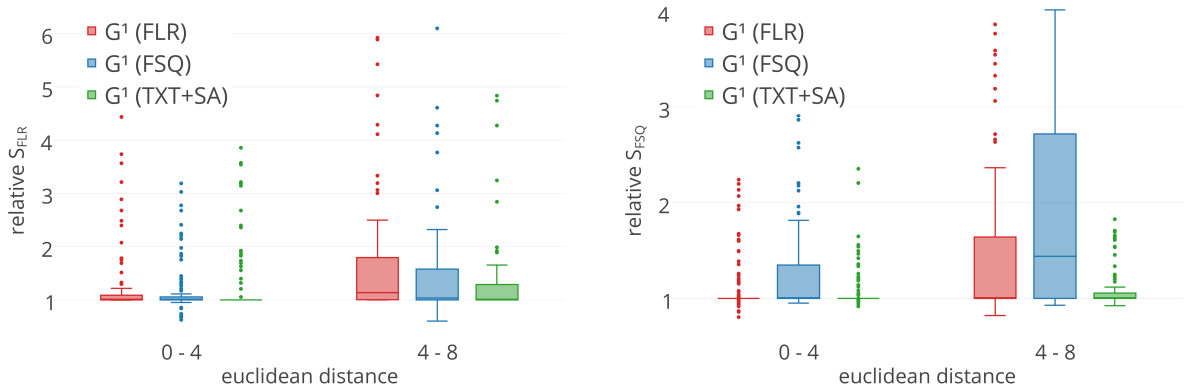
Single Occurrences

For single occurrences of POIs we rely on all three data sources: Flickr, Foursquare and the travel blog entries. From these data sources we derive three score-discounted cost functions and thus three enriched networks, as detailed in Subsection 11.4.1. We respectively denote the enriched networks by

$$G^1(\text{FLR}), G^1(\text{FSQ}), G^1(\text{TXT+SA}).$$

For each experimental setting, we choose one of the scores as evaluation measure. For each run of a setting we choose start and target as described above. For every start and target, we compute the cost-optimal path in all four networks using Dijkstra's algorithm, and for each of the four results, we compute the chosen score function. We group the resulting scores according to the Euclidean distance between start and target in two distance brackets.

The results are shown in Figure 11.6.1 for distance, in Figure 11.8(a) for Flickr score S_{FLR} and in Figure 11.8(b) for Foursquare score S_{FSQ}. The results for $S_{\text{TXT+SA}}$ show a similar behavior and are therefore omitted here. All results are relative to the conventional shortest path in G. The conventional shortest path serves as a baseline for distance as well



(a) S_{FLR} scores (b) S_{FSQ} scores

Figure 11.8: Scores of optimal paths in graphs $G^1(FLR), G^1(FSQ)$ and $G^1(TXT+SA)$ relative to scores of shortest paths (in G).

as for any score function. This is because the conventional shortest path reflects the score inherent in the query itself, the score attained "by chance".

Figure 11.6.1 displays how far the cost-optimal paths in enriched networks stray from the shortest path. Evidently, the increase in path length is marginal, rarely above 5%. The increase in path length is, of course, expected to provide an increase in score. We evaluate the paths generated in each graph w.r.t. to each of the scores. Considering Figure 11.8(a), unsurprisingly, the highest S_{FLR} is achieved by the optimal path in $G^1(FLR)$, analogously for S_{FSQ} and $G^1(FSQ)$ (see Figure 11.8(b)). Overall, the paths in the enriched networks score around 10% to 20% higher than the shortest path. For longer distances, considerable increase in scores is attained, while for the smaller distance bracket, 0 to 4 kilometers, the increase in scores is merely marginal. This emphasizes the importance of networks enriched with binary or sequential occurrences of POIs. As we will see later, scores are boosted when relying on non-single occurrences and POI graphs. Nevertheless, the results shown here can be understood as a proof of concept for our work: Optimal paths in networks which are enriched with crowdsourced data, indeed gain higher scores than simple shortest paths[12]. Also, optimal paths in a network enriched from one data source gain higher scores w.r.t. this source's score than paths in networks enriched with other sources. Furthermore, we observe a certain correlation between the different score values. An increase in score S_{FLR} implies an increase score S_{FSQ} and vice versa. The score S_{TXT+SA}, however, stays largely unaffected by the increase of both other scores. This could imply that the notion of popularity induced by Flickr and Foursquare data sets are rather dual to each other, while the popularity induced by the sentimentally analyzed travel blog mentions is rather orthogonal.

[12]We note that this effect is independent of the underlying cost criterion. The same holds for other hard metrics like travel time or energy consumption. However, for reasons of brevity, we omit this evaluation here.

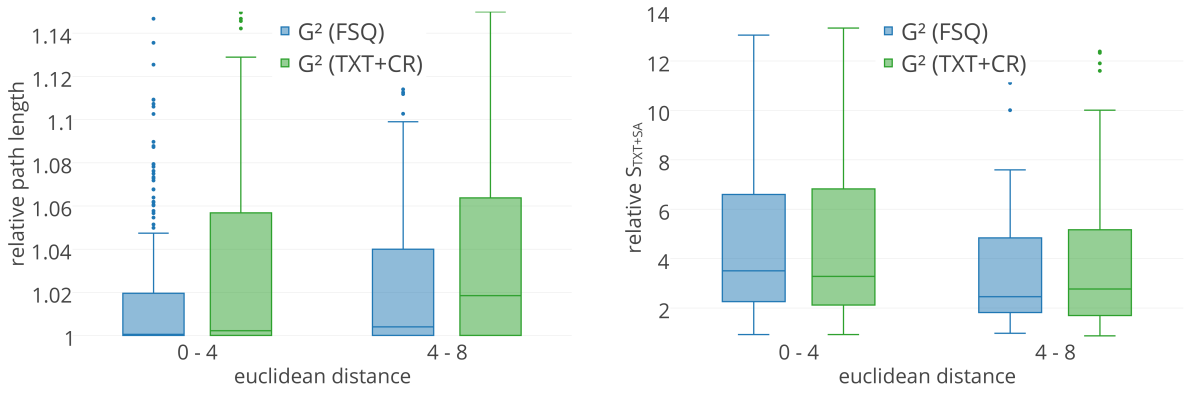
Occasionally, it is possible for optimal paths in enriched networks to undercut the score of the conventional shortest path. This is because the score of enrichment and the score of evaluation differ. For the enrichment, the density of data around nodes within a data source-specific radius is considered. For our experimental evaluation we chose a fixed radius of 40 meters, independent of the data source. Therefore, it is possible that the radius around the course of the shortest path "accidentally" attains a higher score.

Pairwise Occurrences

For pairwise occurrences of POIs we rely on two data sources: Foursquare and the travel blog entries. The knowledge extraction phase follows Subsection 11.4.2, for the enrichment we follow Subsection 11.5.1. Specifically, for each Foursquare user, we extract the pairs of consecutive check-ins, resulting in just under 28K pairwise occurrences. For the textual blog entries, we build a probabilistic model and compute the closeness scores based on the spatial closeness relations such as "next to" and "nearby", yielding around 2K significant pairwise occurrences. As detailed before, we may use this knowledge to either enrich the underlying road network directly or to construct a meta-network, the POI graph. For now, we focus on the comparison of different data sources. How the different network types perform, is evaluated in Subsection 11.6.2. Here, we restrict oursevles to the enriched networks $G^2(\text{FSQ}), G^2(\text{TXT+CR})$. The meta-networks $G^2_{\text{POI}}(\text{FSQ}), G^2_{\text{POI}}(\text{TXT+CR})$ are examined later. Dijkstra's algorithm is used to compute cost-optimal paths in the enriched networks.

Figures 11.9(a) and 11.9(b) show the results for networks enriched with pairwise occurrences of POIs, evaluated w.r.t. path lengths and scores. All results are relative to the conventional shortest paths in the road network G. Regarding path length, we observe only marginal increase, similar to the networks enriched with single occurrences. However, the increase in scores (here: $S_{\text{TXT+SA}}$) is significantly higher than before, around double to triple the score of the shortest path. Interestingly, the accumulated textual sentiment score $S_{\text{TXT+SA}}$ of the paths computed in the Foursquare graph $G^2(\text{FSQ})$ are on a par with those in the textual closeness graph $G^2(\text{TXT+CR})$. This can be attributed to the discrepancy between the score $S_{\text{TXT+SA}}$ and the enrichment of $G^2(\text{TXT+CR})$. While $S_{\text{TXT+SA}}$ scores positive textual mentions of POIs, the edges of $G^2(\text{TXT+CR})$ are score-discounted and therefore favored if they are part of shortest paths between "close" POIs. The similar score values therefore imply that the textual sentiment score, the notion of POI popularity mined from Tweets, is equally reflected in the enrichment with consecutive Foursquare check-ins and POI pairs which are close according to the crowd. From an application standpoint this can mean that $S_{\text{TXT+SA}}$ is insufficiently selective, i.e., it does not qualify well for demarcation. On the other hand, the effect might also be considered a benefit: The knowledge extracted from two different data sources is reflected in one score derived from another data source. Of course, further tests are required to substantiate either claim.

Overall, we may state that different data sources produce different paths. Which knowledge is mined from which data source is application-dependent, correlations occur, and implications are subject to expert supervision. We find that already for networks enriched



(a) Path lengths (b) S_{FSQ} scores

Figure 11.9: Path lengths and scores of optimal paths in enriched networks G^2(FSQ) and G^2(TXT+CR) relative to score of shortest path (in G).

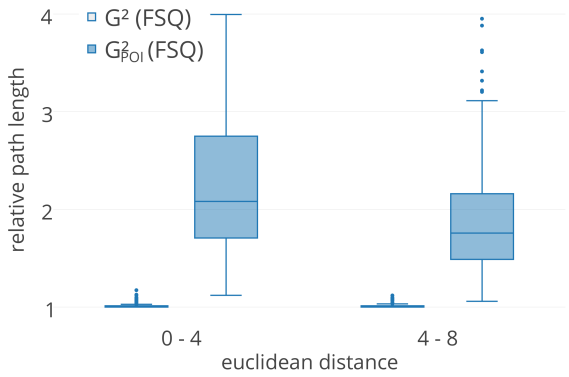
with single occurrences of POIs, there is an increase in scores at the cost of only marginal increase in length. As we will see, this effect can be amplified considerably when employing networks enriched with pairwise occurrences and, even stronger, when employing POI graphs.

11.6.2 Comparing Enriched Networks and Meta-Networks

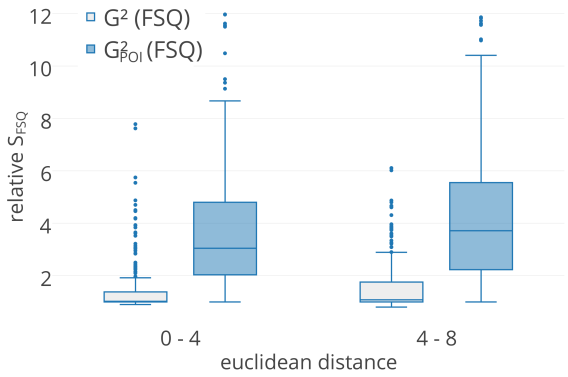
Having compared different data sources, we now compare the different network types. For this set of experiments, we use networks enriched with pairwise occurrences, i.e., G^2 versus G^2_{POI}. We rely on the same data sources as before for pairwise occurrences, consecutive Foursquare check-ins and textual POI mentions that stand in closeness relation to each other. From these sources, four graphs are derived, two enriched networks, G^2(FSQ), G^2(TXT+CR) and two meta-networks G^2_{POI}(FSQ), G^2_{POI}(TXT+CR).

Figures 11.10(a) and 11.10(b) show the results for the Foursquare graphs G^2(FSQ) and G^2_{POI}(FSQ) relative to path lengths and S_{FSQ} scores of the shortest path in G. Figures 11.10(c) and 11.10(d) show the results for the textual closeness graphs G^2(TXT+CR), G^2_{POI}(TXT+CR) relative to path lengths and S_{TXT+SA} of the shortest path. Note that when comparing data sources in Subsection 11.6.1, we evaluated paths generated in an enriched network (e.g., G^1(FLR)) with the scores induced by other data sources (e.g., S_{FSQ}, S_{TXT+SA}). Now, we compare the paths generated in an enriched network to those in a meta-network. Therefore, we evaluate the paths in one networks (e.g., G^2(FSQ), G^2_{POI}(FSQ)) with the score induced by the same data source (e.g., S_{FSQ}).

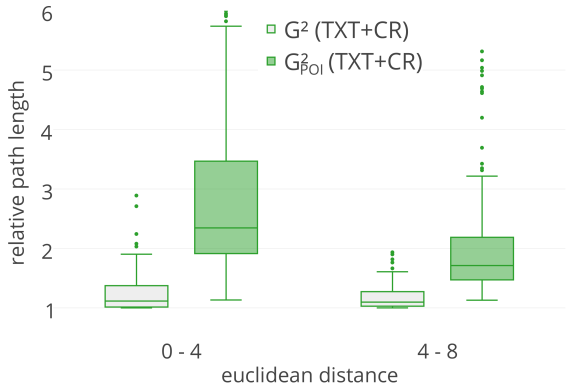
In terms of path length, both enriched networks, G^2(FSQ) and G^2(TXT+CR), create results only slightly longer than the shortest path. Of course, when routing in a POI graph, the paths increase considerably in length. This is due to path computation, which (recalling Algorithm 7) for the most part follows paths between POI pairs and therefore



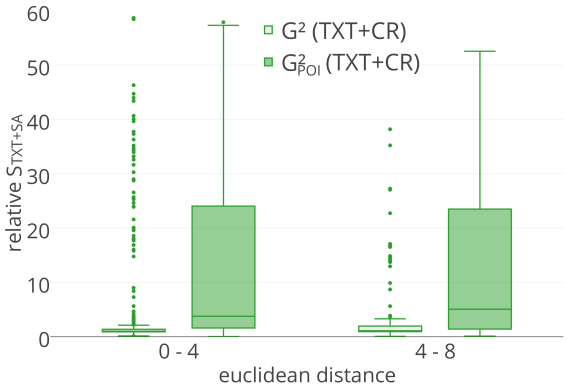
(a) Path lengths in G^2(FSQ) and G^2_{POI}(FSQ)



(b) S_{FSQ} scores in G^2(FSQ) and G^2_{POI}(FSQ)



(c) Path lenghts in G^2(TXT+CR) and G^2_{POI}(TXT+CR)

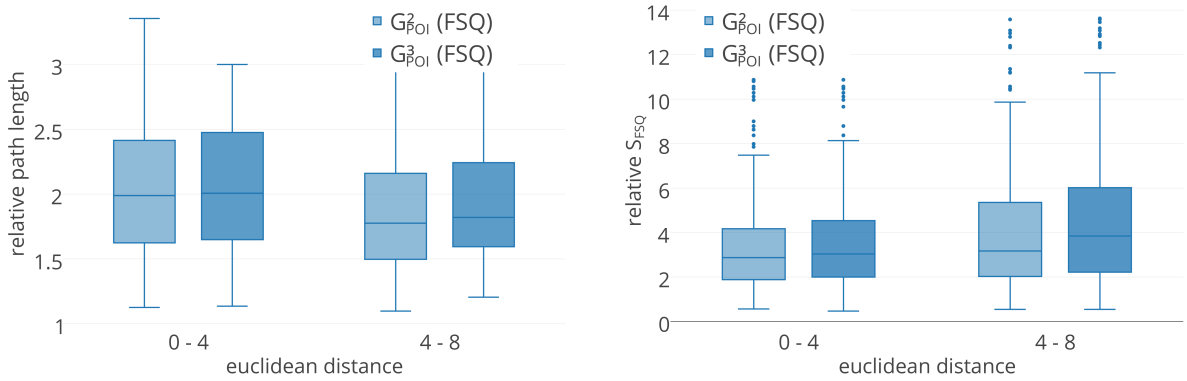


(d) S_{TXT+SA} scores in G^2(TXT+CR) and G^2_{POI}(TXT+CR)

Figure 11.10: Path lengths and scores of optimal paths in Foursquare graphs (top) and textual closeness graphs (bottom) relative to shortest path (in G).

enforces stronger binding to parts of the network which are scored higher. Hence, compared to the shortest path, we observe roughly doubled path lengths in G^2_{POI}(FSQ) and G^2_{POI}(TXT+CR). Interestingly, the increase in length is greater when start and target are closer to each other. This can be attributed to the density of the POI graphs which, compared to the road network, is low. Therefore, the detours which have to be made due to the structure of the POI graph carry less weight as the overall distance grows.

For the marginal increase in length using enriched networks G^2(FSQ) and G^2(TXT+CR), results attain considerably higher scores (see Figures 11.10(b) and 11.10(d)). While path lengths are almost the shortest path distance in G^2(FSQ), S_{FSQ} is around 30% higher than that of the shortest path. For G^2(TXT+CR), even more so. At a length increase of 20%, paths attain roughly double the S_{TXT+SA} score. For both data sources, the POI graphs intensify the effect. By doubling path length, three (G^2_{POI}(FSQ)) to five times (G^2_{POI}(TXT+CR)) the popularity scores of the shortest paths are attained. Of course,



(a) Path lengths (b) S_{FSQ} scores

Figure 11.11: Path lengths and scores of optimal paths in graphs G^2_{POI}(FSQ) and G^3_{POI}(FSQ) relative to shortest path (in G).

this effect is dependent on parametrization and data density, but the results substantiate the claim that routing in POI graphs serves the purpose of increasing data-specific scores. For any application where straying further from the shortest path is unproblematic, meta-networks may be employed. This may hold, for example, for tourist route recommendation systems where path length is a minor criterion. Also, in the case of sportive routing, e.g., for cyclists and runners, path length might be a limiting factor, but attractiveness of the path itself is probably the most important factor. When only minor increase in length is tolerated, then enriched networks yield the better trade-off. For negligible additional path length, considerable gain in score is achieved. This may be of beneficial for car navigation systems. For drivers, travel time is usually the highest ranked criterion. However, some drivers might be willing to accept a minor detour for a more scenic trip, i.e., knowledge that may be extracted from crowdsourced data like Flickr photos, for instance.

11.6.3 Comparing Pairs and Triples

For our final setting, we examine the benefit of enriching meta-networks with sequential POI knowledge. We rely on consecutive Foursquare check-ins for this experiment, i.e. the POI graphs G^2_{POI}(FSQ) and G^3_{POI}(FSQ). As mentioned before, we were able to extract just under 28K pairwise occurrences, i.e., consecutive check-ins pair by the same user. Extending the sequence of consecutive check-ins to triples, we were able to mine 14K triple occurrences. Combining pairs and triples of POIs, we may build a POI graph G^3_{POI}(FSQ) which, in addition to the links between POI pairs, holds shortcuts for relevant occurrences of POI triples. As mentioned before, Algorithm 7 may equally be applied to POI graphs for sequences. The structure of the graph, however, can lead to cycles during path computation. In Subsection 11.5.2 we detailed which kinds of cycles in result paths are problematic. We explained which are to be avoided (indirect cycles with direct return) and how to en-

sure this by asserting that a property related to the cost of edges holds. As established in Lemma 11.2, if $c_{ijk} + c_{kj} > c_{ij}(\star)$ holds for any triple of POIs P_i, P_j, P_k, then no indirect cycles with direct return can occur. Ensuring this property and therefore correctness is simple. For any extracted triple of POIs P_i, P_j, P_k where the corresponding pairs P_i, P_j and P_k, P_j exist, it is checked whether $(\star)$ holds. If $(\star)$ does not hold, then c_{ijk} is increased such that the inequality holds. This means, the reduced cost of the triple (P_i, P_j, P_k) is increased again. Building $\mathrm{G}^3_{\mathrm{POI}}(\mathrm{FSQ})$, we encountered violations of this property in a negligible number of triples, less than 100. Hence, ensuring correctness in this sense is easily enforced prior to query time and no limiting constraint.

Figure 11.11(a) shows the path lengths, Figure 11.11(b) shows the Foursquare scores $\mathrm{S}_{\mathrm{FSQ}}$. Additionally using triples of POIs has hardly any increasing effect on path lengths. This can be attributed to presence of links between pairs as well as shortcuts in $\mathrm{G}^3_{\mathrm{POI}}(\mathrm{FSQ})$. If a shortcut is not beneficial, i.e., its trade-off between length and score is suboptimal, the routing algorithm may often also choose the path corresponding to one of the pairs. Thus, the POI graph for triples offers additional knowledge without degrading the knowledge extracted from pairwise occurrences. The increase in scores of paths generated in $\mathrm{G}^3_{\mathrm{POI}}(\mathrm{FSQ})$ compared to those in $\mathrm{G}^2_{\mathrm{POI}}(\mathrm{FSQ})$ is not as strong as between $\mathrm{G}^2_{\mathrm{POI}}(\mathrm{FSQ})$ and $\mathrm{G}^2(\mathrm{FSQ})$. Nevertheless, at the cost of virtually no increase in path length, we are able to attain 15% to 30% higher scores. This emphasizes the value of POI sequences for routing purposes. Seeing as the overhead in programming is low, it can be recommended to integrate this kind of knowledge if it may be extracted from the corresponding data source.

Chapter 12

Tourismo: A Demonstration

12.1 Introduction

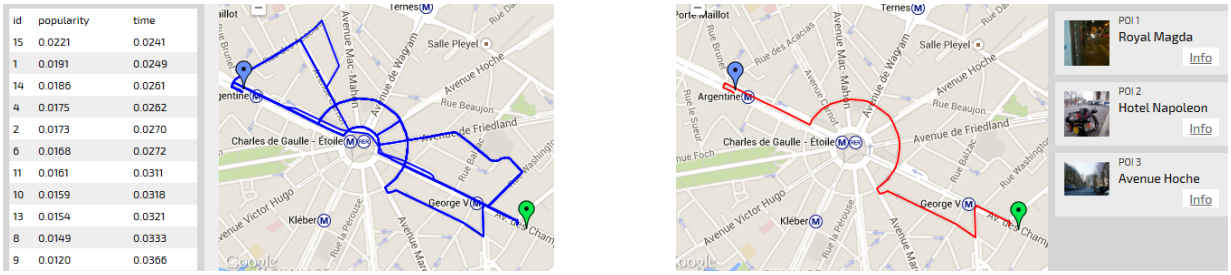
The core question of Tourismo is the same as before: How to measure the subjective concept of quality? How to generate popular, attractive, scenic, recreational routing suggestions? As machines are not (yet) capable to reflect this concept, we rely on the crowd to answer this question. We propose to use crowdsourced data to estimate the appeal of areas, streets or POIs. In particular, this demonstration paper relies on three kinds of crowdsourced data, as presented in Subsections 11.3.1 and 11.3.2. For reasons of comparability, it focuses on notions of popularity mined from single occurrences of POIs and applying the extracted knowledge to routing algorithms.

The data sets have partially been evaluated in Subsection 11.6. As before, Flickr data and textual travel blog data linked with twitter sentiments are employed (yielding G^1(FLR) and G^1(TXT+SA)). Additionally, trajectory data from Endomondo[1] is taken into account. In conclusion, the original road network is enriched with different knowledge reflecting the following notions of popularity:

- if the density of Flickr images in the vicinity of a POI is high, this indicates touristic and/or scenic appeal, increasing its *image popularity* score;
- if a POI, which has been mined from travel blog entries, is positively mentioned in Twitter's feed, this indicates popularity according to the crowd, increasing its *textual popularity* score;
- if a graph edge is part of an exercise route chosen by the users of Endomondo (mostly runners and cyclists), this indicates recreational popularity, increasing its *trajectory popularity*.

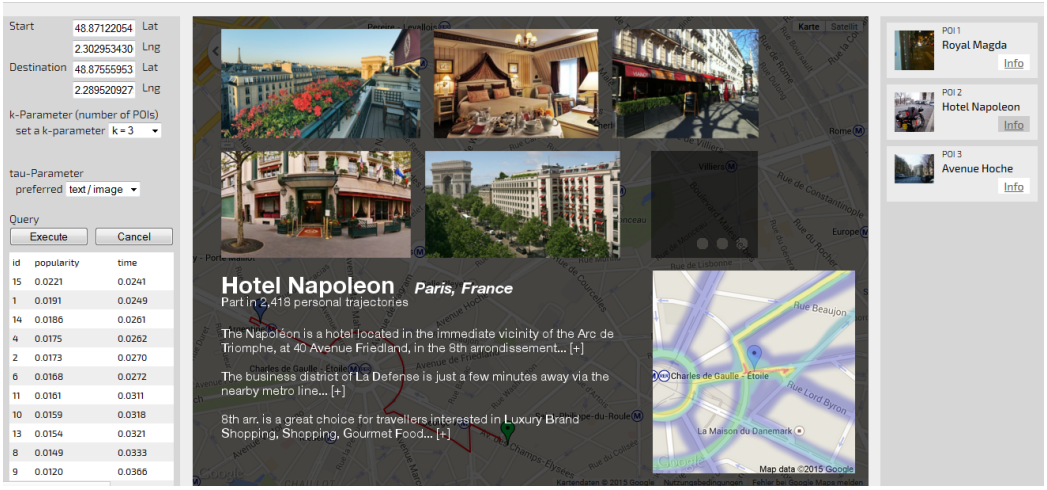
Furthermore, we incorporate meta-information from OpenStreetMap (OSM) and categorize POIs, allowing the user to specify particular categories if desired. *Tourismo* provides

[1] www.endomondo.com



(a) Particular Path Skyline

(b) Detailed Path Information



(c) Detailed information about selected POI

Figure 12.1: Functionality of the presented framework.

the opportunity to validate that the notions of popularity derived from the data indeed coincide with the intuition.

12.2 Framework Description

The main feature of this demo is the estimation of popularity from text, image, and trajectory data. Details covering text and image data can be found in Subsection 11.4.1. Let us now briefly describe the underlying road network is enriched using historical trajectory data. We rely on trajectories of walkers, runners and cyclists that have uploaded their exercise routes to Endomondo. Our data set contains eight million trajectories, which are located all around the world, but have a strong regional focus in Northern Europe. To match each of the GPS trajectories, we apply state-of-the-art map matching techniques, similar to those presented in [104]. In a first step, we perform a basic enrichment: For each edge e of the spatial network, we count the number $\mathrm{tra}(e)$ of historical trajectories that contain this edge. Following the assumption that runners, cyclists and walkers are, on average, likely to choose an appealing exercise route, this indicates popularity according

to the crowd of athletes. The respective popularity scores are subsequently combined with the expected travel times, deriving three different cost criteria, as presented in Subsection 11.5.1. This yields an enriched network with three cost criteria where conventional routing algorithms, which aim at minimizing edge costs, may be applied. *Tourismo* implements [128] and [125] which compute the path skyline and linear path skyline, respectively, as presented in Chapters 6 and 8.

Upon selecting an origin and a destination location in the city of Paris, France, the user is presented with the skyline view as shown in Figure 12.1(a) where all skyline paths are drawn on the map. For each skyline path, the corresponding cost values are shown in a table in the lower left corner of Figure 12.1(a). Using this table, the user may browse the result and select a desired path. This brings the user to the path view shown in Figure 12.1(b). For the selected path, this view shows the POIs on the respective path with the highest popularity score. Once a POI is selected, the Flickr pictures in the immediate vicinity of the POI are displayed, as shown in Figure 12.1(c). The bottom left corner shows current meta-information about the POI queried from web services. The bottom right corner shows a heatmap derived from all trajectories that pass the particular POI.

Additionally, *Tourismo* features category-specific path queries. If the user chooses to specify his personal touristic interests, they can choose one or more options from a list containing outdoor activities, cultural sightseeing, culinary interest and more. In order to provide paths which fulfill these requirements, we mine OSM meta-information. Thanks to a very active community, the data contains well-tended information about POIs, that is named, categorized, and subcategorized. For instance, the categories “food” and “tourist” contain subcategories “bar”, “restaurant”, “fastfood” and “monument”, “museum”, “archeological”, respectively. Mapping these categories onto the preferences, we filter POIs which correspond to the particular interest of the user. When querying a path with a specific set of interests, the user is provided a number of pareto-optimal paths, guiding him along POIs tailored to his preference.

Chapter 13

An Extension to the Arc Orienteering Problem

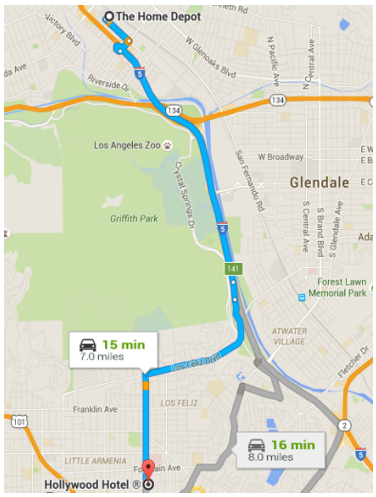
13.1 Introduction

Not long ago, finding the shortest or fastest path in road networks was the core challenge of route planning. Nowadays, challenges are manifold – due to the abundance of sensor data, due to the existence of increasingly complex traffic models and due to the demand for efficient yet convenient solutions. Modern navigation systems often take multiple, sometimes time-dependent, cost criteria into account and solve complex queries in order to increase driver convenience and traffic efficiency. In view of this progress, we propose a novel query with multiple applications. As an extension to the family of Orienteering Problems, we present the Twofold Time-Dependent Arc Orienteering Problem (2TD-AOP).

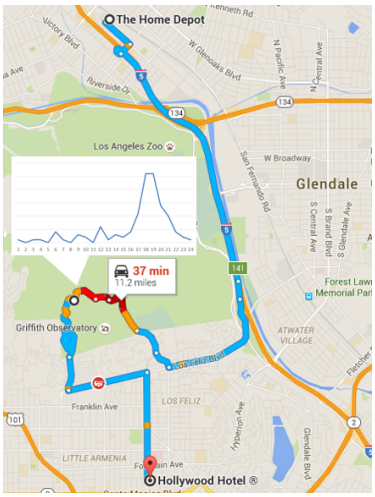
In its original formulation [56], the Orienteering Problem (OP) requires to find a path from a given source to a given destination abiding by a given cost budget but maximizing the value collected along the way. In most cases, the cost function corresponds to the travel time in the network. The problem may for instance occur in the field of spatial crowdsourcing [76, 17, 146] where individuals complete advertised tasks in order to collect rewards. A recent popular example of this is the augmented reality game Pokémon Go. A worker who is willing to complete crowdsourced tasks will want to maximize the collected rewards within his chosen time frame. The OP combines the NP-hard Knapsack and Traveling Salesman Problems, hence is NP-hard itself. Numerous variations of the problem have been proposed, for instance, extending it to multiple workers (Team Orienteering Problem) [59] or restricting the collectible values to certain time windows ((Team) Orienteering Problem with Time Windows) [47]. Another variation is the Arc Orienteering Problem (AOP) [3]. In the OP, the value is associated with the vertices of the graph, whereas in the AOP, it is associated with the arcs[1]. Thus, the OP intuitively corresponds to logistic

[1]In line with the terminology of the problem and its related work, throughout this chapter, we refer to nodes as vertices and edges as arcs. Furthermore, we use the terms source and destination for start and target.

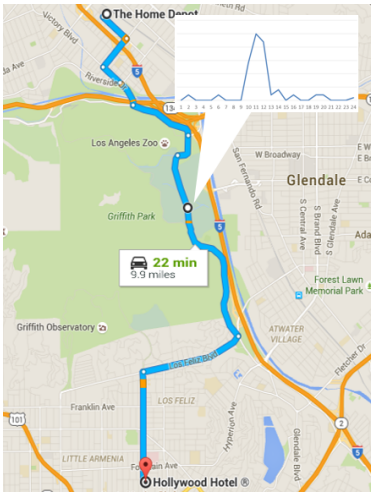
or crowdsourcing tasks, where the value is collected at particular points. The AOP, on the other hand, corresponds to tasks where the value is collected "along the way". Examples of such tasks are the routes of firefighting planes or the planning of scenic bike trips [139].



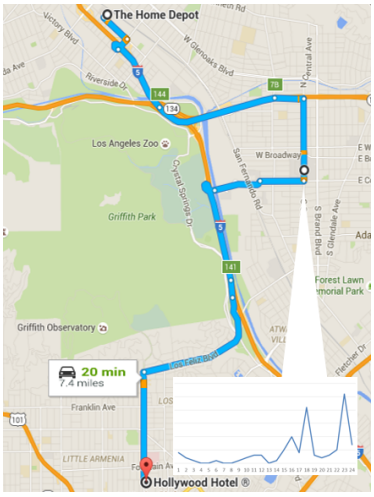
(a) Fastest path



(b) Static scenic path



(c) Time-dependent scenic path (11:00 am)



(d) Time-dependent scenic path (06:00 pm)

Figure 13.1: Exemplary paths with hourly value distribution for particular areas along the path.

Until recently, the AOP has only been answered for networks with static travel times. However, assuming static costs is not reasonable for most real-world scenarios. For road networks, the significance of time-dependence was empirically proven in [27] and substantiated by the incorporation into many navigation services such as Waze[2]. In a recent study,

[2]waze.com

Verbeeck et. al [150] first introduced time-dependent travel times but the values have been assumed to be static. In this chapter, we propose to incorporate time-dependent values in addition to time-dependent travel times, introducing the 2TD-AOP. As will be shown, results benefit greatly from time-dependent values which model the varying benefit for different times. Without loss of generality, we restrict ourselves to the application of scenic route planning. In this use case, a view of the sea might only be worthwhile during day, whereas a view of the skyline might be particularly worthwhile at night. Thus, the value of a coastline road is high during the day but low at night, while a road passing through downtown may have the opposite values. Other use cases include the aforementioned firefighting planes, collecting street view data or gathering Pokémon Go items.

Consider the example given in Figure 13.1 which shows different result paths for a scenic path query with the same source and destination. Figure 13.1(a) shows the time-dependent fastest path according to our algorithm for two departure times. While the path is the same at 11 am and 6 pm, the travel times are not the same (15 and 20 minutes, respectively). In this example, the fastest path does not pass through particularly scenic areas, according to the crowdsourced data set. The other paths pass through such areas, and for particular scenic parts of the respective paths, the hourly value distributions are highlighted. At first glance it is evident that the value functions show great variance over the course of a day. If not allowing for time-dependent values, these functions are commonly averaged, evening out the significant peaks of the distribution.

Suppose a driver is willing to spend 25 minutes for his trip from Hollywood Hotel to the Home Depot north of Griffith Park. Employing an algorithm for the AOP in static networks[90], we obtain the *static scenic path* displayed in Figure 13.1(b). The path was computed in a static network derived from our time-dependent network. The static travel time along an arc is set as the average of all travel times along this arc in the time-dependent network. The static scenicness values are derived analogously. The generated result path has a static travel time of 22 minutes which abides by the budget. However, the actual travel time might of course deviate in either direction. For a desired departure at 11 am, the time-dependent travel time coincides with the static time of 22 minutes, but for a desired departure at 6 pm, the time-dependent path takes 37 minutes. Thus, at 6 pm the result path clearly exceeds the budget and is therefore invalid. In a similar way, the averaged scenicness values can deviate from the time-dependent ones. While the static scenic path at 11 am does not exceed the budget, it is not optimal either. Griffith Observatory, which it passes along the way, is a popular sight but particularly scenic during sunset or in the evening. Instead, the best solution at 11 am is displayed in Figure 13.1(c). This *time-dependent scenic path* takes 20 minutes and passes a golf course, offering particularly nice views at daytime. The best solution at 6 pm is shown in Figure 13.1(d). This path takes 22 minutes and passes through central Glendale, a bustling area in the evening and at night.

Taking time-dependence into account when computing AOP paths is crucial. Not only result quality but also result validity depends on it. Our experiments show that every second static solution in a moderately complex settings is invalid. Users benefit considerably from time-dependent travel times as every 2TD-AOP path is valid. Furthermore, results are improved significantly when the values reflect time-dependence as well. Our ex-

periments show that 2TD-AOP paths generate three to four times more value than static solutions. We therefore propose to employ value distribution models whenever possible.

Due to their NP-hardness, static variations of the OP and AOP are usually solved employing metaheuristics. Allowing for time-dependent travel times adds further complexity to the task. Slight modifications of valid solutions may invalidate them. For instance, when taking a minor detour at the beginning of the path, travel times for the rest of the path may change and exceed the budget. Additionally incorporating time-dependent values multiplies this effect. Taking a minor detour at the beginning of a path may result in a change of value and travel time. We propose to solve the 2TD-AOP with a novel dynamic programming approach which makes use of pruning techniques to limit the candidate set and reduce intermediate path computations.

To the best of our knowledge, no previous work has allowed for both travel times and values to be time-dependent. In addition to handling the increased complexity, we require fast response times. We focus on the practicality of the solution where fast response times are crucial and large road network are standard, for instance, in order to meet the requirements of a mobile application. In conclusion, the contributions of this chapter are as follows:

1. We motivate and introduce the 2TD-AOP, as a new graph-based routing problem and a Mixed Integer Programming formulation.

2. We present an efficient approximate solution to the 2TD-AOP.

3. We present experiments on a large-scale real-world road network. Our evaluation is the first of this dimension for any time-dependent AOP. We show the value of twofold time-dependence empirically and compare 2TD-AOP solutions to static solutions as well as to optimal results.

The remainder of this chapter is organized as follows. In Section 13.2 we give an overview of research relevant to our work. In Section 13.3 the 2TD-AOP is formally defined and formulated as an Mixed Integer Programming. In Section 13.4 we lay the theoretical foundation for our algorithm which is described in Sections 13.5 and 13.6. Subsequently, our algorithm is evaluated in Section 13.7.

13.2 Related Work

We divide research relevant to this work into three groups. First, we review research on how to attain profit values from data in general and crowdsourced content in particular. Next, we briefly introduce related research from the database community such as the Trip Planning Query (TPQ). These problems have a similar field of application but are not congruent to the problem discussed in this chapter. Finally, we present research on the family of OPs which mostly stems from the field of Operations Research. We give an overview of solutions to the static and time-dependent OP and AOP.

13.2.1 Attaining Value Functions

Qualitative information, for instance, mined from crowdsourced data as in Chapter 11 may be utilized as value in the OP and its variations. The score functions derived in Section 11.4 may directly be applied to the static versions of the problem. In order to reflect time-dependence, the mining process has to be amended. For the experiments in this chapter, we employ an approach presented in Section 11.4.1. We use Flickr data to infer scenicness of arcs in the area of Los Angeles, CA. More details are given in Section 13.7. As emphasized in Chapter 11, relying on crowdsourced data for qualitative information facilitates scalability. In contrast, Most works from the field of Operations Research rely on specific bench mark data sets, some synthetic, others real. The authors of [139], for instance, evaluate their approach to cycle trip planning on real-world tourist data from Flanders, Belgium. The authors of [60] use data from Chinese theme parks. Relying on specifically recorded data like the above has two major drawbacks. First, expert knowledge or an official mandate may be required to attain the data. Second, the data may not be available globally.

13.2.2 TPQ and Related Queries

The most related query in the database community is the TPQ [19] (alternatively, Route Planning Query [14] or Route Search Query [83]). In this problem setting, the user specifies a different POI categories, e.g., “ATM”, “restaurant”, “florist”, “cinema”. The result of a TPQ is the fastest path from a given source to a given destination visiting exactly one instance of each resource. The TPQ is NP-hard because in the case that each category occurs exactly once, the TPQ degenerates to the Traveling Salesman Problem. There exist several variations, for instance introducing order constraints [78, 72] or modeling the fulfillment at a location probabilistically or time-dependent [73, 67, 25]. Solutions to this problem include full enumeration [25], heuristics [14, 78, 25], or backtracking and branch-and-bound approaches [67]. Partially time-dependent variations of these queries exist, however, most of the research pertains to static networks. Most importantly, albeit related, the problem settings of the OP/AOP and the TPQ are not congruent.

13.2.3 OP and AOP

Blueprint for the OP was the family of orienteering sports where efficient navigation from checkpoint to checkpoint is the goal [56]. Alternative names for the OP include the Selective Traveling Salesman Problem [81] or the Traveling Salesman Problem with Profits [36]. The OP is executed on a graph whose arc weights represent traversal costs and whose vertices may yield certain value. Given a source, a destination and a cost budget, the output of an OP is the path with maximum value among all paths not exceeding the budget. In the variation referred to as AOP or Privatized Rural Postman Problem [2], the profit is not assigned to vertices but to arcs, while the rest is as before. Extensive surveys on these problems and their numerous variants are given in [149, 59, 3] (or from the perspective of

the related Tourist Trip Design Problem in [47]).

Although NP-hard [56], exact solutions to the OP exist [81, 117, 36]. Most solutions, however, are approximative. While in older research specific heuristic approaches were proposed [147, 57, 12], the trend shifted towards algorithms following particular meta-heuristics such as Tabu Search [51], Genetic Algorithms [144] or Ant Colony Algorithms [86]. This holds similarly for the AOP where exact algorithms [24, 151] are rare but available. Generating approximative solutions is the standard approach [49], most effectively following the meta-heuristics Iterative Local Search (ILS) [151] and Greedy Randomized Adaptive Search Procedure (GRASP) [139, 90]. Recently, ILS and GRASP have been improved to solve the AOP on large scale real-world road networks in near-interactive time [90].

Introduced in [39], the OP with time-dependent (TD-OP) costs has gained attention recently [60, 45, 48, 150, 85]. Typical applications for the TD-OP include electronic tourist guides [45, 44] or routing in theme parks [60]. The time-dependent AOP (TD-AOP) as presented in [150] is not directly linked to an application but a time-dependent extension of the authors' cycle trip planning application [151] is imaginable.

The above works are all evaluated on small graphs (with the exception of [90]) with no more than a couple of hundred vertices. For most arcs in these graphs holds that they connect two non-zero value vertices. For instance, in the theme park data set of [60], vertices correspond to attractions and the arcs are fastest paths between them. The travel times and the values are derived from user data of the theme parks and their attractions. Similarly, [48] and the demonstration paper [44] are evaluated on data sets consisting of Points of Interest (POIs) and fastest paths connecting these POIs (by walking or by public transportation). When considering real-world road networks, this is not feasible. A moderate-sized network usually contains over one million vertices and two to three million arcs. In a time-dependent network, there usually exist numerous fastest paths from a given vertex to another, depending on the departure time. Hence, introducing shortcuts for precomputed fastest paths as in the approaches above is not possible. This is particularly the case, if the travel times are not statistically inferred but real-time, i.e., depending on live information, as is the case in the network of our evaluation.

The most closely related problem to the 2TD-AOP is the TD-AOP with time-dependent costs and static value. Verbeeck et. al recently proposed to solve this problem with an Ant Colony System algorithm [150]. Evaluated on specific benchmark instances with at most 100 vertices, the proposed approach finds near-optimal to optimal solutions within an average runtime of one second. The complex solver has excessive time requirements ("often more than 100 hours" [150]) to generate optimal solutions on these benchmark instances of the TD-AOP. In comparison, we increase complexity of the problem in two ways. First, by including time-dependent value functions 2TD-AOP adds another dimension of time-dependence to the problem. Second, we aim to solve the problem on large-scale real-world road networks with hundred thousands of vertices. Established solutions to related problems like the TD-AOP or the TD-OP rely on metaheuristics such as the Ant Colony System [150], Iterative Local Search [151] or Greedy Randomized Adaptive Search Procedure [139, 90]). Instead, we propose a heuristic dynamic programming approach.

13.3 Problem Definition

We model the road network as a graph $G = (V, A, val, tt)$, where V denotes the set of vertices (or vertices), $A \subseteq V \times V$ denotes the set of directed arcs (or edges), $val : V \times V \times \mathbb{R}_0^+ \to \mathbb{R}_0^+$ and tt (with the same domains) denote the non-negative functions mapping each arc onto its respective value and the travel time, respectively

$$val_{i,j}(t) := val(v_i, v_j, t) \geq 0, \quad tt_{i,j}(t) := tt(v_i, v_j, t) \geq 0$$

For the representation of the time-dependent travel time along an arc, we use a piecewise constant function with equal step width τ. In the following, we refer to the intervals where travel time is constant as time windows and denote the k-th time window by τ_k, where $0 \leq k \leq K$ for some maximum time window K.

A consecutive set of arcs which visits no arc twice is called a path. A path from source to destination departing at t_0 is denoted by $p = ((v_{p_0}, v_{p_1}), (v_{p_1}, v_{p_2}), \ldots, (v_{p_{N-1}}, v_{p_N}), t_0)$ where v_{p_0} is the source vertex and v_{p_N} the destination vertex. In the remainder of this chapter, we simplify notation for the sake of clarity. We omit the double indexes and denote a path p by

$$p = ((v_0, v_1), (v_1, v_2), \ldots, (v_{N-1}, v_N), t_0)$$

For each query, we denote source and destination by v_0 and v_N, respectively. The travel time of a path p is defined as $tt(p) = \sum_{i=0,\ldots,N-1} tt_{i,i+1}(t_i)$, where t_i denotes the time of arrival at (and departure from) vertex v_i. t_i is dependent on the travel times along the preceding arcs and is defined iteratively:

$$t_i := \sum_{j=0}^{i-1} tt_{j,j+1}(t_j)$$

The value of a path p is defined as:

$$val(p) = \sum_{i=0,\ldots,N-1} val_{i,i+1}(t_i)$$

For a given source v_0, destination v_N, and departure time t_0, we denote the set of all paths from v_0 to v_N starting at t_0 by $\mathcal{P}_{0,N}(t_0)$.

The input for the 2TD-AOP is a graph G, a source vertex v_0, a destination vertex v_N, a departure time t_0, and a time budget b. When speaking of a query, we mean the set of these inputs. The optimal path to a 2TD-AOP query is the path maximizing the collected value among all feasible paths. Both notions are defined in the following.

Definition 13.1 (feasible path)**.** *Given a query, we call any path p departing from v_0 at t_0 and arriving at v_N no later than $t_0 + b$ (i.e., $tt(p) \leq b$) a* feasible path*.*

Definition 13.2 (optimal path)**.** *Given a query, among all feasible paths, we call one with maximum value an* optimal path*.*

Extending the work in [149, 150], we give a mixed integer programming (MIP) formulation of the problem. We introduce the following two decision variables: $p_{i,j,k}$ is a binary decision variable which equals 1 if the result path enters the arc (v_i, v_j) in time window τ_k and 0 if the arc is not traversed at all or entered in a different time window. $t_{i,j,k}$ is a continuous decision variable containing the time of departure at v_j when entering the arc (v_i, v_j) in time window τ_k. Again, it is 0 if the arc is not traversed at all or entered in a different time window.

$$\max \sum_{i \neq j} \sum_{k=0}^{K} p_{i,j,k} \cdot val_{i,j}(\tau_k) \tag{13.1a}$$

$$\text{subject to:} \quad \sum_{i \neq j} \sum_{k=0}^{K} p_{i,j,k} \cdot tt_{i,j}(\tau_k) \leq b \tag{13.1b}$$

$$\forall\ 1 \leq l \leq N-1 :$$

$$\sum_{i=0}^{N-1} \sum_{k=0}^{K} p_{i,l,k} = \sum_{j=1}^{N} \sum_{k=0}^{K} p_{l,j,k} \leq 1 \tag{13.1c}$$

$$\sum_{j=1}^{N} p_{0,j,0} = \sum_{i=0}^{N} \sum_{k=0}^{K} p_{i,N,k} = 1 \tag{13.1d}$$

$$\forall\ 1 \leq l \leq N-1 :$$

$$\sum_{i=0}^{N-1} \sum_{k=0}^{K} t_{i,l,k} + tt_{i,l}(\tau_k) = \sum_{j=1}^{N} \sum_{k=0}^{K} t_{l,j,k} \tag{13.1e}$$

$$p_{i,j,k} \in \{0,1\},\ \forall\ i,j,k \tag{13.1f}$$

$$0 \leq t_{i,j,k} \leq \tau_K,\ \forall\ i,j,k \tag{13.1g}$$

$$t_{0,j,0} = t_0,\ \forall\ j \tag{13.1h}$$

Equation 13.1a gives the objective function while Equation 13.1b gives the budget constraint. Note that due to the time-dependent nature of the problem, all time slots have to be taken into account when summing the collected value and cost. Remember, however, that $p_{i,j,k}$ is a binary decision variable which for each arc can at most once be equal to 1, as each vertex (and thereby also each arc) is at most visited once. This is expressed in Constraint 13.1c. Constraint 13.1d ensures that the result path starts at the source and ends at the destination. Next, Constraint 13.1e ensures that the departure time at each arc is the departure time at its predecessor plus the travel time within the corresponding time slot. Note that this condition implicitly forbids "waiting" at vertices. Finally, Constraints 13.1f and 13.1g determine the domain of the decision variables and Constraint 13.1h ensures that the path departs from the source at t_0.

13.4 Solving the 2TD-AOP

In this section, we present terminology and insights essential to our solution of the 2TD-AOP.

Definition 13.3 (earliest arrival)**.** *Given a graph G, a source v_0, a vertex v_i, and a departure time t_0, we define the* earliest arrival (time) *for reaching v_i from v_0 when departing at t_0, denoted by $ea_{0,i}$ (ea_i for short), as follows:*

$$ea_i := t_0 + \min_{p \in \mathcal{P}_{0,i}(t_0)} tt(p)$$

Definition 13.4 (latest departure)**.** *Given a graph G, a destination v_N, a vertex v_i, and a latest arrival time $t_0 + b$, we define the* latest departure (time) *for reaching v_N from v_0 no later than $t_0 + b$, denoted by $ld_{i,N}(t_0 + b)$ (ld_i for short), as follows:*

$$ld_i := t_0 + b - \min_{p \in \mathcal{P}_{i,N}(-\infty)} tt(p)$$

i.e., the latest departure time among all feasible paths w.r.t. v_i, v_N, and t.

Definition 13.5 (forward-reachable)**.** *Given a graph G, a source v_0, a departure time t_0, and a time budget b, we refer to the vertices reachable within the budget b from v_0 (when departing at t_0) as* forward-reachable *vertices, denoted by $FWR_0(b)$ (FWR for short if v_0 and b are clear from context).*

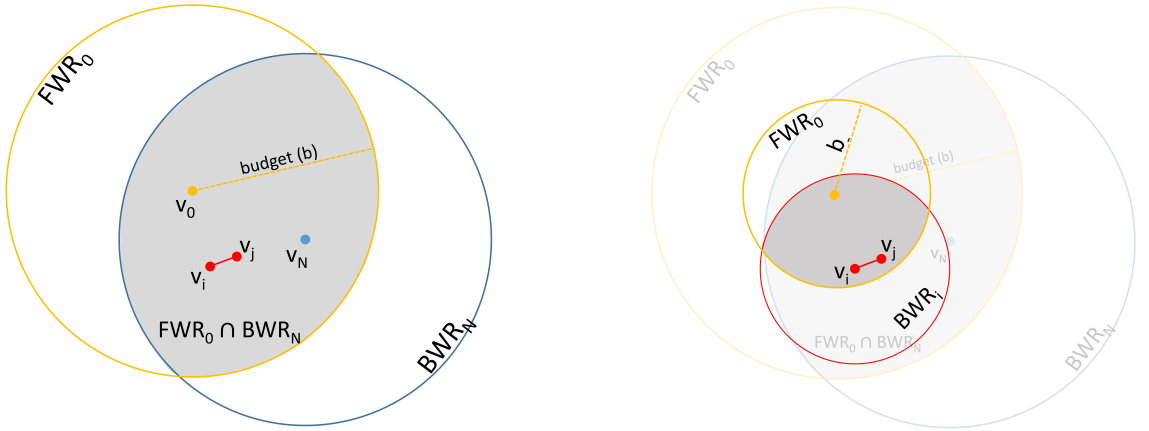
$$FWR_0(b) := \{v_i \in V \mid ea_i \leq t_0 + b\}$$

Definition 13.6 (backward-reachable)**.** *Given a graph G, a destination v_N, a departure time t_0, and a time budget b, we refer to the vertices from which the destination can be reached within the budget b when departing no earlier than t_0 as* backward-reachable *vertices, denoted by $BWR_N(b)$ (BWR for short).*

$$BWR_N(b) := \{v_i \in V \mid ld_i \geq t_0\}$$

Intuitively, all forward-reachable vertices lie in a quasi-circular region around the source. This region is similar to the circular expansion of a Dijkstra-search. The difference is that in this case the cost criterion is time-dependent travel time. Analogously, all backward-reachable vertices lie in a quasi-circular region around the destination. This graphical intuition is visualized in Figure 13.2(a). If the destination is not forward-reachable from the source (and/or vice versa), then the query has no answer.

Our solution to the 2TD-AOP is based on an extension of this observation: For an arc (v_m, v_n) to be part of a feasible path, it is a necessary condition that v_m is forward-reachable from the source v_0 and v_n is backward-reachable from the destination v_N. Hence, if an arc is not contained in the intersection of the forward-reachability region and the backward-reachability region, it cannot be part of a feasible path. This property will be proved later on. In our solution to the 2TD-AOP, arcs which increase the value are recursively inserted



(a) $\mathrm{FWR}_0(b)$ and $\mathrm{BWR}_N(b)$ and their intersection

(b) Recursive inclusion of the reachability regions and their intersection

Figure 13.2: Reachability regions as computed by FWR and BWR.

until the budget is exhausted. With every insertion of an arc, the overall travel time of the path increases. Hence, the reachability regions iteratively contract until no more arcs can be inserted. Before we detail this approach in the next section, let us explain the above definitions and their computation with an example.

The definitions are visualized in Figure 13.3. Figure 13.3(a) shows a graph with source v_0 and destination v_N. Departure time is $t_0 = 0$, and let the time budget $b = 4$. The time-dependent travel times along the arcs are visualized in Figure 13.3(b). The arcs in Figure 13.3(a) are labeled with letters which correspond to the travel time functions in Figure 13.3(b). Figure 13.3(c) shows the earliest arrivals and latest departures according to this query. If not indicated otherwise, vertices are forward-reachable as well as backward-reachable. The earliest arrivals are computed as in the definition. For example, $ea_1 = 0 + tt((v_0, v_1), 0) = 1$, implying that the fastest path from v_0 at $t_0 = 0$ arrives at v_1 at time 1.

Consider the following path:

$$((v_0, v_1), (v_1, v_2), (v_2, v_5), 0) = 1 + 1 + 3 = 5 > b$$

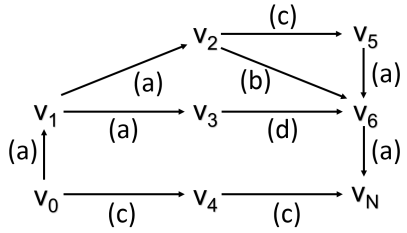
Since there is no shorter path from v_0 to v_5, v_5 is not FWR within the given budget. From v_0 there are two feasible paths to v_N:

$$((v_0, v_4), (v_4, v_N), 0) = 1 + 2 = 3$$

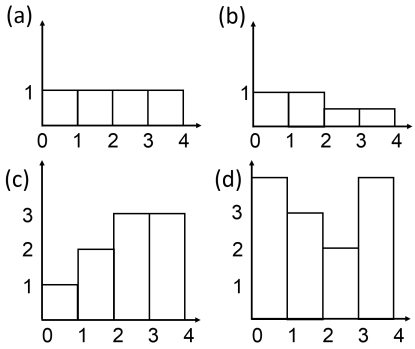
$$((v_0, v_1), (v_1, v_2), (v_2, v_6), (v_6, v_N), 0) = 1 + 1 + 0.5 + 1 = 3.5$$

Which of the two constitutes the optimal path depends on the time-dependent value functions and will be discussed later on.

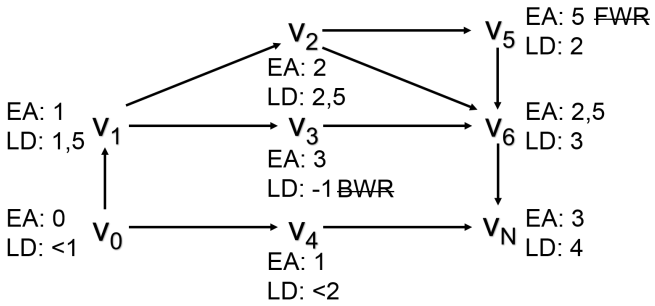
Let us turn to the computation of the latest departures. For example, $ld_6 = 0+4-1 = 3$, implying that in order to arrive at v_N no later than $t_0 + b$, one must depart at time 3.



(a) Example graph



(b) Example travel times functions



(c) Earliest arrivals and latest departures

Figure 13.3: Example graph for a query with source v_0, destination v_N, departure time 0 and time budget $b = 4$.

Now, consider ld_4: the only path to the destination follows the arc (v_4, v_N). Departing at time 3 results in exceeding the budget, since $tt_{4,N}(3) = 3$. The same holds for time 2. However, when departing at $t < 2$, then $tt_{4,N}(t) \leq 2$, resulting in an arrival within the budget b. Hence, in order to compute the latest departure time of a vertex v_i a linear scan among the time windows might be required. Note that the difference between "at time 2" and "before time 2" is infinitesimal. In theory we set the latest departure at v_4 to $2 - \varepsilon$, in practice, we adjust it by a sufficiently small number. Let us note that this special case only occurs if the latest departure at a vertex by chance coincides with the border between two time windows τ_k which is very unlikely in practice. Finally, we prove that in order to compute all feasible paths it suffices to consider the vertices which are forward-reachable and backward-reachable.

Lemma 13.1. *All vertices along all feasible paths are contained in the intersection of the sets of forward-reachable and backward-reachable vertices.*

Proof. Suppose there exists a feasible path p visiting a vertex $v_i \notin \text{FWR} \cap \text{BWR}$. Case 1: $v_i \notin \text{FWR}$, this implies $ea_i > t_0 + b$, meaning that the subpath from v_0 to v_i already exceeds

Algorithm 8 Computing forward-reachable vertices

Input: $G, v_0, t_0, t_0 + b$
Output: FWR

```
1  begin
2      Q ← empty queue of vertices v_i sorted asc. by ea_i
3      result = ∅
4      ea_0 = t_0
5      Q.add(v_0)
6      while Q is not empty do
7          v_i ← Q.removeFirst
8          for (v_i, v_j) ∈ G.A do
9              êa_j ← ea_i + tt_{i,j}(ea_i)
10             if êa_j < t_0 + b then
11                 if ¬ result.contains(v_j) then
12                     ea_j ← êa_j
13                     Q.add(v_j)
14                     result.add(v_j)  else
15                         if êa_j < ea_j then
16                             ea_j ← êa_j
17                         end
18                     end
19                 end
20             end
21         end
22     end
23     return result
24 end
```

the budget, which contradicts the assumption that p is feasible. Case 2: Analogously, if $v_i \notin \text{BWR}$, then $ld_i < t_0$, hence p cannot be a feasible path. □

13.5 Computing Reachability

Having clarified the importance of the sets of reachable vertices, we will now go into detail on their computation. Both sets are generated by expansions around source and destination with "time-radius b". By the above lemma, all vertices along all feasible paths are contained in the intersection of the two reachability regions.

In order to compute the set of forward-reachable vertices (for a given source v_0, departure time t_0, and a time budget b), we perform a modified time-dependent all-target Dijkstra search starting at v_0 at time t_0, called FWR and illustrated in Algorithm 8. All

earliest arrival times are initialized as ∞ except for the source v_0 for which it is 0. Additional to a result set containing all visited vertices, we maintain a priority queue containing vertices sorted in ascending order by the earliest arrival time of each vertex. In every iteration, the top element of the queue is removed. In the `for`-loop spanning lines 8-21, the outgoing arcs of the current vertex v_i are explored. In line 9, the travel time from v_i to its neighbor v_j is added to the earliest arrival time at v_i. If this value does not exceed the budget b, v_j is forward-reachable. If v_j was previously not visited, ea_j is set accordingly and v_j is added to the queue as well as the result set. If v_j was previously visited and the updated earliest arrival time is better than the previous, it is updated accordingly. Note that once a vertex v_i has been dequeued, the label ea_i is final. This is equivalent to the Dijkstra property which ensures that any processed vertex is reached by the shortest path, in this case the fastest time-dependent path. Finally, the result set contains all vertices reachable from the source within the budget.

The set of backward-reachable vertices is computed in a similar fashion but slightly more complicated. The procedure is presented in Algorithm 9 and called BWR. In addition to the result set containing all backward-reachable vertices, we maintain a queue of vertices sorted in descending order by ld_i. Initially, the latest departure at the destination v_N is set to the latest possible arrival time, $t_0 + b$. The algorithm operates backwards from the destination, following incoming arcs which are explored in the `for`-loop spanning lines 8-25. As was illustrated in the above example (Figure 13.3), in order to find the latest departure, it is required to linearly scan the time windows. When exploring an incoming arc from v_i to v_j, τ_k is initially set to the latest time window not exceeding the latest departure from v_j. If at τ_k the travel time from v_i to v_j is too high (i.e., v_j cannot be reached in time for its latest departure), k is decremented implying an earlier departure at v_i but possibly also a different travel time along the arc. Note the $[-\varepsilon]$ in line 13, corresponding to the aforementioned special case that $\hat{ld}_i = \tau_l$ for some l. In this case, $\hat{ld}_i$ is decreased by a small number in order to avoid computational errors which may occur at time window boundaries. Once a valid latest departure time for v_i is found, ld_i is set accordingly. If v_i was previously visited, ld_i is only set if it is later than the previous latest departure. Upon termination, all vertices which are backward-reachable from the destination within the given budget are in the result set and labeled with their respective latest departure.

13.6 Computing Solutions

Due to the NP-hardness of the problem, we forgo giving an optimal solution as it would not meet our requirements of providing solutions on large graphs in efficient time. Instead, we propose a heuristic solution to the 2TD-AOP. So far no other solution to the 2TD-AOP exists. Our dynamic programming solution recursively fills gaps with arcs in order to improve the value while not exceeding the budget. Upon initialization, the set of arcs `arcs` is empty, the set of gaps `gaps` contains (v_0, v_N), i.e., the gap given by the query. Assume that in the first iteration the arc (v_i, v_j) is inserted. Then `arcs` holds the new arc (v_i, v_j) and `gaps` holds two new gaps, (v_0, v_i) and (v_j, v_N). Into the new gaps, further arcs will be

Algorithm 9 Computing backward-reachable vertices

Input: $G, v_0, t_0, t_0 + b$
Output: BWR

```
1  begin
2      Q ← empty queue of vertices v_i sorted desc. by ld_i
3      result = ∅
4      ld_N = t_0 + b
5      Q.add(v_N)
6      while Q is not empty do
7          v_j ← Q.removeFirst
8          for (v_i, v_j) ∈ G.A do
9              τ_k ← arg max_{τ_l}{τ_l < ld_j}
10             while τ_k + tt_{i,j}(τ_k) > ld_j do
11                 k ← k − 1
12             end
13             l̂d_i ← ld_j − tt_{i,j}(τ_k) [−ε]
14             if l̂d_i ≥ t_0 then
15                 if ¬ result.contains(v_i) then
16                     ld_i ← l̂d_i
17                     Q.add(v_i)
18                     result.add(v_i)  else
19                         if l̂d_i > ld_i then
20                             ld_i ← l̂d_i
21                         end
22                     end
23                 end
24             end
25         end
26     end
27     return result
28 end
```

inserted. This procedure is repeated until the budget is exhausted. Hence, the 2TD-AOP is solved in a divide-and-conquer manner by recursive insertions. We therefore refer to the algorithm, pseudo-coded in Algorithm 10, as RecInsert.

We will now explain in detail how the algorithm proceeds. The two sorted lists `gaps` and `arcs` hold two-tuples of vertices. `gaps` holds pairs between which arcs are to be inserted, and `arcs` holds the converse information, i.e., the arcs already inserted. With every insertion, the budget is decreased by the duration of the detour that has to be taken to insert the new arc. However, this detour is not explicitly computed as a path. Instead,

Algorithm 10 Solving the 2TD-AOP

Input: G, `gaps`, `arcs`, b
Output: RecInsert

```
1  begin
2      Q ← empty queue of vertices v_i sorted desc. by ld_i
3      bestArc ← (null, null)
4      bestGap ← (null, null)
5      bestRatio ← 0
6      for (v_i, v_j) ∈ gaps do
7          FWR_i ← FWR(G, v_i, b)
8          BWR_j ← BWR(G, v_j, b)
9          G'_ij ← FWR_i ∩ BWR_j
10         for (v_m, v_n) in G'_ij: ea_m + tt_{m,n}(ea_m) < ld_n do
11             b_− ← ld_n − ea_m − tt_{m,n}(ea_m)
12             val_+ ← val_{m,n}(ea_m)
13             ratio ← val_+ · b_−^{-1}
14             if ratio > bestRatio then
15                 bestRatio = ratio
16                 bestArc = (v_m, v_n)
17                 bestGap = (v_i, v_j)
18             end
19         end
20     end
21     gaps.remove(bestGap)
22     if bestArc = null then
23         compute fastest path to close all gaps
24         return concatenation of fastest paths
25     end
26     else
27         gaps.add((v_i, v_m))
           // at former index of bestGap
28         gaps.add((v_n, v_j))
           // after (v_i, v_m)
29         arcs.add((v_i, v_j))
30     end
31     RecInsert(G, gaps, arcs, b_−)
32 end
```

it is implicitly derived from the information generated during FWR and BWR. The earliest arrivals and latest departures of every gap allow to decide whether an arc can be *feasibly inserted*, i.e., without exceeding the budget. This is crucial to the performance

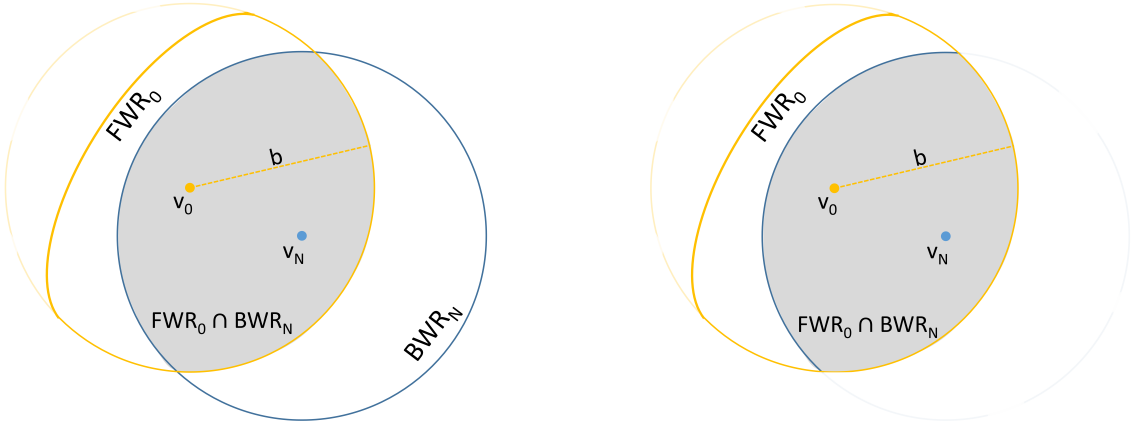
of our algorithm. Established metaheuristics usually compute, improve and perturb solutions. Starting from an initial path, promising arcs are (usually randomly) inserted and under certain conditions (usually randomly) deleted, to avoid local value maxima. Instead, RecInsert holds a sorted list of arcs which may be feasible inserted into a path. The actual paths between these arcs, however, are not computed until the budget is exhausted.

The arc which is to be inserted into a gap is chosen based on a heuristic. Among all arcs which can be feasibly inserted, the arc is selected which yields the best ratio of value to detour. This step is performed in the `for`-loop spanning lines 9 to 19. As established, all feasibly insertable arcs lie in the intersection of the reachability regions. The intersection is performed in line 8, therefore the set in line 9 comprises all arcs which can be feasibly inserted. In line 10, the detour incurred by inserting an arc is expressed as a decrease in budget. Assume that for the gap between v_i and v_j, the arc (v_m, v_n) is examined for insertion. ea_m is the earliest arrival at v_m coming from v_i, and ld_n is the latest departure from v_n to reach v_j "in time" (depending on the recursive budget b). Thus, $ld_n - ea_m - tt_{m,n}(ea_m)$ corresponds to the budget that would remain when taking the detour of traveling from v_i to v_m, then along (m, n) (at time ea_m) and from v_n to v_j. Hence, if inserting (v_m, v_n), the budget will be set to $ld_n - ea_m - tt_{m,n}(ea_m)$. In line 11, the gain in value is estimated. Note that the actual gain in value may be higher (due to other arcs along the path). However, in order to compute the actual gain in value, the fastest paths to and from each arc would have to be computed. Also, the value of all succeeding arcs would have to be reassessed. For the sake of efficiency, we therefore only estimate the value as proposed. The estimated gain in value divided by the decrease in budget yields the ratio by which we evaluate the arcs. The arc with the best ratio is inserted, i.e., the arc is added to `arcs` as a pair of vertices, and two pairs of vertices are added to `gaps` (the gaps left and right of the new arc). As long as at least one arc qualifies for insertion, RecInsert is recursively called with the extended sets `gaps` and `arcs` and the reduced budget $b\_$. If no arc can be inserted (into any of the gaps), the time-dependent fastest paths for all gaps are computed and the concatenation is returned which constitutes our approximate solution to the 2TD-AOP.

13.6.1 Pruning Strategies

Let us discuss three variations which are omitted in the algorithm description for reasons of brevity. (*i*), it is recommended to employ a simple forward estimation during FWR. (*ii*), it is recommended to compute the intersection of FWR_i and BWR_j during computation of BWR_j. (*iii*), it is possible to make the recursive call with a restricted portion of the graph. Variations (*i*) and (*ii*) are based on the following observation. The sets FWR_i and BWR_j are not needed separately, only their intersection is needed. Therefore, we may exclude vertices which are not contained in the intersection during computation of the respective sets.

Regarding (*i*): We propose to employ a simple forward estimation for pruning. If information about the speed limits is available, the following lower bound can be used



(a) FWEST (basic forward estimation)

(b) EALD (earliest arrival / latest departure)

Figure 13.4: Illustration of the effects of the proposed pruning techniques.

with negligible overhead. During FWR, in line 10, the `if`-clause may be tightened:

$$ea_j + \frac{ed_{jN}}{ms} < t_0 + b$$

where ed_{jN} is the Euclidean distance from v_j to the destination v_N and ms is maximum speed in the given network. If traveling from v_j to the destination at maximum speed exceeds the budget, so will any actual path. Any such vertex cannot be forward-reachable. Figure 13.4(a) illustrates the effect of this pruning strategy yielding a trimmed forward-reachability region. We refer to this pruning strategy as *FWEST* pruning. At marginal computational cost, FWEST yields a perceptible reduction in candidates. In our experiments, over 15% of vertices are pruned during FWR using this approach. Of course, the method holds analogously for BWR and can equally be applied. However, during BWR it is recommended to use the following pruning technique which is superior but exclusive to BWR.

Regarding (*ii*): One may use that for all vertices in the intersection $\text{FWR}_i \cap \text{BWR}_j$ holds $ea_i \leq ld_i$. This follows from the definition:

$$\begin{aligned} & ea_i > ld_i \\ \Leftrightarrow\ & t_0 + tt(p^\star_{0i}) > t_0 + b + tt(p^\star_{iN}) \\ \Leftrightarrow\ & tt(p^\star_{0i}) + tt(p^\star_{iN}) > b \end{aligned}$$

where $p^\star_{i,j}$ denotes the fastest path from v_i to v_j at ea_i. Hence, the `if`-clause in line 14 of BWR can be tightened to $ld_i \geq ea_i$. In this case, the simple forward estimation cannot be applied because

$$ea_i \geq t_0 \frac{ed_{0i}}{ms} \geq t_0$$

By construction, ea_i is based on an actual path expansion and therefore serves as a tighter bound than any estimation. We refer to this pruning strategy as EALD pruning. Figure 13.4(b) illustrates the effect of this pruning strategy, where the vertices which are not forward-reachable are not visited during BWR. Thus, instead of computing both reachability regions separately and subsequently their intersection, the information of FWR may be used in BWR. This limits the search space and implicitly computes the intersection. Every vertex in the intersection will be visited during BWR. Unvisited vertices are either not forward-reachable or backward-reachable. The effect of this pruning strategy is remarkable. Particularly when taking into account that it causes no computational overhead. When visiting a vertex during BWR, it suffices to retrieve its earliest arrival to decide if it qualifies for pruning.

Regarding (*iii*): It is possible but not necessarily recommended to issue the recursive call of RecInsert with a restricted portion of the graph. As established in Lemma 13.1, all vertices along feasible paths are in the intersection of the reachability regions. This property holds recursively. For instance, consider the insertion of a first arc (v_i, v_j) into the initial gap (v_0, v_N). In the next recursion, two gaps will be examined, (v_0, v_i) and (v_j, v_N), as illustrated in Figure 13.2(b). Consider the first gap (v_0, v_i), the respective reachability regions (forward from v_0 and backward from v_i) are contained in the original reachability regions (forward from v_0 and backward from v_N). Moreover, the intersection is contained in the original intersection. Thus, it suffices to input a restricted graph into the next recursion:

$$G_{\text{next}} = \cup\{G'_{mn} \mid (v_m, v_n) \in \texttt{gaps}\}$$

Albeit correct, actually computing the union of these graphs may cause more computational overhead than it saves. In our experiments, computing the restricted graph has not proved efficient. We input the initial intersection of FWR_0 and BWR_N into all recursions as it restricts the graph significantly while not requiring costly unions of graphs.

Let us note that Gunawan et. al use similar terminology in their approach to solving the TD-OP [60]. However, there are significant differences to the approach presented here. First, the authors solve TD-OP where travel times are time-dependent but values are assumed to be static and associated with vertices. Second, the authors propose a different algorithmic approach. While also using the terms earliest arrival and latest departure, they are computed differently and for different purposes. Given a specific path, a routine called ForwardPropagation computes the earliest arrival times at all vertices along the path. Analogously, BackwardPropagation computes the latest departures. In comparison, our FWR and BWR procedures compute earliest arrivals and latest departures for all vertices which are feasibly insertable into a given gap. From this information, the travel time for a detour to any feasibly insertable arc is derived. This is done without explicit path computation. While we use dynamic programming in a divide-and-conquer manner, Gunawan et. al implement an Iterative Local Search (optionally Variable Neighbor Descent) where actual paths are computed, improved and perturbed. In their experiments, they produce near-optimal to optimal solutions with a maximum runtime of one second. Their approach is evaluated on small data sets with at most 40 vertices and 40 time

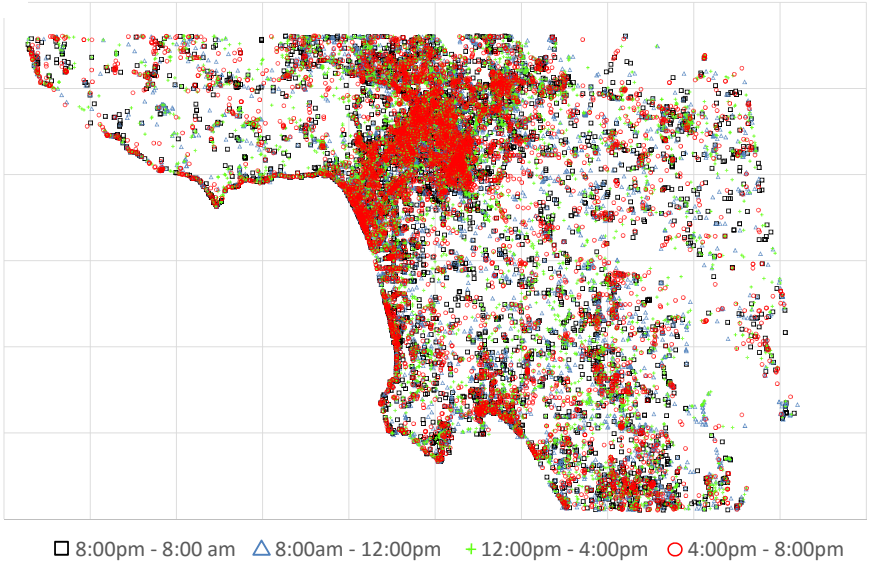


Figure 13.5: Value arcs in experimental road network of Los Angeles. Each value arc corresponds to a colored shape which indicates the arc's value peak interval.

peak interval	# of arcs	% of all arcs
8:00 am - 12:00 pm	6,238	0.57%
12:00 pm - 4:00 pm	7,366	0.67%
4:00 pm - 8:00 pm	5,851	0.53%
8:00 pm - 8:00 am	6,417	0.59%
all intervals	25,872	2.38%

Table 13.1: Statistics about arcs with non-zero value.

windows. The arcs correspond to precomputed fastest paths. This restrictive scenario is not comparable to the use case modeled in this chapter where fast response times are required for a more complex problem on graphs which are orders of magnitude larger.

13.7 Experimental Evaluation

We evaluate our solution to the 2TD-AOP on a time-dependent road network of Los Angeles, CA, USA [27, 65] which contains about 500K vertices and 1M arcs. The travel time functions in this network are derived from large-scale and high-resolution (both spatial and temporal) sensor data (both live and historic) from different transportation authorities in California. The step length of the piecewise constant travel time functions is 5 minutes, details can be found in [26]. Based on the data, streets are uncongested between 9 pm and 6 am. Therefore, travel times are assumed to be static in this interval.

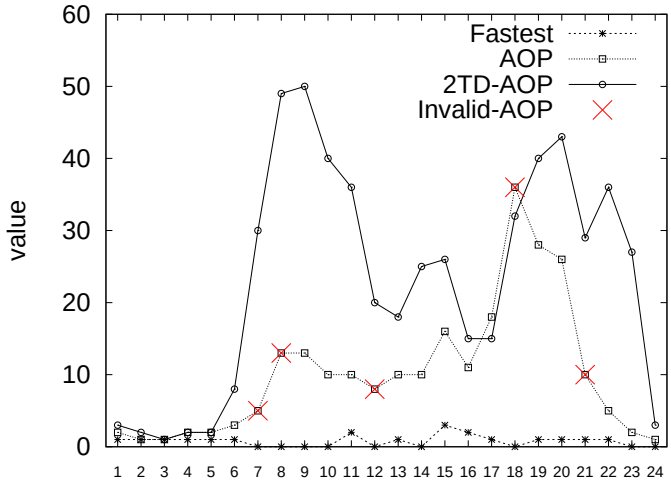


Figure 13.6: Value results for the introductory example in Figure 13.1 when varying departure time.

The time-dependent value functions are derived from a geo-tagged set of Flickr photos [102] consisting of 217,391 photos in the area of Los Angeles. For each arc (v_i, v_j) the value $val_{i,j}(t)$ is the number of photos which have (v_i, v_j) as their nearest arc (but not further than one kilometer) and which were taken in the hour-interval that contains t. Hence, the step length of the piecewise constant value functions is 60 minutes. Figure 13.5 illustrates the size of the network and the distribution of non-zero value arcs. Each arc which has positive value in some time window is visualized as a colored shape. Shape and color depend on the time interval in which the value of the arc is the highest. For example, an arc with highest value between 6:00 and 7:00 pm will be displayed as red circle. Table 13.1 presents the absolute and relative numbers of arcs with non-zero value. The number of photos taken during the day is about three times higher than during night.

For the following experiments, we arrange paths into six time buckets. We randomly draw source and destination vertices from the network and compute the respective time-dependent fastest paths using [26]. If the travel time of a path is k minutes $\pm\varepsilon$, its source and destination pair is assigned to the k minutes time bucket for $k \in \{5, 10, 15, 20, 25, 30\}$. Each time bucket consists of twenty source and destination pairs. Since travel times are empirically static at night and value peaks less frequent at night, we randomly sample departure times from the interval 8 am to 8 pm. We make a case for twofold time-dependence, therefore, a network with significant time-dependent impact on travel time and value is essential. Our standard experimental setting has the following inputs. Standard time bucket is 20 minutes, standard query budget is 200% (i.e., 40 minutes), and departure times are randomly drawn from 8 am to 8 pm. Deviating values are explicitly mentioned. For each query, the departure time is the same across time-dependent algorithms.

We omit providing results of the MIP formulation given in Section 13.3 generated by a complex solver. This is due to the exorbitant computatial requirements. Even for

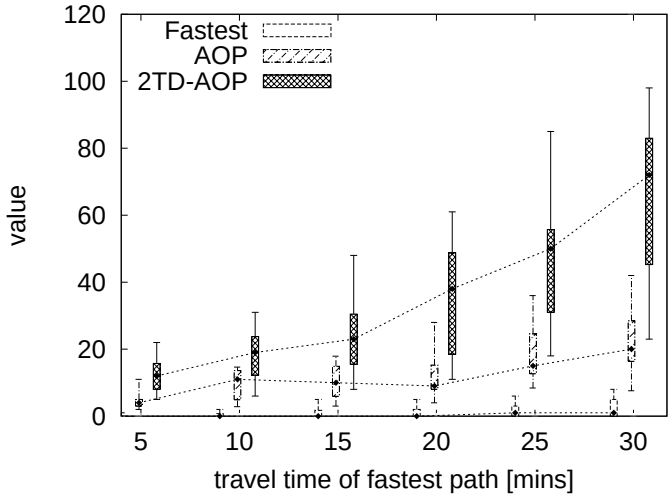


Figure 13.7: Value results for varying fastest path lengths.

small instances with at most 100 vertices, solving a less complex MIP often requires days [150, 60]. However, for comparison, we compute optimal solutions with a custom dynamic programming method.

13.7.1 Experimental Results

In a first set of experiments, we compare the value of result paths generated by different algorithms taking varying degrees of information into account. As in the introductory example, we compare 2TD-AOP paths computed by RecInsert with static scenic paths and to time-dependent fastest paths. For the static scenic paths, we use the AOP solution presented in [90] which employs an Iterative Local Search approach combined with spatial pruning techniques. In accordance with the metaheuristic, it generates an initial solution which is subsequently improved by inserted arcs and perturbed by deleting arcs. The insertion follows a heuristic, the deletion is pseudo-random. We refer to solutions created by this approach as AOP paths and call this algorithm AOP-ILS. It operates on a network with static travel time and static value which are both derived by averaging the respective time-dependent functions. Note that AOP queries are given the same time budget but require no departure time. For the time-dependent fastest paths, we use the solution proposed in [26] which relies on hierarchical routing and forward estimations to conduct efficient bidirectional path computation. We refer to the results as Fastest paths. The results generated by RecInsert are referred to as 2TD-AOP paths.

The values and travel times of all paths are computed w.r.t. the time-dependent functions. For a given path, the value and travel time are the sum of value and travel time of each arc along the path upon arrival. Recall the introductory example in Figure 13.1. For varying departure times in this example, the result values are displayed in Figure 13.6. As mentioned in the example, it is possible that paths computed in the static network are

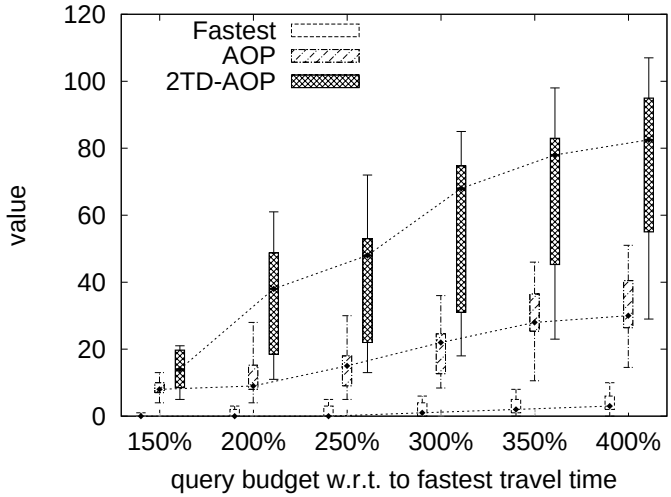


Figure 13.8: Value results for varying time budget.

invalid in the time-dependent network. This is the case if the time-dependent travel time exceeds the budget. Static paths where this is the case are marked with a red cross. For instance, when departing at 6 pm, the time-dependent travel time is 37 minutes, exceeding the budget of 25 minutes by almost 50%.

Figure 13.7 shows the value for the different approaches. Fastest paths serve a baseline which shows how much value can be attained "by chance" as it does not optimize for value. Indeed, Fastest paths collect negligible value. Since the non-zero value arcs are rather scarce, this is not surprising. 2TD-AOP and AOP, in contrast, optimize for value. It can be observed that on average 2TD-AOP generates three times the value of AOP. This gap widens for increasing path length.

Figure 13.8 illuminates the effect of the query budget. We observe that with increasing budget, 2TD-AOP paths generate significantly more value than AOP solutions. This is particularly noteworthy as the problem becomes more complex with increasing budget. A higher budget allows for greater detours which in turn increases the search space. RecInsert leverages this effect to create better solutions. In contrast, AOP results barely improve. Therefore, albeit increasing complexity, twofold time-dependence is able to enhance results considerably.

As mentioned before, static paths may prove invalid when considering time-dependent travel time. Given the departure times of 2TD-AOP and Fastest, we determine the portion of static paths that prove invalid when evaluated in the time-dependent network. The percentage of AOP which are invalid increases from 25% to almost 50% across the increasing time buckets. Note that Figures 13.7 and 13.8 show the values of all AOP paths, both valid and invalid, as there is no significant difference. However, taking into account the percentage of invalid results and the low overall value, static results cannot compete with twofold time-dependent results.

The proposed pruning strategies are evaluated in Figure 13.9. It shows the percentage

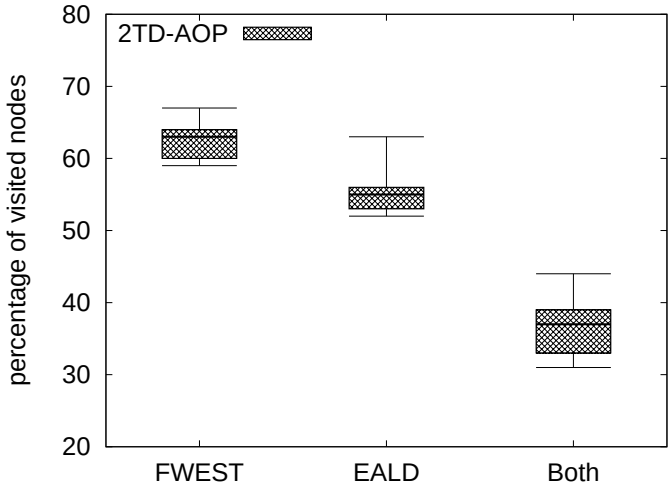


Figure 13.9: Percentage of visited vertices when employing the proposed pruning techniques.

of visited vertices relative to the number of visited vertices when employing no pruning technique. In this case, all vertices in FWR and BWR have to be visited. In Section 13.6 we propose two pruning techniques for RecInsert, FWEST and EALD. FWEST is a basic forward estimation which can be applied during FWR and BWR. EALD uses the information generated during FWR to limit the search space of BWR and can only be applied during BWR. During BWR, EALD is superior to FWEST. Hence, when both are available, FWEST is obsolete during BWR. In the experiments displayed in Figure 13.9, we show the effect of each pruning technique separately as well as of both pruning techniques combined. When only employing FWEST (during both FWR and BWR), almost 40% of the vertices can be pruned. Although EALD can only be applied during BWR, it prunes about 45% of the vertices. As FWEST causes negligible and EALD causes no computational overhead, it is recommended to employ both. Combining both approaches, 65% of all vertices in FWR and BWR can be pruned.

In terms of general complexity, the 2TD-AOP is considerably more intricate than the AOP. While both problems are NP-hard, twofold time-dependence incurs additional complexity. This is mainly due to two aspects. First, fastest path computation is more efficient in static networks. Second, in order to assess the time-dependent value of an arc, it does not suffice to simply retrieve its value. The time frame of arrival has to be considered. In light of these challenges, the efficiency of RecInsert is all the more surprising. Figure 16.6 shows the processing times of RecInsert solving the 2TD-AOP and AOP-ILS solving the AOP. 2TD-AOP solutions can be computed in less than twice the processing time of AOP solution, usually under two seconds. The increase in processing time for both algorithms in linear in the travel time of the fastest path. Taking the result quality into consideration, the additional processing time of RecInsert compensates for the often invalid AOP paths with less value. It should also be noted that in an application where

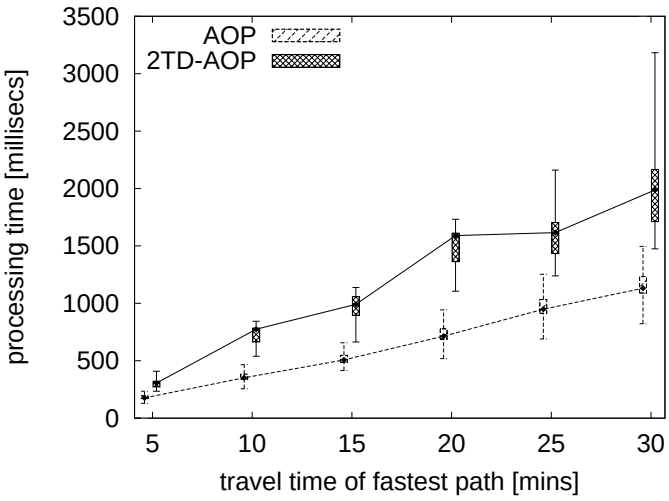


Figure 13.10: Processing time for varying path lengths.

fixed response times are required, it would be possible to terminate the arc insertion in RecInsert prematurely and construct an early result path.

Finally, we compare 2TD-AOP and AOP paths to optimal solutions in terms of their value. The optimal solutions are computed with a custom dynamic programming approach using a depth-first search and several pruning techniques. Any naïve algorithm, much like a complex solver, would not be able to compute solutions in feasible time. Our custom approach takes between 3 to 6 hours per optimal path computation for the standard query setting. Even for minor instances where the fastest path takes 5 minutes and the budget is 10 minutes, computing an optimal solution takes between 1 and 2 hours. Hence, generating optimal solutions to the 2TD-AOP is infeasible. This is irrespective of the application. Users will not tolerate response times in the hours, and precomputation is not possible in time-dependent networks. Figure 13.11 compares the accuracy of 2TD-AOP paths generated by RecInsert and of AOP paths generated by AOP-ILS. Values are shown relative to values of optimal paths. Figure 13.11 also illustrates the effect that the density of non-zero value arcs has on the result. The value is given relative to the optimal value for different networks when varying the percentage of non-zero value arcs. In our original network about 2% of the arcs have non-zero value during some time window. For this experiment, we randomly copied value functions to arcs with zero value to increase density to 4%, 6%, 8% and 10% of all arcs. Also, we set some of the value functions to zero, generating a network with 1% value arcs. As both algorithms follow a heuristic during arc insertion, they are more easily sidetracked in a network with many value arcs. When the density of value arcs decreases, however, the employed heuristics prove effective. In this case, RecInsert result paths attain between 50% and 60% accuracy. The static solutions, in contrast, hardly exceed 25%. Again, this establishes the superiority of 2TD-AOP over AOP solutions. While computing an optimal solution takes up to 6 hours, RecInsert takes about 2 seconds. Thus, RecInsert produces paths with about 50% accuracy in

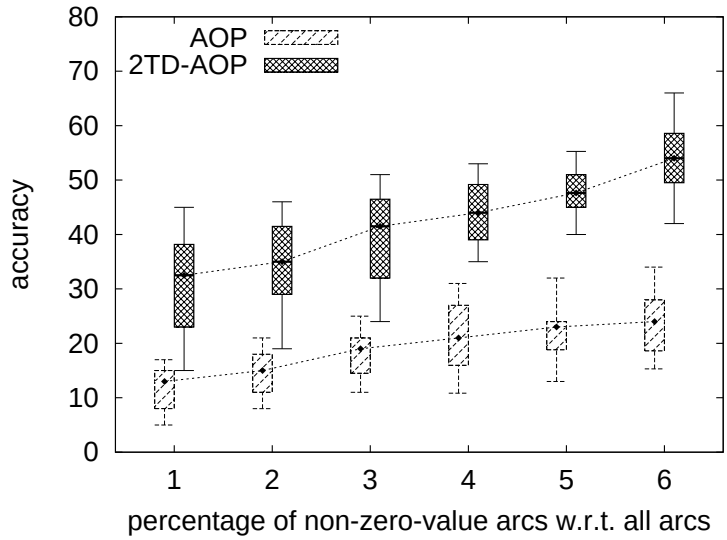


Figure 13.11: Accuracy of result values relative to optimal results when varying value arc density.

10^{-5} the processing time. As the 2TD-AOP cannot be expected to be solved optimally in efficient time, RecInsert yields a promising trade-off.

In conclusion, our experimental evaluation has established the importance of twofold time-dependence. Solutions to the 2TD-AOP are superior to solutions to the static AOP. For increasingly complex query settings, this trend is intensified. Furthermore, RecInsert computes solutions that achieve 50% accuracy compared to optimal results within seconds. The efficiency can essentially be attributed to the pruning techniques employed in RecInsert. As of now, no time-dependent AOP has been evaluated on large-scale real-world road networks. We prove that even in such networks, the higly complex 2TD-AOP can be feasibly solved.

Chapter 14

Summary

This part studied the extraction of qualitative information from crowdsourced data, in order to enrich road networks. The knowledge of the crowd reflects different notions of quality. We made the extracted knowledge applicable to routing purposes and produced paths which do not purely rely on cost measures but instead reflect some notion of popularity.

Chapter 11 and Chapter 12 covered the topics of knowledge extraction and network enrichment. Methods for knowledge extraction from heterogeneous data, including spatially enriched data, spatio-temporal data, purely textual as well as trajectory data, were presented. Using this knowledge to enrich road networks, we mapped the qualitative preferences of users onto the actual road network. Our theoretic basis was substantiated by several experiments using real-world data sets with different characteristics. Our results show that incorporating qualitative information into road networks is not only feasible but leads to enhanced solutions for the task of path computation. Result paths which reflect qualitative knowledge without sacrificing their quantitative aspect. Furthermore, the demo framework *Tourismo* was described. *Tourismo* employs some of the presented methods to the application of tourist route recommendation.

In Chapter 13, a related problem was proposed. The Twofold Time-Dependent Time-Dependent Arc Orienteering Problem (2TD-AOP) is a novel extension of the family of Orienteering Problems (OP). In this family of NP-hard graph routing problems, the goal is to find a path from a given source to a given destination within a given time budget which, additionally, maximizes the value collected along the way. In comparison to static versions of this problem, 2TD-AOP allows for travel times and value functions to be time-dependent. The incorporation of time-dependent values is a novel extension which has not been studied yet. The importance of time-dependent values was showcased and experimentally substantiated. Due to the NP-hardness and the twofold time-dependence, a heuristic approximation for solving the 2TD-AOP is provided. Our approach was evaluated on a large-scale real-world road network. Thus far, no time-dependent variant of the AOP has been evaluated on a network of this dimension. Twofold time-dependent results gain significantly more value than static results, showing the importance of twofold time-dependence empirically. Our algorithm produces result paths with 50% accuracy where no optimal solution is feasible.

Part IV

Sensor Data Applications and Query Processing

Chapter 15

Introduction

This part of the thesis is dedicated to novel queries in modern traffic networks. Aside from the archetypical cost-optimal path query from A to B, there are various other facets of traffic to be optimized. Especially urban areas are increasingly overstrained, while at the same time efficiency is a vital asset. In order to make this balancing act work, traffic needs optimization. In the past years, advances in the field of sensor technology and data science have facilitated such optimizations. For instance, using the crowd and their mobile devices' GPS sensors, apps like Waze[1] provide routes which take real-time traffic conditions into account in order to minimize time in traffic. Transportation apps like Mytaxi[2] and Uber[3] show available drivers in real-time, increasing user-friendliness through instant feedback and predictable waiting times. Projects like SFPark[4] provide real-time information about vacant parking spaces enabling flexible pricing models while shortening the search for parking spaces.

The benefit of sensor data for traffic is manifold. In the next years, the use of such data will undoubtedly increase further. However, optimization of traffic in general or special routing tasks in particular are not the only aspects which benefit from sensor data. Another important direction is driving convenience, i.e., besides reducing the time in traffic, making the time which has to be spent in traffic as comfortable as possible. This may for example be achieved by providing user-dependent directions. Taking personal preferences into account when computing routes can lead to paths which better reflect the user's liking and thereby contribute their overall driving experience.

In the following, works are presented which address problems with such practical application. In Chapter 16, we first examine the topic of personalized routing. We present an approach which learns a user-dependent preference distribution. This distribution may in turn be used to provide paths which correspond to the preference of the particular user. Several methods exist in the area of personalized routing, but in contrast to existing approaches, we model driving preferences in an abstract manner, i.e., independent of the area

[1] www.waze.com
[2] www.mytaxi.com
[3] www.uber.com
[4] www.sfpark.org

and independent of the absolute cost values of the recorded trajectory.

In Chapter 17, we investigate how to increase the efficiency of specific traffic problems by employing sensor data. We introduce the concept of resources in road networks such as vacant parking spaces or vacant charging stations for electric vehicles. While it may be known where such resources are, it is generally not known if they are available. We present a probabilistic model for the availability of resources which allows to incorporate sensor data in the form of short-term as well as long-term observations, i.e., ad-hoc information about available resources as well as aggregated historic information. Given this model, we develop routing algorithms which maximize the probability of finding a particular resource.

Finally, in Chapter 18, we present the demonstration tool EasyEV which combines features that have been developed for the management of fleets of electric vehicles (EVs). Due to range limits and long charging intervals, the use of EVs is often more problematic than that of cars with combustion engines. These drawbacks are even more limiting when EVs are used in a fleet of vehicles. In a research project with numerous industry partners, smart car data shaped up to be the solution. By employing sensor data into the fleet management system, deficiencies were reduced and the degree of capacity utilization increased. Some of the features developed for the project have been implemented in EasyEV and are described in Chapter 18.

This research has been published in [4, 67, 71]. Preliminary results have been published in [70, 69].

Chapter 16

Mining Driving Preferences in Bicriteria Networks

16.1 Introduction

When proposing a path to a driver, most navigation systems rely on the optimal path between start and destination. However, when comparing the suggested paths to real-world trajectories, it can be observed that drivers often select paths which differ significantly from the proposed path. One reason for this effect is that the travel time between two places depends on a variety of time-dependent and uncertain parameters like the behavior of other vehicles or the synchronization between the travel progress and traffic lights. Thus, instead of trying to predict the travel time directly, experienced drivers rather try to select a path based on the trade-off between the risk of delay and the opportunity to reach the destination in minimal time. Consider a risk averse driver who wants to make certain that he reaches a goal in time but is not necessarily in a hurry. In this case, our driver would most likely prefer to avoid areas with a large likelihood of congestion and a high density of traffic lights, even if the resulting path is considerably longer than the shortest path containing such risk factors. On the other hand, a more optimistic driver would choose the shortest path while assuming that delays at traffic lights or due to dense traffic will not be severe. The path a driver actually chooses depends not only on its measurable qualities but also on the driver's preferences. To capture this effect, the research community has begun to model the driving behavior of people. The idea which most approaches follow is to interpret the number of times a certain driver has chosen a particular segment or subpath as a personal preference. From this kind of data it is possible to build a statistical model describing the preferred paths of a driver. However, this is only applicable in areas where sufficient data has been collected. Also, if for a certain subpath there are no alternatives, then taking this subpath cannot be assessed as intent.

We therefore propose a new approach to modeling driving preferences. In this Chapter, driving preference are conceived as set of trade-off factors between two cost criteria. For example, a path might be described by its distance and the number traffic lights. Our

model is based on the assumption that the selection of a path depends on a combination of two cost criteria. Consider a path with 5 kilometers distance but 10 traffic lights. If there exists an alternative path of 6 kilometers distance but only 4 traffic lights, a user might prefer this path over the other because, to this particular driver, the 6 additional traffic lights outweigh the additional kilometer.

Formally, the preferences of a driver are captured by the trade-off of one cost criterion compared to another one. A driver selects a certain path because it is optimal w.r.t. this particular trade-off. Since people do not necessarily display consistent preferences, our method estimates a preference distribution over all available observations. To estimate this distribution, it is important to distinguish between the observed cost trade-offs and the preferences that can be derived from them. If there are no alternatives for a certain path, then taking this path is no implication of intent. Hence, for an observed path, we compute its alternatives in order to decide why this path was chosen instead of its alternatives. To derive a reasonable set of alternative paths, we rely on the path skyline [128, 80]. We assume that a rational and informed driver always selects cost-optimal paths.

To build a model of the preferences of a driver, our method requires a set of trajectories and the corresponding road network. Each sample trajectory is mapped to the network and compared to the path skyline for the same start and target. From this, we receive a cost vector for the observed trajectory and a set of cost vectors for the unselected skyline paths. Subsequently, we derive a preference interval in which the selected path is better than the alternative skyline paths. By aggregating these preferences, we obtain a distribution over the preferences of a driver. If the preferences are consistent, the distribution will exhibit a tight window for the tolerated trade-off between costs.

The contribution of this chapter are:

- A novel model for describing driving preferences as the set of trade-offs between cost factors.
- A basic method for estimating the preference distribution based on observed trajectories of a user compared to the alternative skyline paths.
- An improved algorithm that significantly decreases computational cost.

The rest of this chapter is organized as follows: In the Section 16.2, we survey related work on multicriteria routing, path prediction and modelling driving preferences. Section 16.3 gives basic definitions. The probabilistic model is formalized and the algorithm is introduced in Section 16.4 and Section 16.5. The experimental evaluation in Section 16.6 examines the quality of the derived preference distributions and shows the driving preferences on real-world data. The chapter concludes with a summary in Section 16.7.

16.2 Related Work

There often is a significant difference between the paths being suggested by a navigation system and the paths being driven by experienced drivers with knowledge about the area.

There are two major reasons for this difference. The first is that the model of the road network used for routing is rather incomplete because it lacks important information about advantages and disadvantages of road segments. The second shortcoming is that the routing algorithm cannot consider the personal preferences of users. To handle these problems there exist approaches analyzing observed trajectories to gather more accurate information and learn personal driving preferences.

In [75] the authors describe one of the first approaches to path prediction for optimizing a shared fleet of cars. The setting shares some similarity to this work but does not personalize the prediction. [132] describes a system that uses GPS tracks to build a hidden Markov model for predicting paths and destinations. The system strongly relies on the fact that users tend to prefer the same set of paths and often travel to the same locations. In [42], the authors focus on improving the input data by proposing a pipeline from raw GPS data to map-matched trajectories on a road network. An important aspect of the approach is that frequently driven paths have to be discovered and outlier trajectories should not be considered for learning models about preferred paths. In [16], the authors propose a model for path prediction on patterns of visited locations. The system is built upon a client-server architecture considering the privacy aspects of the collected trajectory data. The T-Drive system [162, 161] is based on the trajectories of 33,000 taxis in the city of Beijing which were analyzed over the time period of three months. The idea behind this system is to research the paths of professional drivers and build a model which imitates their behaviour when proposing a path to a driver. Technically, the system exploits the information by deriving time-dependent traversal times for central road segments. Furthermore, the system generates a landmark graph of preferred road segments called landmark road segments. The landmarks form the backbone for a specialized network containing streets used by the taxi drivers. In combination with the improved time-dependent travel time estimation, the authors demonstrate that the system is capable of proposing paths which require significantly less time than the paths proposed by conventional routing algorithms. This method adapts the proposed paths to the preferences of taxi drivers, however, the system does not distinguish between different preferences. Instead, the general assumption in T-Drive is that all drivers prefer the fastest path and taxi drivers share the joint experience to achieve this goal. A system which actually models personal driving preferences is the TRIP system [82]. Similar to T-Drive TRIP is based on observed trajectories but does not join these observations to build a global model of professional driving behavior. Instead, TRIP works with a much smaller set of personal trajectories and models each driver separately. Similar to T-Drive, the system employs all observed trajectories to generate time-dependent travel times. Furthermore, the system computes an inefficiency ratio for each user, modeling the importance of travel time in his routing decisions. The inefficiency ratio is then employed to lower the cost of road segments a driver has actually travelled before. Thus, the proposed path will contain more already known road segments if the driver has a low efficiency ratio.

Though all of these systems share a common motivation with the work proposed in this chapter, there is an important difference. In previous approaches, riving preferences are modeled w.r.t. preferred localities in a given area. Thus, none of these approaches is

applicable in areas where no trajectories have been recorded. However, this is the case where a navigation system is most needed.

16.3 Problem Setting

Our approach is based on the assumption that a driver chooses his paths consciously, i.e., with knowledge of the alternatives. Hence, any path p selected by a driver is expected to be optimal w.r.t. their personal cost combination function is a trade-off of the given attributes. We call these trade-offs preference gradients or gradients, for short.

Definition 16.1. *Given a path p with costs $p_1 := c_1(p)$ and $p_2 := c_2(p)$, we refer to the gradient p_2/p_1 as the* (preference) gradient*. W.l.o.g., we assume any gradient to be scaled such that $p_1 + p_2 = 1$. Hence, any gradient is uniquely defined by its first component.*

When comparing paths for different query settings, it is possible that the driver will choose according to his previously used preference. The driver's preference is encoded in the preference gradient of each path. Note that the gradient is an abstract trade-off in the cost space. It is independent of start and target nodes, and thereby of locality in general. Any preference gradient can be used as a weight vector when computing paths employing a scalarized search. The concept is equivalent to exactly specifying the desired trade-off which is usually not possible. In relation to the gradient, we define the notion of a weight and its weighted cost.

Definition 16.2. *Given a path p and a weight $\gamma \in [0, 1]$, we refer to*

$$\gamma' := \begin{pmatrix} \gamma \\ 1 - \gamma \end{pmatrix}$$

as the gradient vector *of γ and define the* weighted cost *of p w.r.t. γ as*

$$c_\gamma(p) := \max\{\gamma p_1, (1 - \gamma) p_2\}$$

Of course, no driver will always navigate according to the same preference. This might be due to different moods (casual driving on weekends as opposed to efficiency-optimized driving during the week) or simply due to a lack of paths reflecting the preference. Thus, we assume a preference distribution which the driver implicitly constitutes over time with every path he chooses. The preference distribution models his characteristic behavior in traffic. Any chosen path may be perceived as a sample drawn from the driver's preference distribution and computing the cost-optimal path w.r.t. this trade-off value. This concept of deriving preferences and personalized routing suggestions is exemplified in Figure 16.1 and formally introduced in the following section.

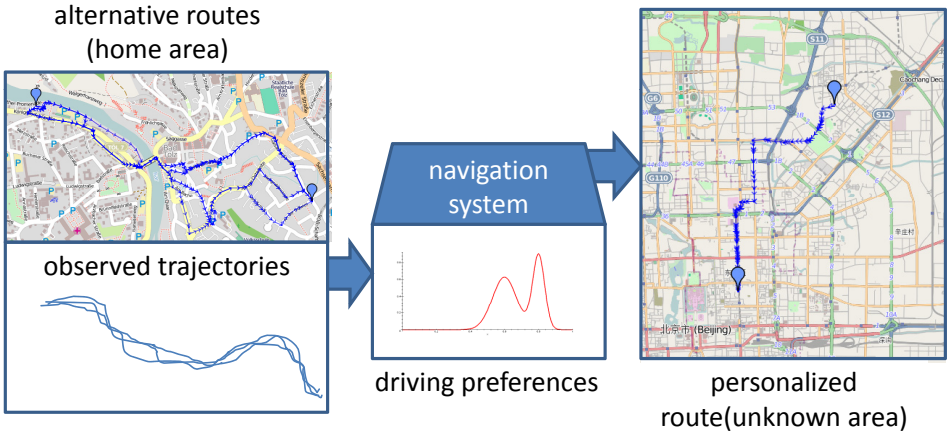


Figure 16.1: Illustration of the concept of our solution.

16.4 Preference Distributions

If a driver chooses a particular path, we assume it to be tracked by GPS and refer to the raw data as a trajectory. Matching the raw data onto the underlying road network (e.g., using [105] or techniques presented in [163]), we obtain the corresponding path and may derive its cost vector. Essential to our concept of driving preferences are not only the chosen path but also which have not been selected. Without knowledge of the alternative paths a driver had prior to his decision, no preferences can be inferred. For instance, consider a driver who is interested in maximizing the driving flow. If every path to their current destination has a significant number of traffic lights, then whichever path is chosen will not properly reflect their preference due to a lack of suitable options.

We regard every chosen path independent of its locality, in the abstract cost space. For a chosen path p, connecting start and target nodes s and t, we take the alternatives to p into account, i.e., $\mathcal{P}(s,t)$. By comparing p to its alternatives in the cost space, we obtain what we refer to as preference interval of p.

Definition 16.3. *Given a path p from s to t, we define the* preference interval *of p, $I(p)$, as follows:*

$$I(p) := \left\{\gamma \in [0,1] \mid c_\gamma(p) \leq \min_{q \in \mathcal{P}(s,t)} c_\gamma(q)\right\}$$

Note that the preference interval is independent of the actual locality of the chosen path. For a given start and target pair, there exist countless possible paths (e.g., every arbitrarily long detour), of which Definition 16.3 takes every one into consideration. However, it suffices to consider non-dominated paths. It is evident that for any path q' which is dominated by path q holds $c_\gamma(q) < c_\gamma(q')$ for all γ.

Limiting the definition of the preference interval to skyline paths corresponds to the assumption that an informed driver chooses optimal paths, ideally according to his prefer-

ence. In reality, of course, drivers also choose suboptimal paths. This might be for different reasons. For instance, an ad-hoc decision to bypass a congested area might be faulty or the data used by the navigation system might not be up to date. We exclude the possibility that drivers deliberately decide to take suboptimal, i.e., dominated paths. Thus, when comparing chosen paths to their alternatives which were not chosen, we restrict ourselves to skyline paths. If a chosen path is suboptimal, we map it onto its skyline correspondent (see Figure 16.2(a)).

Definition 16.4. *Let $\mathcal{S}$ be the skyline for the same start and target nodes as path p. Then the* skyline correspondent *of p is*

$$\arg\min_{s\in\mathcal{S}} \left\| \begin{pmatrix} s_1 \\ s_2 \end{pmatrix} - \frac{s_1 p_1 + s_2 p_2}{\|(p_1, p_2)\|} \right\|$$

the skyline element whose projection onto the line $p_2/p_1 \cdot x$ has minimal length.

The inherent one-dimensionality of the preference interval is captured in the following definition.

Definition 16.5. *Let $p \in \mathcal{S}$ be a skyline path. The* lower and upper preference boundaries *of p are defined as follows:*

$$\alpha := \frac{1 - \max I(p)}{\max I(p)}$$
$$\beta := \frac{1 - \min I(p)}{\min I(p)}$$

i.e., the $\big(\gamma, 1-\gamma\big)^T$ of $I(p)$ with minimal and maximal gradients, respectively. If $\min I(p) = 0$, we set $\beta = \infty$ We refer to

$$\mathcal{D}(p) := [\min I(p), \max I(p)]$$

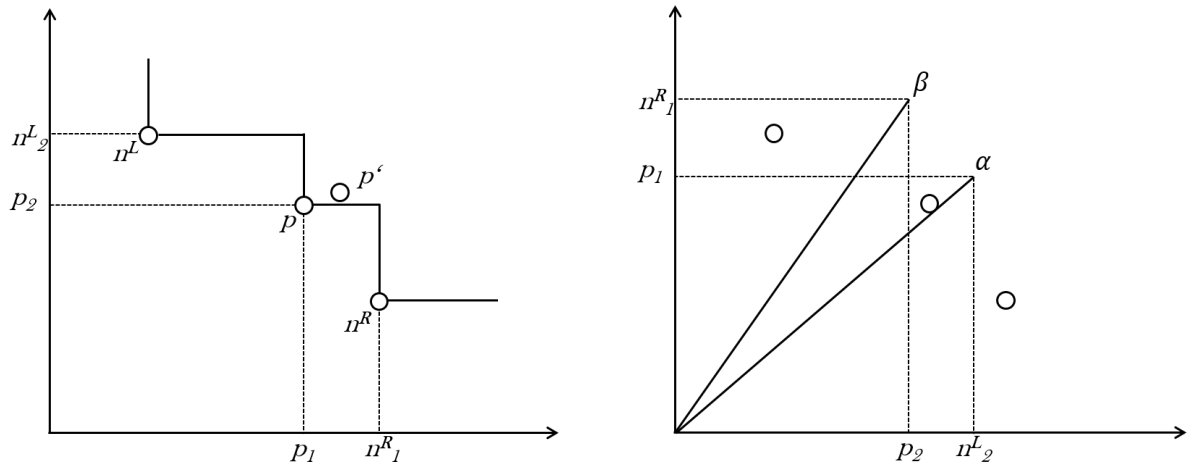
as the driving preference *of p. It is a subset of the preference space $[0, 1]$.*

Figure 16.2(b) shows the lower and upper preference boundaries of a path depending on its skyline correspondent and its skyline neighbors (visualized in Figure 16.2(a)).

Our goal is to generate a preference distribution which reflects a driver's particular behavior. Mapping their recorded trajectories onto the road network and computing the corresponding upper and lower preference boundaries for each path, we obtain a set of preference boundaries. We approximate the actual distribution with a histogram of user-defined granularity. Our preference distribution is defined as follows.

Definition 16.6. *Given a set of paths P, and a granularity parameter k, the joint* preference distribution *consists of k equisized partitions of the preference space, $I_j := [j/k, (j+1)/k], 0 \le j \le k-1$. The preference distribution $\mathcal{D}(\mathcal{P})$ is defined as follows:*

$$\mathcal{D}(\mathcal{P})[j] := \frac{\sum_{p\in\mathcal{P}} \mu_j(p)}{\sum_{0\le j\le k-1} \sum_{p\in\mathcal{P}} \mu_j(j)}$$



(a) Chosen path p' visualized in cost space including skyline correspondent p and neighbors n^L, n^R.

(b) Preference gradients derived from p, n^L and n^R.

Figure 16.2: Preference boundaries depending on chosen path and its skyline neighbors.

where $\mu_j(p) := \frac{\lambda(I_j \cap D(p))}{\lambda(I_j)}$ *is the fraction of* $D(p)$ *which lies in* I_j *normed by the size (i.e., Lebesgue measure* λ*) of* I_j*.*

Before we present our basic Algorithm 11, we introduce an alternative definition of the preference boundaries and show that the definitions coincide.

Definition 16.7. *Let* $p \in \mathcal{S}$ *be a skyline path. Let* $n^L, n^R \in \mathcal{S}$ *denote the left and right skyline neighbors of* p*, respectively, i.e.,* $n^L := \arg\max_{s \in \mathcal{S}} s_1 \leq p_1$ *and* $n^R := \arg\min_{s \in \mathcal{S}} s_1 \geq p_1$*. The* lower and upper preference boundaries *are defined as:*

$$\alpha := \begin{cases} 0, & \text{if } n_L = p \\ \frac{p_1/(p_1+n_2^L)}{n_2^L/(p_1+n_2^L)}, & \text{else} \end{cases}$$

$$\beta := \begin{cases} \infty, & \text{if } n_R = p \\ \frac{n_1^R/(n_1^R+p_2)}{p_2/(n_1^R+p_2)}, & \text{else} \end{cases}$$

Note that the normalization done in both nominator and denominator is only for the purpose of summing to one.

Lemma 16.1. *The Definitions 16.5 and 16.7 coincide.*

Proof. We first show the extreme cases ∞ and 0 by equivalence, the rest is proved constructively.

(i) The following equivalence holds:

$$\begin{aligned}\alpha = 0 &\Leftrightarrow \max I(p) = 1\\ &\Leftrightarrow c_1(p) < \min_{p\neq q\in\mathcal{S}} c_1(q)\\ &\Leftrightarrow p_1 < \min_{p\neq q\in\mathcal{S}} q_1\\ &\Leftrightarrow n^L = p\end{aligned}$$

(ii) Analogously:

$$\begin{aligned}\beta = \infty &\Leftrightarrow \min I(p) = 0\\ &\Leftrightarrow c_0(p) < \min_{p\neq q\in\mathcal{S}} c_0(q)\\ &\Leftrightarrow p_2 < \min_{p\neq q\in\mathcal{S}} c_2(q)\\ &\Leftrightarrow n^R = p\end{aligned}$$

(iii) Let n^L be distinct from p, then according to Definition 16.7

$$\frac{1-\gamma}{\gamma} = \alpha = \frac{p_1/(p_1+n_2^L)}{n_2^L/(p_1+n_2^L)}$$

We show that α in fact constitutes the maximum of the preference interval if defined as above. By this definition, $\gamma = n_2^L/(p_1+n_2^L)$. In the following, we omit the denominator as it has no influence on the maximum. It holds

$$\begin{aligned}c_\gamma(p) &= \max\{n_2^L p_1, p_1 p_2\} = n_2^L p_1\\ c_\gamma(n^R) &= \max\{n_2^L n_1^L, n_2^L p_1\} = n_2^L p_1\end{aligned}$$

because $n_2^L > p_2$ and $n_1^L < p_1$ from the neighbor property. This shows equality for the lower preference boundary. If we put on more weight on the first component, e.g., by increasing the first component (n_2^L), then $c_\gamma(n^L) < c_\gamma(p)$. When putting more weight on the second component, $c_\gamma(p)$ constitutes the minimum. Hence, $n_2^L/(p_1+n_2^L)$ is indeed the maximum of the preference interval $I(p)$. The situation is exemplified in Figure 16.2(a), the corresponding boundaries are visualized in Figure 16.2(b).

(iv) Analogously to (iii), for

$$\frac{1-\gamma}{\gamma} = \beta = \frac{n_1^R/(n_1^R+p_2)}{p_2/(n_1^R+p_2)}$$

β indeed constitutes the minimum of the preference interval.

$$\begin{aligned}c_\gamma(p) &= \max\{p_2 p_1, n_1^R p_2\} = n_1^R p_2\\ c_\gamma(n^R) &= \max\{p_2 n_1^R, n_1^R n_2^R\} = n_1^R p_2\end{aligned}$$

Algorithm 11 Basic Preference Distribution

Data: Set of mapped trajectories $\mathcal{P}$
Output: Normalized preference distribution

```
1  begin
2      foreach path p ∈ P do
3          compute skyline S for corresp. start and target, sort asc. by c_1
4          if p ∉ S then
5              p ← skyline correspondent of p
6          end
7          foreach q ∈ S with q_1 < p_1 do
8              compute preference boundary between p and q
9              store largest preference boundary α
10         end
11         foreach q ∈ S with q_1 > p_1 do
12             compute preference boundary between p and q
13             store smallest preference boundary β
14         end
15         add interval [1/(β+1), 1/(α+1)] to preference distribution
16     end
17     return normalized preference distribution
18 end
```

If we put more weight on the second component, e.g., by decreasing p_2, then $c_\gamma(n^R) < c_\gamma(p)$. Conversely, if we put more weight on the first component, $c_\gamma(p)$ constitutes the minimum. Hence, $p_2/(n_1^R + p_2)$ is indeed the minimum of the preference interval $I(p)$.

This proves the lemma. □

16.5 Algorithms

In this section, we describe our algorithms to derive location-independent preference distributions from a set of trajectories. We begin with the basic algorithm which is described in Algorithm 11. For each of the given trajectories, the corresponding path skyline is computed. Subsequently, the preference boundaries are computed and the corresponding preference interval is used to build the according preference distribution.

Computing the entire path skyline can be costly, even in bicriteria networks. Therefore, we optimize Algorithm 11 in such way that it reduces the number of computed skyline paths. As proved in Lemma 16.1, it suffices to know the left and right skyline neighbor of the skyline correspondent of the input path. Algorithm uses this property for acceleration.

Instead of computing the complete path skyline, Algorithm 12 employs global bounds

and a prioritization to restrict the number of path computations. First, the cost-optimal paths w.r.t. to both criteria c_1 and c_2 are computed, they constitute upper bounds for c_2 and c_1, respectively. Similar to ARSC [80], a local skyline is maintained at every node, and paths are expanded iteratively if locally non-dominated. In every iteration, all locally non-dominated paths at a certain node are expanded. The node is the top element of a priority queue sorted by the best maximum value of all paths in the local skyline. Once a path expansion reaches the target, we check if the path constitutes one of the skyline neighbors of the reference path. If it is a skyline neighbor, the upper bounds may be decreased. The algorithm terminates once the queue is empty or the top element's cost exceeds the bounds. Algorithm 12 uses the left and right skyline neighbors of each trajectory's skyline correspondent to compute the respective preference interval. As before, the normalized preference distribution is returned.

16.6 Experimental Evaluation

In this section, we describe the results of our experimental evaluation. We examine three aspects of the proposed algorithms. First, we deliver a proof of concept, i.e., we investigate whether it is possible to reproduce a given preference distribution. Second, we examine real driving behavior derived from recorded trajectories. Third and last, we analyze the efficiency of the proposed methods in terms of runtime. Both algorithms are implemented the MARiO framework [58] which is based on OpenStreetMap (OSM) data. Experiments are conducted in the areas of Beijing, China, and Bad Toelz, Germany, the road network graphs derived from the raw data have around 30K nodes, 38K edges and 9K nodes, 14K edges, respectively. As cost criteria, we used distance and the number of traffic lights. All experiments were conducted on a machine with an Intel Centrino 2 processor and 2 GB of RAM.

In our first experimental setting, we show that our approach is capable of reconstructing driving preferences from observed trajectories. In order to do so, we generate preference distributions based on Gaussians with given mean and standard deviation. Subsequently, we produce a trajectories by drawing a preference from the distribution and computing the cost-optimal path for a randomly selected pair of nodes. For the following tests, we generated 200 random trajectories and employed our method to reconstruct the distribution function.

In a first experiment, we tested the method for different mean values, 3.0 and 10.0 with the same standard deviation of 0.5. The preference distribution derived by the algorithms presented in Section 16.5 is displayed in Figures 16.3(a) and 16.3(b). For both mean values, the peak of the distribution coincides with the chosen values. In a second experiment, we tested the tolerance w.r.t. different standard deviations. We compared a standard deviation of 0.5 to a standard deviation of 3.0. The resulting preference distributions are depicted in Figures 16.4(a) and 16.4(b). Even for the large standard deviation, the preference distribution bears significant resemblance to the generating distribution.

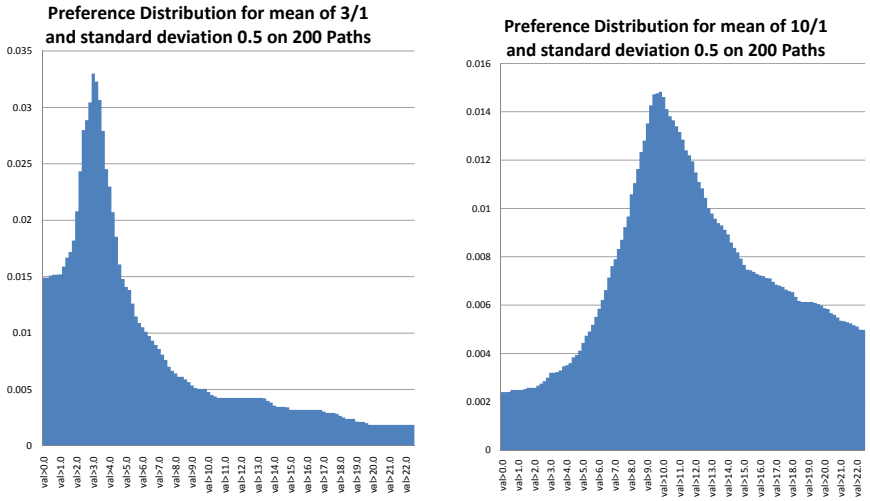
In a third setting, we tried to reconstruct a mixture of two equally weighted Gaussian

Algorithm 12 Fast Preference Distribution

Data: Set of mapped trajectories $\mathcal{P}$
Output: Normalized preference distribution

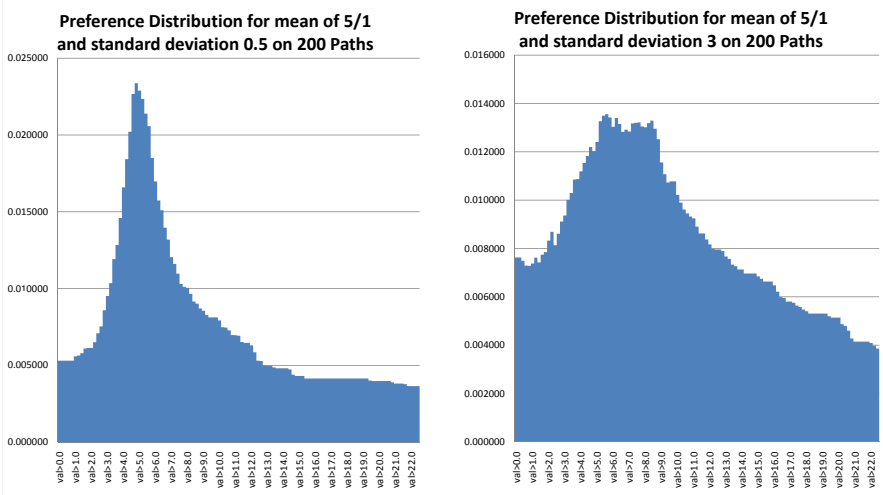
```
1  begin
2      foreach path p ∈ P (or their skyline correspondents) do
3          compute optimal paths o_1, o_2 w.r.t. c_1, c_2
4          set max values for both criteria, m_1 := c_1(o_2), m_2 := c_2(o_1)
           // maintain local skyline S(s,n) at each node n
5          init queue Q of nodes n, sort asc. by min_{q∈S(s,n)} max{c_1(q), c_2(q)}
6          add s to Q
7          init skyline neighbor variables n^L, n^R as null
8          while Q not empty and top value of Q < max{m_1, m_2} do
9              n ← top element of Q
10             if n = t then
11                 foreach paths q in S(s,n) with q_1 < m_1 and q_2 < m_2 do
12                 end
13                 if q_1 < p_1 then n^L ← q and m_2 ← q_2
14                 if q_1 > p_1 then n^R ← q and m_1 ← q_1
15             end
16             else
17                 foreach path q in local skyline S(s,n) do
                       // if lower bound forward estimation available, apply
18                     {q_1, ..., q_n} ← extend q by adjacent node n'
19                     foreach q_i non-dominated in S(s,n') do
20                         insert into Q and possibly update priority
21                     end
22                 end
23             end
24         end
25         compute α and β from n^L and n^R, add [1/(β+1), 1/(α+1)] to preference distribution
26     end
27     return normalized preference distribution
28 end
```

distributions with a standard deviation of 0.5 and mean values of 3.0 and 10.0. The results are shown in Figure 16.5(a). Two clear peaks corresponding to the means of the contributing Gaussians are visible. In conclusion, the proposed technique is indeed of reconstructing preference distributions. We note that there is no clear rule on the number of paths needed to provide meaningful distribution. Aside from the quantity, the size of the preference areas of the used paths plays a significant role. Paths with many alternatives typically yield smaller and therefore more decisive preference areas. Hence, even relatively



(a) Mean 3.0 (b) Mean 10.0

Figure 16.3: Preference distributions for a mean gradient of 3/1 (left) and 10/1 (right).

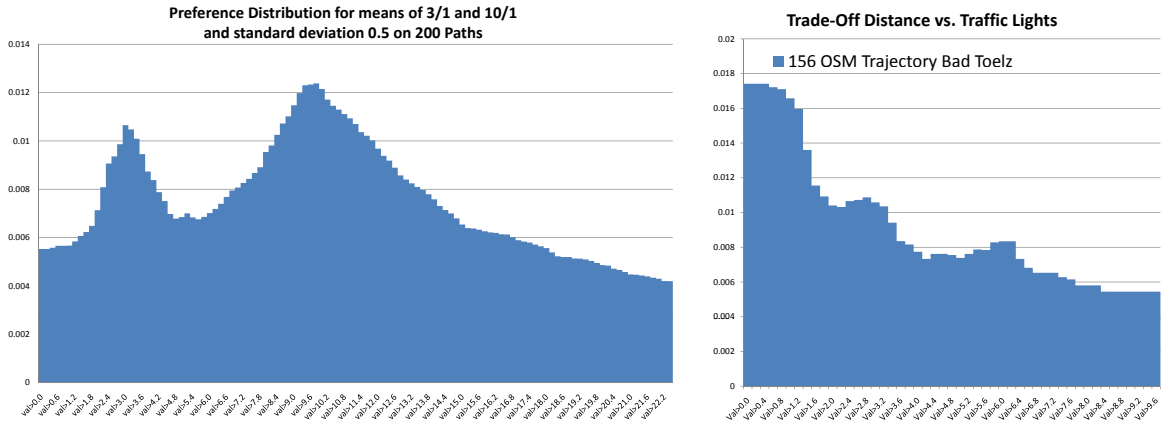


(a) Standard Deviation 0.5 (b) Standard Deviation 3.0

Figure 16.4: Preference distributions for a mean gradient of 5/1 and standard deviations 0.5 (left) and 3.0 (right).

few well-selected observations can be sufficient to provide a solid approximation of the preference distribution.

In the next block of experiments, the preference distributions of real trajectories are examined. In order to generate realistic, trajectories and network data has to be of high quality. Although there is some trajectory data publicly available, e.g., from [162], it often has drawbacks. Either it is located in areas where the map data is incomplete (e.g., in Beijing road segments and traffic lights are missing) or the quality of the recorded data is



(a) Mixture Model (b) OpenStreetMap Trajectories

Figure 16.5: Left: Computed distribution for a mixture model of two Gaussians with standard deviation 0.5 and means 3/1 and 10/1. Right: Preference distribution of 156 raw trajectories the OpenStreetMap map of Bad Toelz is built upon. The distribution shows three different peaks. generated by various drivers.

poor (e.g., weak GPS signal leading to measurement errors, many stationary trajectories or other outliers). Thus, we chose to evaluate our method on the OSM trajectories which serve as the ground truth of the well-tended OSM network of Bad Toelz, Germany. We employed a simple topological map-matching approach to map the trajectories onto the road network graph. After sorting out cyclic paths (i.e., obviously inefficiently chosen paths to generate data), 156 paths could be gathered. The corresponding preference distribution is depicted in Figure 16.5(b). As cost criteria, we chose distance and traffic lights. The strongest peak of the distribution reflects the decision of driving the shortest path regardless of the number of encountered traffic lights. However, there are two weaker peaks for the values 2.8 and 6.2, showing that some of the paths were longer but with less traffic lights. Let us note that deriving several peaks from the data at hand is not surprising, as the data has been generated by different users, and the trajectories do not provide the generating users' names. Thus, running personalized tests was not possible. Nevertheless, even for a group of drivers, the preference distribution displays peaks and can therefore be utilized to propose path alternatives which better reflect the trade-offs inherent in the recorded trajectories.

In a final experiment, we examine the performance of Algorithms 11 and 12. Recall that instead of computing the entire path skyline for every trajectory, Algorithm 12 relies on the skyline neighbors of the recorded trajectory's skyline correspondent and additionally employs bounds for pruning purposes and prioritizes path expansion according to the overall costs. The path skyline in Algorithm 11 was computed using the ARSC algorithm [80] with a reference node embedding for lower bound cost estimations. We measure the average runtime for computing one preference area and the amount of visited nodes on the Beijing graph for 1000 queries. Figure 16.6 shows the results. Using Algorithm 12, the

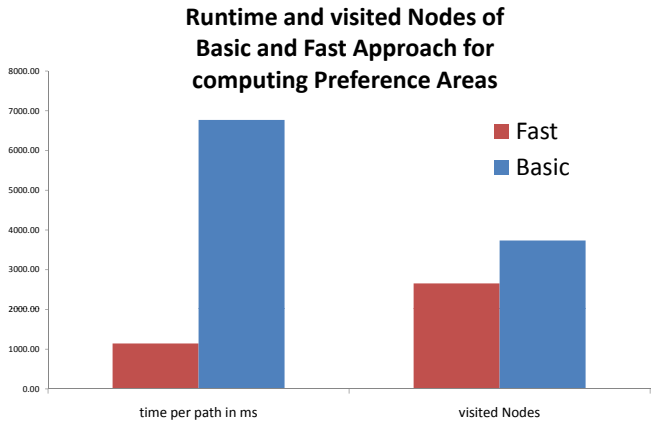


Figure 16.6: Performance comparison between the basic algorithm 11 and the accelerated algorithm 12.

amount of visited nodes can be reduced drastically to 30% of those visited by Algorithm 11. In terms of runtime, Algorithm 12 outperforms its competitor and produces results about 6 times faster. This is due to the fact that the overhead for processing unnecessary paths and their extensions is significantly smaller. For determining preference areas in the Beijing network, our algorithm needed less than one second per trajectory. Thus, it would be possible to maintain preference distributions even in embedded system with computational restrictions.

16.7 Conclusions

In this chapter, we introduced a novel concept for modeling driving behavior. In contrast to existing approaches, our work is independent of location and absolute values of the chosen paths. Furthermore, it allows for and identifies different driving moods by generating a distribution which reflects personal driving preferences. This distribution can be used to predict future driving behavior and help to provide paths which better reflect individual driving styles. We proposed two algorithms, a basic and an improved variant which relies on geometric properties of the set of pareto-optimal paths. Our experiments show the effectiveness as well as the efficiency of our algorithms.

Chapter 17

Probabilistic Resource Route Queries with Reappearance

17.1 Introduction

The basic routing task of finding a path from start to target is a well-explored research area. Other routing tasks are as common in everyday life but have drawn far less attention. An example are trip planning queries (TPQ) specified by start and target locations and a number of resource types which have to be visited along the trip. For instance, the user might provide the resource types "ATM", "gas station", and "department store". The result of such a query is the shortest path from start to target visiting exactly one instance of each resource type. There are several variants to this kind of problem which will be reviewed in Section 17.2.

Similarly relevant to everyday life is the following varation. Instead of expecting the resources to exist at each of their locations, it may be more realistic that resources exist with a certain probability. Examples for this task include the search for parking spaces, the search for vacant charging stations for electric vehicles or (for taxi drivers) the search for customers. In all of these cases, it holds that if the resource is not available upon arrival, the search must be continued to other resource locations until an available resource is found. Hence, guiding a user to the closest resource does not yield a satisfactory solution. Instead, in order to yield a sufficiently large probability of success, a route visiting several resource locations is required. A general problem of finding such routes is that there are two contrary characteristics describing the quality of a route. The first quality measure is the overall probability of finding an available resource when following the route. The second quality measure is the cost, e.g., the expected travel time or distance, until the respective resource is found. These two measures are complementary, because continuing the search to another resource location will always increase the success probability but also without exception the cost. Thus, it is not possible to minimize the cost while maximizing the probability of success.

In order to compute the success probability of a route, it is necessary to employ infor-

mation about the availability of resources at all known resource locations. In this chapter, we assume a system which collects observations on the availability (and conversely the consumption) of resources over time. These so-called long-term observations are used to infer probability distributions modeling general resource availability. Furthermore, the system provides current information about resource status at query time. We refer to this kind of information as short-term observations. For example, long-term observations correspond to the average time a parking space remains vacant or occupied. Short-term observations, on the other hand, provide information about currently vacant spaces. Depending on the scenario, the amount of short-term observations might be limited, e.g., only a limited number of parking spaces are equipped with sensors while the rest is detected and reported by roaming cars. At first glance, short-term observations might seem sufficient. However, knowing that a resource is available at the moment, does not mean that it still will be upon arrival. Thus, as time progresses, the influence of short-term observations on the probability of finding a resource decays. Long-term observations address the shortcomings of short-term observations in three ways. First, if there is no short-term observation available for a resource, the expected vacancy time of a resource can compensate for the lack of current information. Second, long-term observations can be used to predict the probability that a currently available resource will still be vacant upon arrival. Third, long-term observations can serve as an estimate for the expected occupancy time, i.e., the expected time until a consumed resource becomes available again. For example, in the parking space scenario this corresponds to the expected parking duration.

In this chapter, we present a statistical model describing a road network containing resource locations of a specific resource type. Our model incorporates both types of information described above, long-term and short-term observations. From a formal point of view, our model describes each resource location as a continuous-time Markov chain with two states, `available` and `consumed`.

Based on this model, we introduce the following query: For a given query location, compute a route for which the probability of finding an available resource exceeds a given probability threshold (e.g., 90 %) while minimizing the cost, e.g., travel time or distance. A route may be extended infinitely, and each extension adds to the success probability but also to its cost. Thus, in order to optimize one measure, we have to bound the other. More precisely, the best route may be the one with the least cost among all routes exceeding the probability threshold. Or, converesely, the best route may be the one with the highest success probability among all routes not exceeding a cost threshold.

Since our model allows for reapparance of resources, the search space of possible solutions is unlimited. However, in many applications the time frame of finding a suitable answer is rather small as users only tolerate limited answering time. Therefore, we investigate two greedy search heuristics which promise admissible results in near-interactive time. To allow the computation of optimal results, we examine a recursive backtracking approach to avoid exhaustive search. Furthermore, we propose a lower bound for the remaining increase in cost used in a highly efficient branch-and-bound solution providing optimal results. We evaluate our approach in real-world road networks on the application of finding parking spaces and on the application of finding charging stations for electric

vehicles. To conclude, the contributions of this chapter are as follows.

- A novel probabilistc model describing the availability of resources in road networks. We model resources as continuous-time Markov chains which are parametrized by long-term as well as short-term observations and allow to model the reapparance of previously consumed resources.
- A new type of query, the Probabilistic Resource Route Query with Reappearance (PRRQR).
- An approximate solution to the PRRQR employing two different search heuristics, as well as optimal solutions based on backtracking and branch-and-bound.

The rest of the chapter is organized as follows: Section 17.2 summarizes related work about similar types of queries. In Section 17.3, our probabilistic model is described, followed by the formal definition of the PRRQR. Section 17.5 describes the heuristics, bounds and algorithms to process PRRQRs. The results of our experimental evaluation are presented in Section 17.6. The chapter is concluded with a summary in Section 17.7. This research has been previously published in [67].

17.2 Related Work

In this section, we survey existing work on similar tasks. First, we will give a review of basic query types related to the one introduced in this chapter. Then, we address works which model the existence of resources in a probabilistic way. All of the following query types have the same meta-task, namely, guiding a user to a certain resource. In all of these scenarios, a database which stores the resources and their respective locations is assumed. Although this task may also be carried out in Euclidean space, we restrict ourselves to road networks, as most of the applications are traffic-related.

The simplest type of query guiding a user to the next available resource is the nearest neighbor query (NNQ). In this setting, the user specifies a location – typically his current location – as well as some type of resource. The result of the NNQ is the optimal path (fastest, shortest, etc.) to the closest location providing the resource. As NNQ are a well-explored research area, we will not go into detail on their solutions. An extension of this query are trip planning queries (TPQ) [19] (also referred to as route planning queries [14] or route search queries [83]). In this problem setting, the user specifies a selection of different resources, e.g., “ATM”, “restaurant”, “florist”, “cinema”. The result of a TPQ is the optimal path from given start to given target location visiting exactly one instance of each resource. Computing TPQs is NP-hard because in the case that each specified resource type occurs exactly once the TPQ degenerates to the Traveling Salesman Problem (TSP).

In another variation of the problem, the order in which resources are visited may be constrained, as described in [78] or [72]. For instance, if planning a date, the order of resources might be restricted by the constraints that the ATM has to be visited first and

the florist should be visited before the restaurant and the cinema. Since the task usually maintains an NP-hard subproblem, solutions to any of these problems typically employ heuristics ([14, 78]).

In the settings presented so far, the existence of a resource at its location is considered to be guaranteed. In many real world applications, however, a resource will only be available with a certain probability. If no table or seats have been reserved, not all locations of type "restaurant" or "cinema" might have the resource available. The same holds for the resource type "florist" if looking for specific flowers. In all of these cases, the requested resource is available with a certain probability (and consumed with the converse probability), i.e., prior to arrival it is not known with certainty whether the resource is available or consumed. At first glance, these kinds of uncertainty may seem congruent. However, there are significant differences which require specific modeling. We distinguish the following types of uncertainty.

Assume a surfer is looking for good waves and is considering different beaches. At every beach, the waves might be sufficient with a certain probability. If available, the resource "waves" cannot be consumed by the presence of other surfers. Thus, we refer to the probability of finding waves as *static (resource) uncertainty*. This uncertainty is independent from the time of arrival and the presence of competitors.

Now, consider a cinema where seats are a limited resource. If no seats have been reserved, the probability of finding seats for a particular show decays as screening time approaches. Since in this case a limited quantity of the resource is consumed over time, we refer to this type of uncertainty as *time-decaying (resource) uncertainty*. Contrastingly, while all tables at a certain restaurant might be occupied now, there might be one available later the same evening. In this case, the quantity of the resource might decay (or more generally: change) over time but regenerate at a later point in time. This is a significant difference to the other scenarios where revisiting resources does not yield any benefit. However, in this scenario, although the resource might have been consumed upon arrival, it might make sense to revisit after an adequate waiting time. We refer to this kind of uncertainty as *time-dependent (resource) uncertainty with reappearance*.

To the best of our knowledge, there is no previous work supporting short-term observations or taking time-dependent uncertainty with reappearance into account. Therefore, we shall now review works which incorporate static as well as time-decaying resource uncertainty. While we provide an abstract problem definition (cf. Section 17.3) and applications based upon this definition, some works focus on their application and adapt their problem definition to the respective use case. Nevertheless, these works can – with some restrictions – in many cases be extended to incorporate general resources.

The authors of [83], compute paths which guide the user along certain resources. In their follow-up work, [72], this problem is extended by ordering constraints (as in some of the examples above). Both papers present algorithms based on greedy search heuristics as well as on heuristics which minimize the expected distance until search success. In both papers, resources may have assigned success probabilities. However, these probabilities reflect static uncertainty, i.e., non-time-dependent and non-reappearing. The same holds for [30] and its follow-up work [73] where probabilistic k-route queries are introduced and ex-

amined. Here, the authors introduce a confidence value which corresponds to the existence probability of a resource at each location, also reflecting static uncertainty. Employing different heuristics, the presented algorithms find approximate solutions by clustering resource locations that maximize the expected success.

In contrast, the authors of [25] take time-dependence into account and assume a linear decay for the vacancy of parking spaces. Although this model is aimed at a specific application, it may be generalized to abstract resources. Note, however, that this model does not allow reappearance of resources which is a significant shortcoming, especially in this application. This is because a typical strategy looking for a parking space is roaming the target area until someone vacates a space, i.e., a resource reappears. The authors propose two approaches to maximizing the probability of finding a parking space. The first approach finds the optimal result, however, this is done employing full enumeration on the time-varying TSP. Due to the brute force nature of this algorithm, query processing quickly becomes infeasible with increasing number of resource locations. The second approach is an algorithm that clusters resource locations before solving a TSP on the clusters. Subsequently, the optimal solution within each cluster is searched. Based on a heuristic, this algorithm yields an approximation of the optimal result, while providing a considerable speed-up.

There are various methods, focusing on the application of taxi pickups as well as ridesharing, such as [95, 121, 92, 52]. In [95], the task is equivalent to solving a classic TSP, i.e., ordering different but fixed pickup locations such that the total distance is minimized. The authors rely on a genetic algorithm approach to solve this problem. In [121, 52, 92] the task is more complicated. Here, the task is first of all to assign cabs to a set of currently available customers. After this assignment is done, a route for picking up and dropping off the customers has to be found. Thus, only the last part of the query is related to our work. Furthermore, none of the works considers uncertainty w.r.t. customer availability.

17.3 Problem Setting

In this section, we formalize our problem setting. First, we define the graph which represents the underlying road network. Then, we introduce the probabilistic model which describes resource availability and consumption.

17.3.1 Road Network Graph

For a given road network, we let $G = (V, E)$ denote the corresponding graph, i.e., the vertices (or nodes) $v \in V$ correspond to crossings, dead ends, etc., and the edges $e \in E \subseteq V \times V$ represent directed road segments connecting the vertices. We refer to this graph as *road network graph*. Furthermore, let $c : E \to \mathbb{R}_0^+$ denote the function which maps every edge onto its respective cost, e.g., travel time or distance. If the employed cost function is not travel time, we additionally assume the travel time to be known and given by a function

$t : E \to \mathbb{R}_0^+$ (as resource availability is dependent on the time of arrival). By *route*, we mean a consecutive set of edges (possibly with cycles), i.e., $r = (e_1, \ldots, e_n)$ where all e_i are taken from the corresponding set of edges and for all $1 \leq j \leq n-1 : e_i = (u, v) \Rightarrow e_{i+1} = (v, w)$. The cost of a route $r = (e_1, \ldots, e_n)$ is defined as the accumulated cost of its edges, i.e., $c(r) = \sum_{i=1}^{n} c(e_i)$. By *path*, we mean a cycle-free route.

17.3.2 Probabilistic Model

In the following, we introduce our probabilistic model. As mentioned before, at every resource location the respective resource may either be `available` or `consumed`. However, prior to arrival at the location, it is not known which of the two is the case. Our probabilistic model has to be able to reflect all three kinds of uncertainty: static, time-decaying, and time-dependent uncertainty with reappearance. While the former two kinds of uncertainty are rather straightforward, the latter requires more work and a novel approach.

The static uncertainty of a resource is easily expressed by a random variable X which takes the values 0 or 1, representing the states `available` and `consumed`, respectively, where the probabilities of X are $\mathcal{P}(X = 0) = p$ and $\mathcal{P}(X = 1) = 1 - p$ for some $p \in [0, 1]$. For illustration, recall the example of a surfer looking for waves at a beach. Independent of the time of their arrival there will be waves with probability p.

In the case of time-decaying uncertainty, we propose modeling the probability that a resource X is available at time $t > 0$ as $e^{-\lambda t}$ for some $\lambda > 0$. Consequently, at time $t = 0$, the resource is available with probability 1, but it decreases as time progresses and asymptotically approaches 0. Or, in other words, the probability that X is consumed is the cumulative distribution function of an λ-exponentially distributed random variable. This coincides with the intuition of modeling waiting times as exponentially distributed random processes. For illustration, recall the example of buying tickets to the movies, where the probability of available seats decreases as screening time approaches.

Now, let us turn to the case of time-dependent uncertainty with reappearance which is the core of this work, as it is the only concept that can model the use cases of our experiments (parking spaces, charging stations). As before, resource availability has two states, but now, there may occur multiple state transitions at arbitrary points in time. Therefore, we propose modeling each resource as a stochastic process. The most common type of stochastic processes are Markov chains which model the transition probabilities within a system with a given number of states. When a system transitions from one state into another, the future state is only dependent on the present state. This property is central to Markov chains and referred to as memorylessness or Markov property. Markov chains either assume discrete or – as in our case – continuous time. In a discrete model, there exist equal time steps, and for each step, the probability of transitioning into another state can be computed. In a continuous-time model, the sojourn time in each state, i.e., the time until the next state transition, is perceived as a random variable itself. The notion of memorylessness extends naturally to the case of continuous time.

Thus, we model time-dependent resource availability with reappearance at each resource location as a continuous-time Markov chain (CTMC). More precisely, each resource location

r^i is now represented by a family of random variables $\{X_t^i, t \geq 0\}$ with values in the state set $\{0, 1\}$. Note that there exists a one-to-one relationship between each resource location, and its resource modeled by the respective CTMC. Thus, we denote the CTMC of each resource location r^i by X^i and denote the set of all CTMCs by $\mathcal{X}$, where $|\mathcal{X}| = |\mathcal{R}|$ and $\mathcal{R}$ is the set of resources. We may use the term *resource* for the geolocation associated with a resource as well as for the corresponding CTMC. When not clear from the context, we will state explicitly which of the two is referred to. Also, we assume the resources, more specifically their CTMCs, to be mutually independent. The independence assumption keeps the model general and applicable even if the available observations are limited.

Besides its state space, a CTMC is (under the reasonable assumption of time-homogeneity) defined by the family of transition matrices $\{P_t, t \geq 0\}$ and the (infinitesimal) generator matrix Q. Using the Kolmogorow equations, each may be computed from the other by solving the first order differential equation $P'(t) = P(t)Q$. For more mathematical details, we refer the reader to [107]. We omit the explicit calculations here and restrict ourselves to explaining the connection between $P(t), Q$ and the states of a resource.

In our case, Q is a 2×2-matrix. Its diagonal entries reflect the parameters of the random variables modeling the sojourn time of each state while the non-diagonal entries reflect the rate of transition into another state. Q has the following form:

$$Q = \begin{pmatrix} -\lambda & \lambda \\ \mu & -\mu \end{pmatrix}$$

The family of transition matrices $P(t)$ for $t \geq 0$ is defined as follows.

$$P(t) = \begin{pmatrix} \frac{\mu}{\lambda+\mu} + \frac{\lambda}{\lambda+\mu}e^{-(\lambda+\mu)t} & \frac{\lambda}{\lambda+\mu} - \frac{\lambda}{\lambda+\mu}e^{-(\lambda+\mu)t} \\ \frac{\mu}{\lambda+\mu} - \frac{\mu}{\lambda+\mu}e^{-(\lambda+\mu)t} & \frac{\lambda}{\lambda+\mu} + \frac{\mu}{\lambda+\mu}e^{-(\lambda+\mu)t} \end{pmatrix} \tag{17.1}$$

For each resource, the sojourn times of its states `available` and `consumed` are modeled as exponentially distributed random variables with parameters λ and μ, respectively. This, again, coincides with the convention of modeling waiting times as exponentially distributed. The expected value of an exponential distributed random variable $\exp(\phi)$ is $1/\phi$. Thus, the expected sojourn time in the state of availability is $1/\lambda$, and the expected sojourn time in the state of consumption is $1/\mu$. One application on which we evaluate our model in Section 17.6 is finding vacant parking spaces. In this use case, $1/\lambda$ would be the expected vacancy time, analogously, $1/\mu$ would be the time until an occupied space becomes vacant again. Of course, these parameters may be different for each resource location if enough observations for an individual estimation are available. Therefore, it is possible for each resource to have distinctly parametrized Q and $P(t)$.

Let us shortly explain how the assumed observations are used for parameter estimation. As mentioned before, our model allows to incorporate short-term as well as long-term observations. The latter are used for parameter estimation in the following way: Consider a resource X, which has an unknown expected sojourn time in the state `available` but is

assumed to be exponentially distributed with parameter λ. Given a number of observations $x = (x_1, \ldots, x_r)$, i.e., exemplary measurements of the time span during which X stays available, we can easily estimate λ using the maximum likelihood estimator. The according likelihood function is given by:

$$L(\lambda) = \prod_{i=1}^{r} \lambda \exp(\lambda x_i) = \lambda^r \exp(-\lambda r \bar{x})$$

where $\bar{x} = 1/r \sum x_i$ denotes the mean of all measurements. Differentiating the logarithmized likelihood function yields the maximum likelihood estimator $\hat{\lambda} = 1/\bar{x}$, which is simply the inverse of the mean value. The parameter μ may of course be estimated analogously.

Now, let us review some properties of the transition matrices $\{P(t), t \geq 0\}$ central to the model. Given a resource X and its sojourn time parameters λ and μ, we can compute $P(t)$ as in Equation 17.1. If we also have an initial probability distribution based on short-term observations of the states of X at time $t_0 = 0$ (denoted as π_0), we can compute the according probability distribution after an arbitrary point in time $t \geq 0$ (denoted as π_t) as follows:

$$\pi_t = \pi_0 P(t)$$

Note that $P(0) = I$ is the identity matrix (cf. Equation 17.1) which means that if no time has passed, the probability distribution of t_0 is still active. For example, if there is an observation at t_0 of X being in state `consumed`, then $\pi_0 = (0, 1)$. The consumption of resource X is certain – but only at this particular point in time. As time progresses, it becomes more likely that the state changes. Therefore, the original probability distribution π_0 (given by the observation) changes. Note that this reflects the notion of reappearance of previously consumed resources. This is expressed in the (exponentially) decaying influence of the second summands in every entry of $P(t)$ (cf. Equation 17.1). Eventually, the original observation becomes obsolete. This can be seen from the convergence of $P(t)$ as $t \to \infty$.

$$\lim_{t \to \infty} P(t) = \begin{pmatrix} \frac{\mu}{\lambda+\mu} & \frac{\lambda}{\lambda+\mu} \\ \frac{\mu}{\lambda+\mu} & \frac{\lambda}{\lambda+\mu} \end{pmatrix}$$

Asymptotically both rows of $P(t)$ are equal. This implies the initial observation has no more influence on the probability distribution of X as $t \to \infty$. For example, whether the initial observation was `consumed`, i.e., $\pi_0 = (0, 1)$, or `available`, i.e., $\pi_0 = (1, 0)$ is without significance. In any case, $\lim_{t \to \infty} \pi_t$ is the so-called stationary distribution as introduced below.

In our case, a resource X is a finite-state Markov chain where all states communicate and, thus, X has a unique stationary distribution π [107]. By definition: $\pi P(t) = \pi, \forall t \geq 0$. Upon solving this system of equations:

$$\pi = (\pi_1, \pi_2) = \begin{pmatrix} \frac{\mu}{\lambda+\mu} & \frac{\lambda}{\lambda+\mu} \end{pmatrix}$$

This is equal to the rows of $\lim_{t \to \infty} P(t)$ which supports the intuition of t having no more influence on the probability distribution. For example, if no observation for X is available,

the only unbiased assumption is the stationary distribution since it assumes the respective share of both states w.r.t. λ and μ.

Let us use all of the above in an example related to one of our applications: Let X be a CTMC modeling the vacancy of a parking space at a certain location. We make the following assumptions on the model: If `available` (meaning vacant), we expect X to be `consumed` (meaning occupied) within 5 minutes. This means, the sojourn time of state `available` is a 1/5-exponentially distributed random variable. If X is `consumed`, we expect the occupant to leave within 20 minutes. Hence, the sojourn time of state `consumed` is a 1/20-exponentially distributed random variable. As mentioned before, $P(0) = I$. Let us investigate how $P(t)$ changes as time (non-infinitely) progresses. For instance, after 1 and after 3.5 minutes:

$$P(1) \approx \begin{pmatrix} 0.8 & 0.2 \\ 0.05 & 0.95 \end{pmatrix} \quad P(3.5) \approx \begin{pmatrix} 0.52 & 0.48 \\ 0.12 & 0.88 \end{pmatrix}$$

Assume that at $t_0 = 0$ the resource X has been observed as `available`, i.e., $\mathcal{P}(X_0) = (1, 0)$. Then at $t = 1$ the space will still be `available` with probability 0.8. After 3.5 minutes this probability will have decreased to 0.52. Now, consider a different scenario where at $t = 0$ the resource X has been observed as `consumed`, then after 1 minute it is `available` with probability 0.05. After 3.5 minutes this probability will have increased to 0.12.

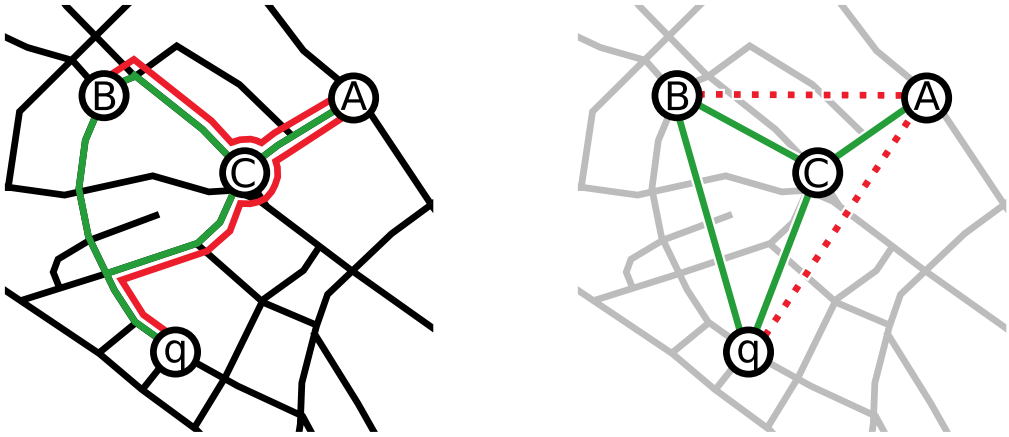
To conclude, we now have a probabilistic model on our hands which is capable of describing all three kinds of resource uncertainty introduced in Section 17.2. The core contribution of this model is its ability to reflect resource reappearance. Also, it allows efficient resource-specific parametrization by incorporating long-term and short-term observations.

17.4 Query Definition and Result Set

Now that we have defined the probabilistic model, we turn to our query and its result. Both are best described using an alternative graph, referred to as *resource graph* $\hat{G}$. Therefore, in this section, we will first define the resource graph. Subsequently, we introduce two measures which are then used to define the Probabilistic Resource Route Query with Reappearance (PRRQR) and its result.

17.4.1 Resource Graph

We assume that for a road network graph and a query node q a set of suitable resources $\mathcal{X}(q) = \{X^1, \ldots, X^N\}$ is given. In the majority of applications, only a reasonable subset of resources might qualify. For example, parking spaces should be within walking distance of the driver's destination. In this case, the query node would be the driver's position when he reaches the vicinity of his destination. An according range query to a database would then retrieve suitable parking opportunities on which a PRRQR would be executed. For every resource X^i, we assume distribution parameters λ^i, μ^i and possibly an initial distribution π_0^i as given.



(a) Optimal direct paths between resources are marked green, fastest paths with intermediate resources are marked red.

(b) Edges of the resource graph (marked continuously green) and excluded transitive connections (marked dashed red).

Figure 17.1: Illustration of a query node q and resources A, B, C in a road network graph (a) and the respective resource graph (b).

The set of vertices of the resource graph is defined as $\hat{V} := \{q\} \cup \mathcal{X}$, where $q \in V$ denotes the query node. The edges of the resource graph, $\hat{E}$, represent cost-optimal paths between resource locations in the underlying road network. Thus, each route in the resource graph can be expanded into a corresponding route in the road network (cf. Figure 17.1(a)). Let us note that even in cases where cost does not refer to the travel time, we will also compute the travel time of every cost-optimal path because it is needed to determine the success probability. Although it is possible to compute the cost-optimal path between each pair of resource locations, we will require $\hat{E}$ to contain only a minimal set of edges by removing transitive connections. A transitive connection is a path within the road network that contains at least one intermediate resource location. For example, if along the cost-optimal path from X^A to X^B resource location X^C is encountered, then X^A and X^B would not be connected in the resource graph (cf. Figure 17.1(b)). However, $\hat{G}$ would contain edges connecting X^A and X^C as well as X^C and X^B. We explicitly exclude transitive connections from the resource graph for two reasons. First, transitive links do not allow computing the success probability correctly because the intermediate resource locations are not considered. Second, the existence of transitive connections leads to the inefficient traversal of identical subpaths.

We also compute the cost-optimal paths from the query node q to all resource locations. As there is no gain in returning to the query node, paths ending at q are excluded from the computation. Algorithm 13 describes the computation of $\hat{E}$ which is illustrated in Figures 17.1. Note that in order to compute $\hat{E}$, we need to compute all cost-optimal paths within the road network graph and then prune the transitive connections. It is not possible

Algorithm 13 Computation of $\hat{E}$

Input: Query setting $\mathcal{X}(q), q$
Output: Edges $\hat{E}$ of resource graph $\hat{G}$

```
1  begin
2      Ê ← ∅
3      foreach X ∈ 𝒳(q) do
4          Compute fastest path p from q to X
5          if no intermediate resource on p then
6              Ê ← Ê ∪ p
7          end
8          Compute multi-target Dijkstra from X to all X' ∈ 𝒳 \ X in G
9          foreach fastest path p from X to X' do
10             if no intermediate resources on p then
11                 Ê ← Ê ∪ p
12             end
13         end
14     end
15 end
```

to avoid the computation of transitive connections directly.

The cost function of the road network graph G naturally extends to $\hat{G}$. Since every edge in $\hat{G}$ corresponds to an cost-optimal path in G, the cost function $\hat{c} : \hat{E} \to \mathbb{R}_0^+$ maps an edge of $\hat{E}$ to the accumulated costs of the respective path in G. Analogously, $\hat{t} : \hat{E} \to \mathbb{R}_0^+$ maps an edge of $\hat{E}$ to the accumulated time of the respective cost-optimal path in G.

Combining the above, we define the resource graph as

$$\hat{G} = \hat{G}_{(q,\mathcal{X})} := (\hat{V}, \hat{E}, \hat{c}, \hat{t})$$

Note that $\hat{G}$ holds all the query-relevant information since it contains the query node q as well as the resource locations $\mathcal{X}(q)$. Therefore, speaking of a query setting, we mean q and $\mathcal{X}(q)$ as well as the according resource graph $\hat{G}$.

17.4.2 Resource Routes and PRRQR

Relying on $\hat{G}$, we may now define the possible solutions to our query. For a given query setting $q, \mathcal{X}(q)$ and the according resource graph $\hat{G}$, a *resource route* is a route r in $\hat{G}$, starting at query node $q = X^0$. Note that by construction of the resource graph a resource route only follows optimal paths between resources. We describe a resource route as a set of edges (in $\hat{G}$), i.e., $r = (e_{r_1}, \ldots, e_{r_n})$, where $e_{r_i} \in \hat{E}$ for all $1 \leq i \leq n$. Since in our context the order of resources is of particular interest, we introduce a specific notation for it. Every edge in $\hat{E}$ connects two resources unless it starts at the query node. Hence, along a resource route r with n edges we encounter n resources. Note that these resources are

not necessarily distinct, as r may contain cycles. Let $X_r = (X^{r_1}, \ldots, X^{r_n})$ denote the n resources along r. To introduce a measure for the success probability of a complete route, we start by defining the success probability of a resource route.

Definition 17.1. *For a given resource route r along resources $(X^{r_1}, \ldots, X^{r_n})$, let t_i denote the* time of arrival *at X^{r_i}, i.e., $t_0 < t_1 < \cdots < t_n$, and let c_i denote the accumulated cost up to resource X^{r_i}. Furthermore, let $\{P(X^{r_i}, t), t \geq 0\}$ denote the transition matrices of X^{r_i} (dependent on the parameters of X^{r_i}) and let $\pi_0(X^{r_i})$ denote the initial distribution of the states of X^{r_i} (dependent on the availability of short-term observations regarding X^{r_i}). Then the probability distribution of X^{r_i} at t_i is defined as:*

$$\big(\mathbb{P}(X_{t_i}^{r_i} = 0), \mathbb{P}(X_{t_i}^{r_i} = 1)\big) := \pi_{t_i}(X^{r_i}) = \pi_0(X^{r_i})P(t_i)$$

Hence, the success probability *of X^{r_i} at arrival time t_i is the probability that resource X^{r_i} is in state* `available` *at time t_i, i.e., $\mathbb{P}(X_{t_i}^{r_i} = 0)$.*

Note that in accordance with the probabilistic model as presented in Section 17.3, we denote state `available` of a resource X by $X = 0$ and the state `consumed` by $X = 1$. Based on this definition, we define the success probability for a complete resource route.

Definition 17.2. *Let $q, \mathcal{X}(q), \hat{G}$ be a query setting and r be a resource route in $\hat{G}$. The* Success Probability *of r, denoted by $\mathbb{P}_S(r)$, is defined as the probability of the complementary event of not finding any available resource along r:*

$$\mathbb{P}_S(r) := 1 - \left(\prod_{i=1}^{n} \mathbb{P}(X_{t_i}^{r_i} = 1)\right)$$

Given the success probability, we now define a second measure which measures the effort of finding an available resource, i.e., the expected cost of a resource route.

Definition 17.3. *Let $q, \mathcal{X}(q), \hat{G}$ be a query setting and r be a resource route in $\hat{G}$. The* Expected Cost *of r, denoted by $\mathbb{E}_c(r)$, is defined as:*

$$\mathbb{E}_c(r) := \sum_{i=1}^{n} \left(\hat{c}(e_{r_i}) \cdot \mathbb{P}(X_{t_i}^{r_i} = 0) \cdot \prod_{j=1}^{i-1} \mathbb{P}(X_{t_j}^{r_j} = 1)\right)$$

where $\hat{c}(e_{r_i})$ denotes the cost of traveling from resource $X^{r_{i-1}}$ to X^{r_i}.

Relying on both measures, we define the *Probabilistic Resource Route Query with Reappearance (PRRQR).*

Definition 17.4. *Let q be a query node, $\mathcal{X}(q)$ the set of corresponding resources, $\hat{G}$ the according resource graph. Furthermore, let $0 \ll \rho < 1$ denote a probability threshold. The result of a PRRQR with threshold ρ, denoted by $PRRQR(\rho)$, is the resource route in $\hat{G}$ with minimal expected cost among all resource routes which exceed the probability threshold ρ.*

$$PRRQR(\rho) = \arg\min\{\mathbb{E}_c(r) \mid \mathbb{P}_S(r) \geq \rho\}$$

There exists a straightforward variation of the PRRQR. Instead of thresholding the success probability, one could bound the maximal cost. Given a cost threshold $\tau > 0$, the query result is the resource route maximizing the success probability while not exceeding the cost bound τ. In this case, it is usually more reasonable to employ τ as a strict bound on the maximal cost instead of bounding the the expected cost. For example, consider driving an electric vehicle with a limited remaining range. Bounding the expected distance misses the point, only bounding the actual driven distance (while maxmimizing the success probability) will provide a suitable route. This variation of the query is also examined our experiments. However, in the following, we focus on the first (and more sophisticated) case to keep the description compact.

17.5 Query Processing

In this section, we present algorithms for processing PRRQRs. First, we propose two heuristics which are employed in a greedy search that computes approximations of the optimal results. Afterwards, we will present two methods computing optimal solutions, relying on backtracking as well as on branch-and-bound. In the following, let $0 < \rho < 1$ be a probability threshold, and let $\hat{G}$ be the resource graph according to a given query setting $q, \mathcal{X}(q)$.

Let us note some aspects central to the PRRQR before going into the details of the algorithms. Given $\hat{G}$ and the query position q, then all resource routes start at q by definition. Hence, the set of possible solutions may be conceived as a search tree rooted at q where each branch is a sequence of resource locations. For a probability-constrained PRRQR with $\rho = 1$, i.e., a PRRQR requiring certainty of finding a resource, this search tree is infinite. The effect is caused by the time-dependent decay inherent in our model. A certain observation of availability of a particular resource will no longer be certain at the time of arrival. As a result, the success probability of a resource route can only asymptotically converge against 1. Even when $\rho < 1$, the search space is generally very large. This is because considering resource reappearance adds considerably to the complexity of the task. Similar to the Traveling Salesman Problem, there is no local optimality w.r.t. the resource subsequences that can be exploited. It is not possible to tell whether a resource route r can be extended into an optimal solution based on its current $\mathbb{P}_{\mathrm{S}}(r)$ and $\mathbb{E}_{\mathrm{c}}(r)$. Furthermore, it is easy to see that all permutations of the set of resources can be found in the search tree. Thus, from a theoretic point of view the problem is NP-hard. Only by setting the threshold $\rho < 1$, the search space becomes finite.

One precomputational step which all algorithms have in common is the computation of the resource graph $\hat{G}$ and its edges which constitute cost-optimal paths between resource locations in the underlying road network graph. The pseudocode for this operation is given in Algorithm 13. The set of edges $\hat{E}$ is realized as an adjacency matrix A of dimension $N + 1 \times N$, where $|\mathcal{X}(q)| = N$ is the number of suitable resources of the respective query setting. For notational reasons, we denote the query node q by X^0. The entries $a_{ij}, 0 \leq$

$i \leq N, 1 \leq j \leq N$ of A are defined as:

$$a_{ij} = \begin{cases} c(p(X^i, X^j)) & \nexists\ X^k \in p(X^i, X^j),\ i,j,k \text{ pairw. inequal} \\ \infty & else \end{cases}$$

where $p(X^i, X^j)$ denotes the cost-optimal path from X^i to X^j within the underlying road network graph. A holds the cost of all cost-optimal paths, both pairwise between resources as well as from q to any resource, if no intermediate resources are located along this path. A, however, does not hold any information about paths from any resource to q, because the query node is not a resource and thus does not yield any gain toward the query goal. In case the inherent cost is not travel time, we also compute the travel times of the cost-optimal paths. Recall the example of a query setting and its resource graph, as illustrated in Figure 17.1.

The according adjacency matrix of this scenario is given by:

	A	B	C
q	∞	$t(p(q,B))$	$t(p(q,C))$
A	∞	∞	$t(p(A,C))$
B	∞	∞	$t(p(B,C))$
C	$t(p(C,A))$	$t(p(C,B))$	∞

As mentioned before, depending on the application, the subset of suitable resource locations may be query-dependent. However, as the total set of resource locations is known prior to the query, it is possible to precompute the above adjacency matrix. If at query time a selection of suitable resources is required, the non-relevant rows and columns may simply be ignored. In the following, we consider an appropriate adjacency matrix as given. This assumption is not to the disadvantage of any of the proposed algorithms, as they all rely on the resource graph $\hat{G}$ and its edges modeled by the adjacency matrix.

17.5.1 Heuristic Solutions

In order to cope with the complexity of the PRRQR, we propose two search heuristics employed in greedy algorithms. Both heuristics aim at exceeding the given probability threshold by extending a (partial) resource route r by the "best" next resource location. The first heuristic greedily chooses the resource location which yields the best success probability upon arrival, while the second heuristic greedily chooses the resource location which yields the best trade-off between success probability upon arrival and cost to reach the location from the present one. Formally, we propose to evaluate a possible extension of resource route r along resources $(X^{r_1}, \ldots, X^{r_{i-1}})$ by one of the resource locations $\{X^1, \ldots, X^N\}$ according to the following heuristics:

(1) Extend r by X^{r_i} such that

$$X^{r_i} = \arg\max\{\mathbb{P}(X^{r_j}_{t_j} = 0) \mid 1 \leq j \leq N\}$$

(2) Extend r by X^{r_i} such that

$$X^{r_i} = \arg\max\{\mathbb{P}(X_{t_j}^{r_j} = 0)/\hat{c}(e_{r_j}) \mid 1 \leq j \leq N\}$$

In conclusion, our greedy approaches **G1** and **G2** proceed as follows: For a given query setting $q, \mathcal{X}(q)$ and a probability threshold $0 < \rho < 1$ all cost-optimal paths from q to all resource locations adjacent to q w.r.t. the adjacency matrix are computed. Then, **G1** and **G2** choose the most promising extension according to the extension strategies (1) and (2), respectively. If the success probability of the obtained resource route does not exceed the probability threshold ρ, the procedure is repeated for all resource locations adjacent to the current one w.r.t. the adjacency matrix. As soon as the success probability exceeds ρ, we have found a viable solution.

17.5.2 Optimal Results

The greedy approach described above aims at the computation of reasonable resource routes in efficient time. However, in some applications, quality is more important than efficiency. For these cases, we propose two different approaches which guarantee optimal results. We present a backtracking algorithm and a further accelerated branch-and-bound approach.

Optimal Results through Backtracking

The backtracking approach, denoted by **BT**, starts at query node q and gradually expands resource routes as long as they qualify as result candidates. A resource route disqualifies as a result candidate if it exceeds the expected cost of the currently best resource route. During the expansion, **BT** explores the search tree (rooted at q) in depth-first order. Note that this search tree is generally infinite. Consequently, it is of even greater importance to exclude resource routes from expansion early on in the algorithm. Therefore, we conduct an initialization step equal to an execution of the greedy **G2** algorithm. This generates a valid resource route in efficient time, its expected cost may be used as a first bound. We omit the initialization step here (since it is equal to the description of **G2** above). Instead, we only give the recursive procedure as presented in Algorithm 14.

The procedure `expandRecursive` is initially called with a trivial resource route only consisting of query node q and an unspecified resource (which is reset to an adjacent resource during the first run). The expected cost of the result generated by **G2** is held in a global variable M_c as an initial upper bound for the expected cost. While traversing the search tree, M_c will be tightened by finding better solutions. In line 3, candidates which do not qualify as results are excluded, while in line 6 possible result are generated (i.e., resource routes exceeding the probability threshold). The actual search tree traversal is realized recursively in lines 10, 11. If an expansion r' of a resource route r is better than the current best route along this subtree $\hat{r}$, then $\hat{r}$ is updated to r' and M_c is updated to $\mathbb{E}_c(r')$ (lines 13, 14). Thus, by sequential traversal of the search tree, **BT** returns the

Algorithm 14 Expansion step of **BT**

`expandRecursive`(Resource route r, resource X)
Data: Upper bound for $\mathbb{E}_c$, M_c
Output: Optimal resource route $\hat{r}$

```
1  begin
2      if E_c(r) > M_c then
3          return ∅
4      end
5      if P_S(r) > ρ then
6          return r
7      end
       // global variable r̂ holds currently best route
8      foreach resource X adjacent to last resource of r do
9          r' ← expandRecursive(r, X)
10         if r̂ = ∅ ∨ E_c(r') < M_c then
11             r̂ = r'
12             M_c ← E_c(r̂)
13         end
14     end
15     return r̂
16 end
```

optimal result upon termination. However, due to the exponential number of branches and the technically infinite length of the branches runtime is prone to degenerate. Therefore, we propose another algorithm which computes optimal results in significantly less time.

Optimal Results through Branch-and-Bound

Like the backtracking algorithm **BT**, this branch-and-bound approach, denoted by **BB**, relies on an upper bound for the expected cost (M_c) which is tightened as the algorithm progresses. Additionally, **BB** incorporates a forward estimation for the expected cost a route minimally needs to exceed the probability threshold ρ. The forward estimation is a lower bound for the expected cost w.r.t. a resource route and ρ. Consequently, if this lower bound exceeds the upper bound for the expected cost M_c, r can be excluded from further expansion, i.e., the respective subtree can be pruned.

Algorithmically, **BB** is similar to **BT**, except for the mentioned forward estimation. This forward estimation is incorporated into Algorithm 14 as an *if*$(m_c < M_c)$*-statement* spanning from line 8 through line 14, where m_c is the output of procedure `forwardEstimation`, as presented in Algorithm 15. Before each possible expansion of a resource route r, `forwardEstimation` is called with r and the probability threshold ρ. In lines 3-8 the parameters are set which are subsequently used to compute the lower bound for the expected travel time. $c^\star$ is the minimal cost between any two resources in $\hat{G}$. t_{now} is the

Algorithm 15 Forward Estimation of **BB**

`forwardEstimation`(Resource route r, current $\mathbb{E}_c$ bound M_c)

Output: Upper bound for the success probability of any extension of r until it exceeds M_c

```
1  begin
2      c* ← min_{e∈Ê} ĉ(e)
3      t_now ← arrival time at last resource of r
4      t_min ← min_{e∈Ê} t̂(e)
5      t* ← t_now + t_min
6      p* ← P_S(r)
7      m_c ← E_c(r)
8      while m_c < M_c do
9          p_max ← max_{X∈X(q)} P(X_{t*} = 0)
10         m_c ← m_c + (c* · p_max(1 − p*))
11         p* ← 1 − ((1 − p*)(1 − p_max))
12         t* ← t* + t_min
13     end
14     return p*
15 end
```

absolute travel time of input resource route r. $t_{\min}$ is the fastest travel time between any two resources in $\hat{G}$. Thus, $t^\star$ is the minimal possible arrival time at the next resource w.r.t. the absolute travel time of r. $p^\star$ and m_c are initialized with $\mathbb{P}_S(r)$ and $\mathbb{E}_c(r)$, respectively. Both values are updated in the while loop (lines 9-14) until $p^\star \geq \rho$, i.e., until the optimal success probability exceeds the threshold.

Now, let us investigate the operations in the while loop. First, $p_{\max}$ is defined as the maximal probability among all resources at the minimal possible arrival time $t^\star$. Note that we only allow observations to be incorporated into the model until query time. Therefore, $p_{\max}$ is monotonically decreasing in $t^\star$, and it converges against the minimal value of all stationary distributions in state `consumed`. m_c is extended by a new summand reflecting a hypothetical and optimal journey to the resource with maximal probability and minimal cost. Consequently, the success probability bound is updated to the probability of the complementary event of not finding any available resource along this optimal journey. By this strategy, in every iteration of the while loop, a journey to an optimal next resource causing minimal cost is simulated. Thus, the maximal success probability is aggregated while assuming minimal cost. We prove this in the following lemmas. We introduce the following terminology: For a given resource route r, we refer to any iteration of the while loop of Algorithm 15 as an *optimal extension*. This coincides with the above described intuition.

Lemma 17.1. *An optimal extension yields the best possible trade-off between expected cost and success probability. More specifically, let r be a resource route and $\mathbb{E}_c(r) = m_c$,*

$\mathbb{P}_S(r) = p^\star$ *as in Algorithm 15. Then the following statement holds. For any possible extension* r' *of* r *to another resource:*

$$\frac{{m_c}' - m_c}{p^{\star\prime} - p^\star} < \frac{\mathbb{E}_c(r') - \mathbb{E}_c(r)}{\mathbb{P}_S(r') - \mathbb{P}_S(r)}$$

Proof. In order to prove this lemma, we need to formulate the success probability of a resource route r differently:

$$\begin{aligned}\mathbb{P}_{\mathrm{S}}(r) &= 1 - \prod_{i=1}^{n} \mathbb{P}(X_{t_i}^{r_i} = 1) \\ &= \sum_{i=1}^{n} \left(\mathbb{P}(X_{t_i}^{r_i} = 0) \cdot \prod_{j=1}^{i-1} \mathbb{P}(X_{t_j}^{r_j} = 1) \right)\end{aligned}$$

Equality holds because the event of finding at least one available resource is the complementary event of not finding any resource in state available. Hence, $p^\star$ in Algorithm 15 may equally be set to

$$p^\star \leftarrow p^\star + p_{\max}(1 - p^\star)$$

For a given resource route r, let r' denote an arbitrary extension of r by another resource X with respective arrival time t cost of travel c. By the alternative definition of $\mathbb{P}_{\mathrm{S}}$, we have $\mathbb{P}_{\mathrm{S}}(r') = \mathbb{P}_{\mathrm{S}}(r) + \mathbb{P}(X_t = 0)(1 - \mathbb{P}_{\mathrm{S}}(r))$.
Before the while-loop in Algorithm 15, $\mathbb{E}_{\mathrm{c}}(r) = m_{\mathrm{c}}$ and $\mathbb{P}_{\mathrm{S}}(r) = p^\star$ are initialized. Furthermore, $c^\star, t^\star, p^\star, p_{\max}, m'_{\mathrm{c}}$ are as after the while-loop, i.e., they denote the optimal extension of route r.

Now, we show our claim:

$$\begin{aligned}\frac{{m_{\mathrm{c}}}' - m_{\mathrm{c}}}{p^{\star\prime} - p^\star} &< \frac{\mathbb{E}_{\mathrm{c}}(r') - \mathbb{E}_{\mathrm{c}}(r)}{\mathbb{P}_{\mathrm{S}}(r') - \mathbb{P}_{\mathrm{S}}(r)} \\ \Leftrightarrow \frac{\mathbb{E}_{\mathrm{c}}(r) + c^\star p_{\max}(1 - p^\star) - \mathbb{E}_{\mathrm{c}}(r)}{p^\star + p_{\max}(1 - p^\star) - p^\star} &< \frac{\mathbb{E}_{\mathrm{c}}(r) + c\mathbb{P}(X_t = 0)(1 - p^\star) - \mathbb{E}_{\mathrm{c}}(r)}{\mathbb{P}_{\mathrm{S}}(r) + \mathbb{P}(X_t = 0)(1 - \mathbb{P}_{\mathrm{S}}(r)) - \mathbb{P}_{\mathrm{S}}(r)} \\ \Leftrightarrow \frac{c^\star p_{\max}(1 - p^\star)}{p_{\max}(1 - p^\star)} &< \frac{c\mathbb{P}(X_t = 0)(1 - p^\star)}{\mathbb{P}(X_t = 0)(1 - \mathbb{P}_{\mathrm{S}}(r))} \\ \Leftrightarrow c^\star &< c\end{aligned}$$

This proves the inequality. As this holds for every iteration of the while-loop, this proves the claim. □

Lemma 17.2. *Let* r *be a resource route. The forward estimation of the expected cost as computed by Algorithm 15 is indeed a lower bound.*

Proof. This follows from the following properties:

(i) The number of optimal extensions needed until r exceeds the probability threshold is at most the number of actual extensions needed.

(ii) Every optimal extension of r yields a better trade-off than an actual extension.

(iii) No sequence of actual extensions of r can exceed the probability threshold while yielding a lower expected cost than the sequence of optimal extensions chosen by Algorithm 15.

(i) follows directly from the definition of $p^\star \leftarrow 1 - \big((1-p^\star)(1-p_{\max})\big)$. In every optimal extension, $p^\star$ is increased by the maximally possible value. Therefore, no other sequence of extensions can yield a faster increase. (ii) is the statement of Lemma 17.1. Finally, (iii) follows from both, (i) and (ii). □

Note that all of the above is easily applied to the case where instead of minimizing the expected cost w.r.t. a probability threshold we maximize the probability w.r.t. an absolute cost bound (as introduced at the end of Section 17.4.2). For example, consider the backtracking expansion Algorithm 14. Instead of dismissing (storing) a route r if $\mathbb{E}_c(r) > M_c$ ("$<$" holds), in the complementary scenario, a route r is dismissed (stored), if $c(r) > M_c$ ("$<$" holds). Similarly for the forward estimation presented in Algorithm 15. The expected cost $\mathbb{E}_c(r)$ is to be replaced with the absolute cost $c(r)$. As long as the absolute cost bound is not exceeded, optimal path extensions are simulated, adding maximal success probability to the path. When the cost bound is exceeded and the maximal current best success probability is not surpassed, the search tree can be pruned. If, on the other hand, the success probability is surpassed by the optimal path extension, the path (and its subtree in the search tree) qualifies as a candidate. As for the theoretic arguments, they apply analogously.

In conclusion, we have presented four algorithms for solving the proposed PRRQR in this section. **G1** and **G2** follow a greedy heuristic to produce approximate results, while **BT** and **BB** produce exact results. **BB** is an extension of **BT** which makes use of a lower bound forward estimation of the expected cost. In the above lemmas, we have shown correctness of the proposed bound.

17.6 Experimental Evaluation

We evaluate our model and our algorithms on settings in real-world road networks extracted from OpenStreetMap (OSM) using the MARiO framework [58]. All experiments were conducted on a desktop computer equipped with an Intel Core i7-3770 CPU and 32 GB RAM, running Java 1.64 (64-Bit) on Linux 3.13 x86_64. Different algorithms are always tested on the same randomly generated scenario before comparing results. Runtime evaluations are based on Java's nanotime clock and performed for each algorithm individually excluding preliminary steps like graph population and building of the adjacency matrix. Computation of the adjacency matrix around 250 milliseconds, for standard settings in the *Parking* and *Charging* scenarios, respectively. Note that all cost-optimal paths

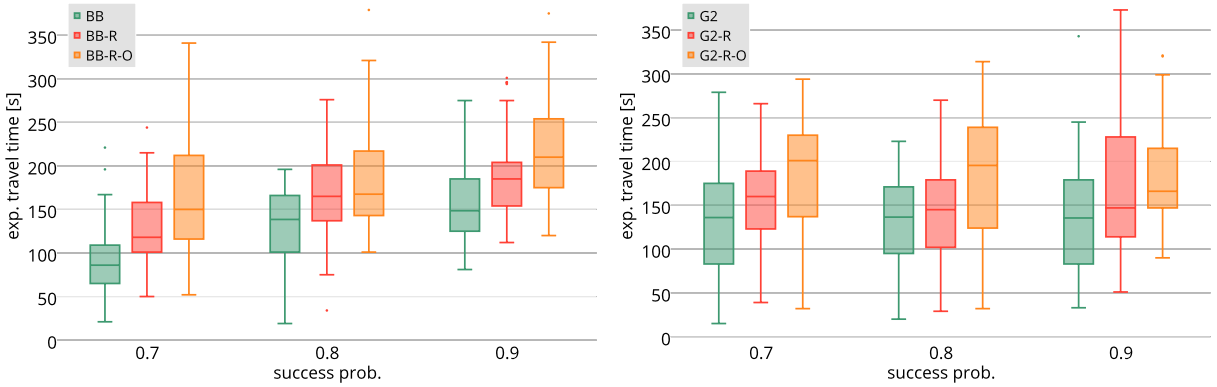
were computed using Dijkstra's algorithm. Choosing a different routing algorithm and/or employing a speed-up technique would yield the same benefit for all compared approaches. Modifying the path computation algorithm is an easy task, however, on a city scale (which the applications require) this would hardly yield any computational benefit. We present experiments for two realistic applications:

- *Parking* scenario (located in Bamberg, Germany): Given a probability threshold, we provide a route along parking spaces which surpasses the threshold and minimizes the expected travel time. This scenario is based on ground truth extracted from OSM metadata.
- *Charging* scenario (located in Brussels, Belgium): Given a query position and a range limit (as used by electric vehicles), we provide a route along charging stations not exceeding the range limit and maximizing the success probability.

Note that these scenarios are complementary w.r.t. the criterion which is bounded and the criterion which is to be optimized, as explained in Section 17.4.1. While *Charging* relies on an hard numeric bound (remaining range), *Parking* relies on the more sophisticated expected value bound. Therefore we choose *Parking* as our main scenario. We will not present all charts for both scenarios, however noting that corresponding charts show the same behavior.

17.6.1 Parking Scenario

We generated the following test cases on the road network of the city of Bamberg, Germany, containing approximately 10K nodes and 20K edges as well as nearly exhaustive metadata regarding parking spaces. For every test case, a target node is randomly drawn from all road network nodes of degree ≥ 1 within a three kilometer radius from the city center. Then, an isochrone of 800 meters walking distance is computed around the target. Let N be the number of resources (according to the ground truth) within the isochrone. In our experiments, resources are rather dense, i.e., $25 \leq N \leq 100$. Subsequently, the query node q is randomly drawn from all nodes within the isochrone. This corresponds to the use case where we expect the user to trigger the query when they are in the vicinity of their target. Finally, $M \leq N$ observations of resource availability are randomly distributed among the resource locations, and the respective sojourn times in the states `available` and `consumed` are set. For reasons of clarity, in our experimental settings the sojourn times are set to the same configurations for all resources. We assume the expected time a space stays vacant (`available`) to be 3 minutes and the expected time a space stays occupied (`consumed`) to be 90 minutes. Note that the resources could easily be parametrized separately to model differently volatile resources. In this scenario, a probability threshold is given, and as a cost function we use travel time as formalized in Definition 17.3. The optimal resource route is the one with the least expected travel time among all resource routes with a success probability exceeding the threshold.

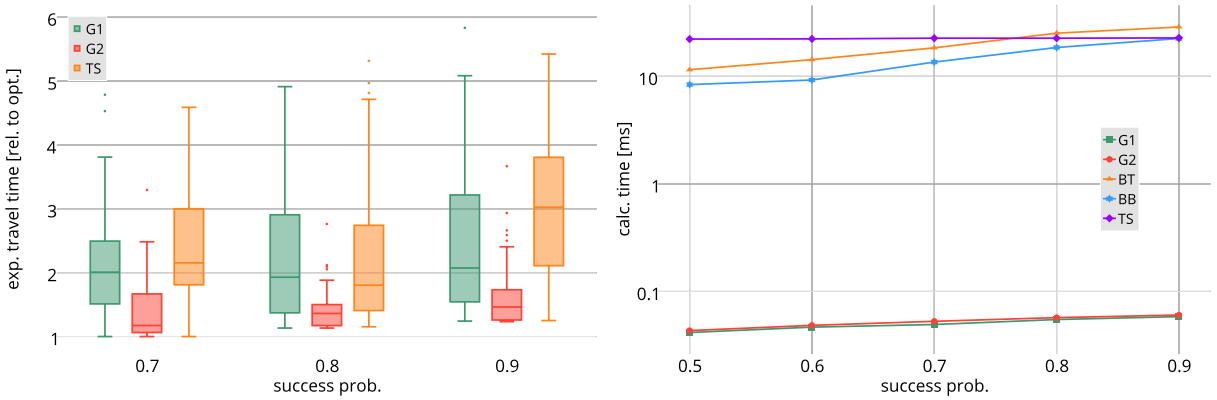


(a) **BB**-related algorithms (b) **G2**-related algorithms

Figure 17.2: Illustration of the influence of model complexity on the quality of results in the *Parking* scenario.

First, we want to evaluate how much the additional information held by our probabilistic model improves result quality. Recall that our model supports reappearance and incorporates short-term observations, two properties that distinguish this work from others. In order to prove that the gain in result quality outweighs the gain in model complexity, we trim our algorithms **BB** (branch-and-bound) and **G2** (greedy approach with probability per cost heuristic) to partially ignore the information provided by the underlying model. In a first step, we disable the possibility of resource reappearance, we denote these approaches by **BB-R** and **G2-R**, respectively. This means, **BB-R** and **G2-R** proceed like their respective counterparts but do not revisit resources which have previously been observed as `consumed`. This corresponds to a simpler probabilistic model without the feature of resource reappearance. In a second step, we additionally disable short-term observations. We denote these approaches by **BB-R-O** and **G2-R-O**. They proceed like **BB-R** and **G2-R**, respectively, but additionally ignore any short term observations. Hence, the variations emulate an even simpler model which only allows static uncertainty, as used in [30], for example.

The results for the **BB**-related and the **G2**-related algorithms are shown in Figures 17.2(a) and 17.2(b), respectively. Both figures depict the same settings. It is obvious that requiring a greater probability threshold results in resource routes with longer expected travel time. Therefore, the overall increase in expected travel time is consequential. Both figures clearly show that the algorithms which rely on greater information, i.e., use a more complex model, yield better results. Figure 17.2(a) visualizes the results of the branch-and-bound approaches which are optimal w.r.t. to the information available. As claimed, **BB** on average outperforms **BB-R**, its counterpart which does not allow reappearance by at least 20 percent. **BB-R**, in turn, outperforms its counterpart which does not incorporate short-term observations, **BB-R-O**. This supports the previously made claim that



(a) Expected travel time relative to optimal solution

(b) Calculation time

Figure 17.3: Illustration of quality as well as efficiency of all algorithms in the *Parking* scenario.

resource reappearance and short-term observations indeed improve the quality of results. From Figure 17.2(b) we observe, that simpler algorithms also benefit from the additional information contained in the model. Comparing the two figures, **BB**-algorithms of course yield better results than **G2**-algorithms and do so with significantly less variance than the greedy approaches. This is because the heuristics rely on chance in the form of beneficial problem settings in order to generate near-optimal results.

Next, let us investigate the performance of the algorithms presented. As mentioned before, there exists no work which is fully comparable. However, as the PRRQR is related to the Traveling Salesman Problem (TSP) and clustering is commonly used to approximate the TSP (as in [25]), we use this concept to implement an approximative comparison partner denoted by **TS**. It is important to mention that **TS** does not support resource reappearance, because otherwise the heuristic would not visit sufficiently many distinct resources to achieve a comparable success probability. Before we present the results, let us explain how **TS** proceeds. In a first step, **TS** conducts a k-medoid clustering on the set of all resources, where experimentally $k = 6$ has proved adequate. Subsequently, a TSP on the cluster medoids (starting at the query node) is solved. Then follows the actual resource route computation. It starts at the query node and computes the cost-optimal path to the first medoid. In the respective cluster, a greedy depth-first search (starting at the medoid) is conducted, returning an approximation of the cluster-internal cost-optimal path. From the last resource of the cluster we compute the cost-optimal path to the next medoid. This procedure is continued until the resource route exceeds the given probability threshold. **TS** serves as an algorithmic competitor based on a simpler probabilistic model but with a solid heuristic that has proven efficient when solving TSP-related problems. Note that the cost-optimal paths between all resources are precomputed in order to make the comparison to our algorithms (which use the precomputed adjacency matrix) fair.

We compare **TS** to all algorithms introduced in Section 17.5, i.e., the two greedy

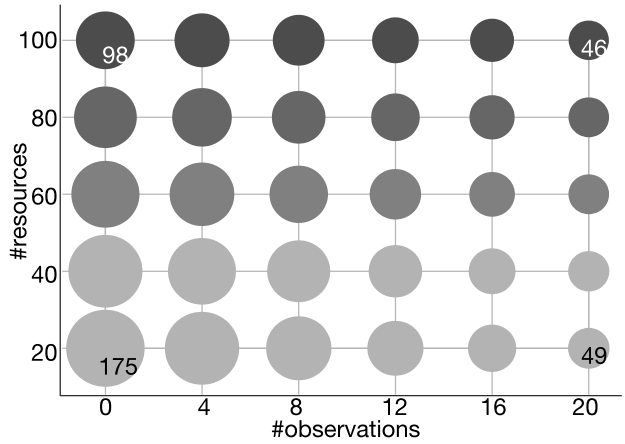
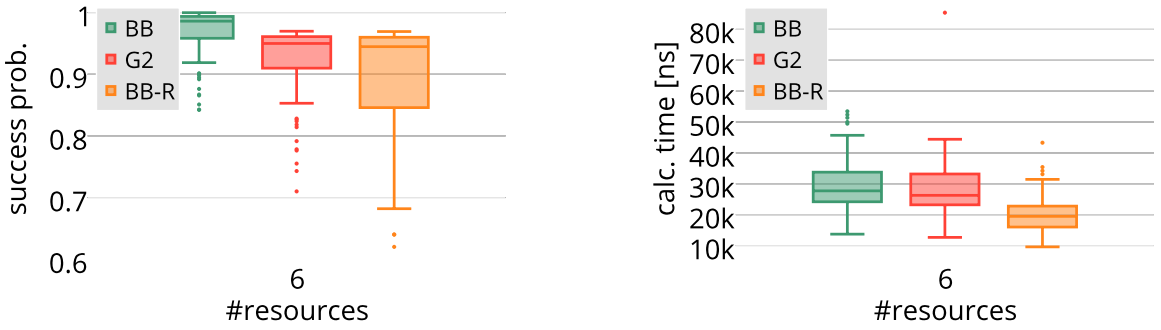


Figure 17.4: Influence of the number of resources and the number of observations on the expected travel time, i.e., result quality (for a probability threshold of 0.7). (The gray shades only serve an aesthetic purpose.)

approaches **G1** and **G2** as well as the exact solutions **BT** and **BB**. Figure 17.3(a) shows the quality of the results produced by the approximative algorithms, i.e., **G1**, **G2**, and **TS**. Their respective expected travel times are given relative to the optimal results. The higher the probability threshold, i.e., the more complex the task, the greater the discrepancy between optimal results and approximation. Although **G2** relies on the rather simple probability-to-cost ratio heuristic, it significantly outperforms its comparison partners. While **G2** yields near-optimal results in the easier settings, the optimal solutions in the most elaborate scenario (probability threshold 0.9) undercut its expected travel times on average by about 30 percent. This gain in quality, however, comes at the price of calculation time, as depicted in Figure 17.3(b). This illustration shows the averaged runtimes of all algorithms when increasing the required probability threshold. The greedy approaches generate results in almost interactive time, while **BB**, **BT**, and **TS** are around two to three orders of magnitude slower. However, it is important to note that two orders of magnitude only correspond to around 100 ms of calculation time. Comparing the exact algorithms, we observe that **BB** outperforms **BT** which can be attributed to the forward estimation. The competitive approach **TS** performs in constant time of about 150 milliseconds (for the same number of resources), however generating the worst results.

Finally, we want to explore how volatile the results are w.r.t. the model parameters. We restrict ourselves to the optimal solution provided by **BB**, seeing as the quality ratio of optimal to approximative solutions has been explored above. Figure 17.4 depicts the influence of the number of parking spaces relative to the number of short-term observations of vacant parking spaces. Each circle in the plot corresponds to a pair of parameter values, and the diameter of each circle represents the average expected travel time of this scenario in seconds, as do the numbers in the corner circles. The result shows the expected behavior that with an increasing number of parking spaces, expected travel time decreases.



(a) Success probability

(b) Calculation time

Figure 17.5: Illustration of quality as well as efficiency of selected algorithms in the *Charging* scenario.

Furthermore, for any given scenario, it can be seen that the increased amount of short-term observations also reduces the expected time until a vacant space is found. Similarly expectable behavior is observed when varying the sojourn time parameters $1/\lambda$ and $1/\mu$, therefore further charts are omitted.

17.6.2 Charging Scenario

For *Charging* we generated test cases on the road network of the city of Brussels, Belgium, containing approximately 30K nodes and 67K edges. For every test case a query node is randomly drawn from all road network nodes of degree ≥ 1 within a 6 kilometer radius of the city center. Then, an isochrone of 6 kilometers is computed around the query node, wherein 6 resource locations are randomly drawn. We have evaluated other numbers of resources but the results do not reveal additional information and are therefore omitted here. Compared to *Parking*, where nearly every street holds at least one resource, this scenario models resource scarcity. Again, if N denotes the number of resources (6 in our experiments), then $M \leq N$ observations of resource availability are randomly distributed among these resource locations. The expected time a charging station remains vacant (`available`) is set to 30 min, and the expected time it remains occupied (`consumed`) is set to 50 minutes. As before, every charging station may be parametrized individually, however it is omitted here for reasons of clarity and lack of ground truth. In this scenario an absolute distance bound of 6 kilometers is given, imitating the remaining range of an electric vehicle with low battery. Note that in contrast to *Parking*, this bound is strict and cannot be exceeded. Every algorithm computes a route with an absolute distance of 6 kilometers, the optimal resource route is the one with maximal success probability.

In a first setting, we compare the result quality of our exact algorithm **BB**, our greedy solution **G2** and **BB-R** (cf. Figure 17.5(a)), the branch-and-bound variation which does not incorporate resource reappearance. Additionally to the scarcity of resources, the re-

maining range (6 kilometers) is only double the average distance from query node to the next resource (3 kilometers, as resources are distributed uniformly within the isochrone). Due to these tightened constraints, superiority of the optimal results generated by **BB** becomes more apparent. In almost three out of four runs, **BB** yields a success probability of over 95 percent, outperforming **G2** significantly. While the greedy heuristic worked well before, it is now easily lead down a considerably less beneficial branch of the search tree. Nonetheless, **G2** still produces slightly better results than **BB-R**. Again, this advocates our model which supports resource reappearance. Even an approximative approach on our model yields better results than an exact algorithms on a less sophisticated model due to lack of information. Of course, a simpler model needs less intricate function evaluations. In our case, however, the difference is merely a matter of microseconds, as depicted in Figure 17.5(b).

To sum up, we have empirically proved the benefit of our probabilistic model. It improves the quality of results by incorporating richer information, especially for complex but also for simpler tasks while not causing significant computational overhead. On the contrary, our greedy approaches deliver competitive results in near-interactive time while our branch-and-bound approach yields optimal solutions in efficient time.

17.7 Conclusions

This chapter examined the problem of probabilistic route queries in road networks where the user is guided along a set of resources in order to maximize the probability of encountering an available resource. The goal is to find a route with minimal expected cost among all routes exceeding a given probability threshold. We proposed a novel approach in which resources are modeled as continuous-time Markov chains with two states, `available` and `consumed`. In contrast to similar problems, our model allows for consumed resources to reappear and takes short term as well as long term observations into account. The introduced query, referred to as PRRQR, is NP-hard and has an unlimited search space.

To solve this problem, we proposed approximative as well as optimal solutions. Two different search heuristics are employed in a greedy algorithm to achieve a trade-off between accuracy and calculation time. Furthermore, solutions using backtracking and a branch-and-bound approach provide optimal solutions in efficient time. We demonstrated the superiority of our model as well as the efficiency and effectiveness of our algorithms on two realistic applications. The first is the search of a vacant parking space, and the second is the search for a vacant charging station for electric vehicles.

Chapter 18

EasyEV: A Demonstration

18.1 Introduction

With increasing air pollution, limited fossil fuel supply and growing pressure to politically enforce reduction of CO_2 emissions, it seems, the days of common fossil fueled engine cars are numbered. Aside from cars which run on alternative fuels, like natural gas, alcohol or rapeseed oil, there are a number of vehicles relying on fuel cell technology, but most mass-produced non-combustion vehicles are electric. While experts' opinions differ on the number of EVs currently on the road and even more on projected sales, they declare unanimously that the market is growing with great potential. Quite a few countries incentivize the acquisition of EVs, especially in Asia (e.g., China, India) and in Europe (e.g., France, Denmark) and/or invest in infrastructure like building fast charging stations (e.g., Estonia) or generally subsidize research in the area. While it is not certain that electric mobility will be the heir to the combustion engine, almost all manufacturers surge onto the EV market with their own models and only very few seem to ignore the trend.

Reasons for a possible triumph of electric mobility are diverse: Electric infrastructure exists in all developed countries, electricity is relatively cheap and less prone to fluctuations of the market than oil, the CO_2 footprint is believed to be ecologically justifiable, the efficiency of the engines is solid and ideas of EVs as energy storage in a smart grid or as emergency generators fire scientists' imagination. However bold these visions may be, in reality, electric mobility faces very simple limitations. The first and most important limitation is the range of EV, which varies mostly from 100 to 150 kilometers, excluding some exceptions like the Tesla Roadster which – according to the manufacturer – travels up to 400 kilometers per charge. In addition, the energy consumption and therefore the range of an EV is dependent on the driving style (as with common combustion engines) but also on factors like heating, air conditioning, sound system and headlights. The second limitation are the rather long recharging times. In contrast to refueling a car, a full recharge can take several hours.

Despite these drawbacks, EVs have great potential to fulfill the mobility needs of urban populations. This is mainly because the average trip in a city is less than 30 kilometers long

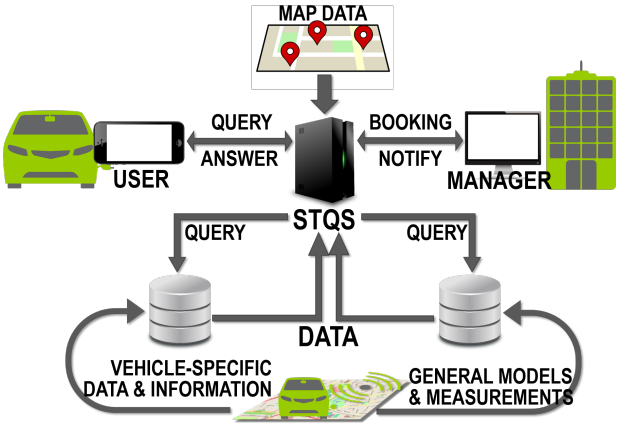


Figure 18.1: System architecture of EasyEV. The manager is informed about anomalies concerning his fleet. The user can query the system, his on-board unit provides information to the system.

and cars are actually parked most of the time. These two aspects can be taken advantage of. Even so, when in a setting where EVs are shared among several people, for instance by the employees of companies residing in the same office building. The business model of a shared fleet of EVs is examined in the project Shared E-Fleet[1]. One goal of this project is to examine how spatio-temporal information systems can counteract the drawbacks and how the use of real-time information can improve the efficiency of a fleet of EVs. For instance, by monitoring the vehicles and keeping track of their remaining range limits, an automated booking system can inform the subsequent user of an expected delay or – even better – assign the user a different vehicle. If the booking schedule is kept stable, charging times of vehicles can be optimized, yielding low electricity costs and reliably projectable charging processes.

In this chapter, we present EasyEV, our prototype for supporting drivers and fleet managers alike with spatio-temporal information services. EasyEV has been employed in pilot projects from July 2014 until September 2015 in four different German cities involving seven BMW i3. Each car was equipped with a custom on-board transmission unit providing data like position, remaining range, charging status and active electric devices. By evaluating this data, we were able to detect anomalies, in order to inform the fleet manager about critical situations like expected belated returns or expected exceeded range limits. In addition to these so-called monitoring features, EasyEV also supports the driver with multiple functionalities, such as providing directions to the nearest charging station, visually informing about reachable destinations and computing alternative paths.

[1] www.shared-e-fleet.de

18.2 System Architecture

EasyEV runs as a platform-independent system which is connected to the fleet management component and to the driver via app as well as to a real-time car database and an aggregated sensor information database. While the driver queries the system directly, the fleet management registers its vehicles and bookings with the system and is informed if anomalies occur. All connections are realized as platform-independent web service interfaces. Thus, the features of EasyEV are easily reused in a system requiring similar functionalities. At the heart of EasyEV lies a spatio-temporal query system (STQS) which answers directly to queries from users (routing features) and indirectly executes all recurrent tasks (monitoring features). For the STQS to answer queries instantly, the road network index has to be kept in memory. Hence, the STQS is modeled as a standalone server entity, while all communication with the server is handled by web services. The architecture is illustrated in Figure 18.1. The STQS relies on OpenStreetMap (OSM) data for all road networks which we model as multicriteria network.

For the pilot projects, specific on-board units have been developed which collect data directly from the car computer as well as from additional built-in sensors, most prominently GPS and temperature sensors. The vehicle-specific data is fed into a database (see Figure 18.1). The environment-related data is aggregated and used to train time-dependent models reflecting outside influences on city-scale traffic, such as congestion or icy roads. We rely on external service providers feeding this kind of information into a data storage from where the STQS can query current influence factors. In order to be independent of the number of pilot test vehicles used during the demo presentation, we also simulate driving behavior according to the recorded trajectories. This enables us to show actual historic driving data, real-time movement and simulated trajectories ensuring a high frequency tasks.

In addition to the STQS, EasyEV offers a GUI to visualize trajectories, traffic-influencing factors and query results. OSM data is easily displayed using the open source library Leaflet[2] combined with the map design tool Mapbox[3]. The GUI is browser-based and thereby platform-independent. It is dependent on the STQS-specific interfaces but can run on an entirely different machine. Hence, the architecture of EasyEV follows the model-view-controller principle, which facilitates portability and reusability.

18.3 Demo Features

We distinguish between routing features and monitoring features. The former are especially interesting from a user perspective, while the latter are particularly interesting from a fleet manager (operator) perspective. Note that in the pilot projects, monitoring and routing features are separated, as users do not have access to the trajectories of other users, while the monitoring system does not need access to all routing features.

[2] www.leafletjs.com
[3] www.mapbox.com

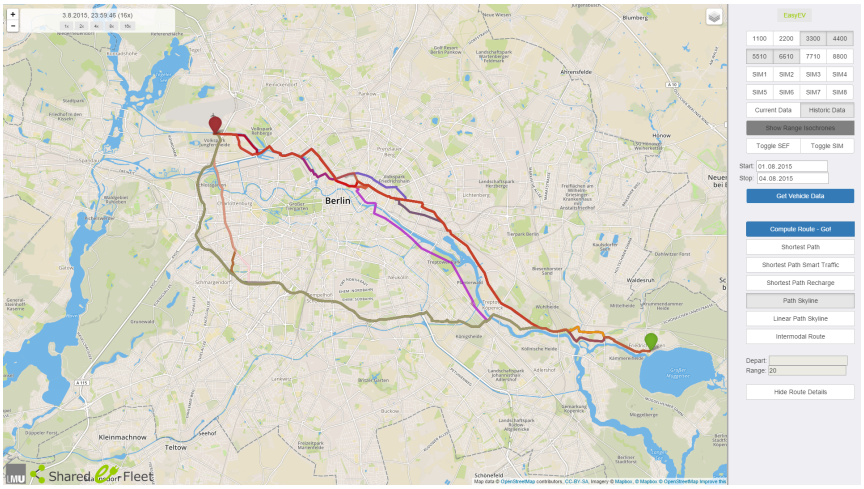


Figure 18.2: Illustration of a set of pareto-optimal paths as displayed by EasyEV.

Let us first introduce the routing features. As an elementary functionality, EasyEV supports shortest path searches given start and target nodes as well as w.r.t. predefined combinations of cost criteria. This has proved useful, as some users prefer a single path over a set of alternatives. EasyEV also supports state-of-the-art algorithms for the computation of the linear and conventional path skyline, as presented in Chapters 6 and 8, as depicted in Figure 18.2. In both cases, a result list is displayed which the user may browse.

As mentioned before, EasyEV does not only rely on static edge costs, constituted by the underlying road network, but also incorporates sensor information, as derived from data acquired by mobile (e.g., the vehicles themselves) and stationary sensors (e.g., traffic loop sensors). In our demo we focus on the influence factors road ice and traffic delay, however noting that, given a wider array of data providers, other factors could easily be included. These factors either impact specific roads (traffic delay) or whole areas (road ice). Both variants are displayed on the map. Given this kind of data, EasyEV computes paths avoiding affected roads or regions, depending on the duration of the delay or on the severity of the road icing.

In addition to sensor data, EasyEV employs static meta-information. Incorporating locations of public charging stations, EasyEV allows for ad-hoc queries in order to intercharge when on the road, e.g., during a customer meeting. For the demo, this information is retrieved from providers like Open Charge Map[4] or Plugshare[5].

Now, let us turn to the monitoring functionalities which serve the purpose of real-time as well as retrospective fleet analysis. In addition to the recorded trajectories, we also simulate trajectories in order to ensure sufficiently high data density. Trajectory simulation is done by computing (some pareto-optimal) path and sending positions along this path according to some randomly drawn travel time from a distribution around the speed limit.

[4] www.openchargemap.org
[5] www.plugshare.com

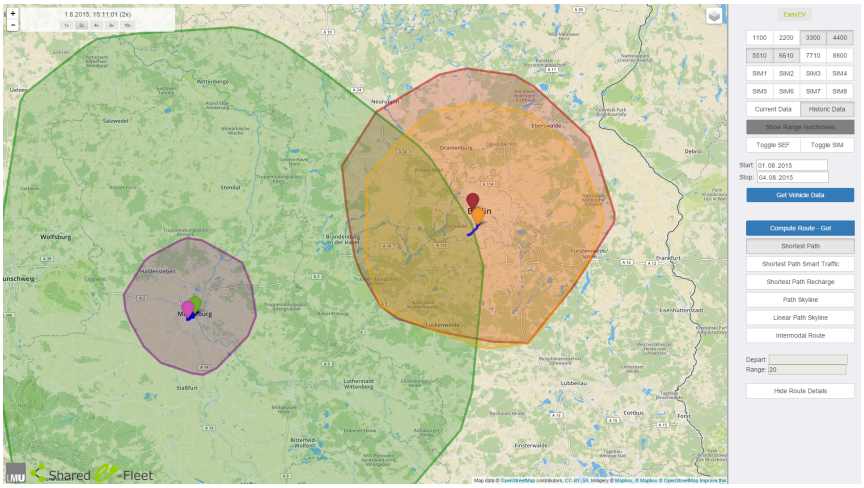


Figure 18.3: Range limits of pilot project vehicles, illustrated as isochrone regions, as displayed by EasyEV.

While the simulated trajectories are computed by the back end, all real-time and historic trajectories are retrieved from the vehicle database via web service. Most pilot test vehicles send their data every minute, some vehicles send their data every ten seconds. It is possible to display the roaming cars either in real-time or for defined past time-spans playing at variable speeds.

The STQS informs the fleet manager about anomalies which occur during operation. There are four types of notifications which originate from the use cases covered in the project. The first type of notification informs the manager of *illegal movement*, meaning that a registered vehicle is moving without a valid booking. If a vehicle is not moved throughout the whole period of a booking, the manager receives a *no show* notification, as this might result in different billing. The other two notifications only occur during driving (with a valid booking). First, if a return within the booking period is unlikely, the fleet manager is informed of an expected *delay*. This is the case, if even the fastest path from the vehicle's current location to the booking-specific return station is longer than the remaining booking period. Second, if even the shortest path from the current location to the return station is longer than the currently remaining range of the vehicle, the fleet manager (and the driver) is informed of an expected *malfunction*.

In addition to the geo-positions at the respective timestamps, various other information is collected by the on-board unit and fed into the database. For example, the total number of kilometers driven, whether the passenger seat is occupied, whether wiper or lights are on, the current energy consumption and the remaining distance. EasyEV displays all this information when hovering over the geo-position of a vehicle. Of all the data collected, the remaining distance is of particular interest, seeing as exceeding the range limit results in vehicle deficiency and often significant rescheduling efforts. Hence, we incorporate the feature of displaying isochrone diagrams around pilot test EVs, as displayed in Figure 18.3.

18.4 Conclusions

EasyEV is a querying and monitoring system for fleets of EVs, which has been actively used in a pilot project investigating the potential of smart systems employed in optimization of EV fleets. EasyEV offers features for fleet managers and drivers alike. All computations are handled by a spatio-temporal query system, while the GUI is browser-based and the communication is handled by web services. Hence, EasyEV ensures high functionality and at the same time great flexibility and reusability.

Chapter 19

Summary

This part illuminated several real-world traffic problems. Solutions to these problems aim at making traffic more efficient. At the same time, the presented solutions aim at providing personalized and effective results, e.g., to increasing convenience and save time for the driver. The key concept of the approaches presented is to cope with growing traffic and transportation demands by incorporating sensor data. This data may come from various sources, among them mobile as well as stationary sensors, complex as well as simple sensors and custom on-board units. By taking this kind of information into account, it may become possible to handle the ever-growing complexity of traffic, especially in urban areas. In addition to introducing algorithmic solutions to real-world traffic problems, this part also gave insight into EasyEV, a framework which came to use in a pilot project for the efficient management of fleets of electric vehicles.

Part V

Concluding Remarks

This thesis is centered around modern traffic networks from the perspective of computer science. Traffic is not only an area of application for many historic and current graph-based problems. It also raises new questions and assigns new tasks to researches. These new tasks are due to growing demands for solutions to ubiquitous traffic problems as well as due to increasing requirements in terms of efficiency, functionalities and input data. For instance, from a collective point of view, traffic flow needs optimization in order to reduce congestion and its negative consequences such as pollution, overstrained infrastructure and wasted money and time. From a personal point of view, modern navigation systems are expected to work efficiently and accurate while providing a wide array of convenient features which is expected to grow constantly. From a practical point of view, different data sources like sensor data, crowdsourced information and curated content have to be combined to meet these requirements. From a theoretical point of view, means for incorporating this data have to be developed, new queries which fulfill modern mobility requirements have to be introduced and existing algorithms have to be optimized to satisfy the demands. In this thesis, some of these aspects are tackled.

Optimal Paths in Multicriteria Networks Part II of this thesis addresses multicriteria networks in which every edge is assigned a cost vector. Every entry of such a vector may represent a different cost criterion. This enables modeling road networks with multiple optimization criteria which in turn allows for routing decisions that take more than one criterion into account. The gain in diversity comes at the cost of complexity. In single-criterion networks the optimal path is usually the path which minimizes the accumulated cost. This notion of optimality does not extend to multicriteria networks. The path skyline, also called the set of pareto-optimal paths, fills this void. However, the path skyline is exponential in the input size [61], making computation widely infeasible in traffic networks which meet current size requirements. Part II addresses this problem in two ways. First, we provide an efficient means for the computation of lower bounds in multicriteria networks. As in the $A^\star$ algorithm, lower bound forward estimations may be employed for goal-direction, thereby limiting the search space and reducing computation time. In fact, most path computation algorithms rely on such bounds. The algorithm provided in Chapter 6 computes optimal lower bounds, i.e., corresponding to an existing shortest path, while outperforming the state-of-the-art approach. Employing these bounds in a path skyline query yields significant speed-up such that path skyline queries in large multicriteria networks become feasible. Second, in Chapters 7 and 8, an alternative definition of optimality is illuminated, yielding the linear path skyline which in general contains less paths and may therefore accelerate computation and provide clearer results. Efficient algorithms to compute the linear path skylines are introduced, first for networks with two criteria, then for arbitrary criteria. Furthermore, a further restriction of the result set is given, the so-called ε-linear path skyline which intensifies the effects of the linear path skyline. In conclusion, the research presented in Part III has facilitated path computation in multicriteria networks. Our technique for the computation of optimal lower bounds may be used in existing and future path computation algorithms. The linear path skyline extends

the notion of optimality in multicriteria networks into a more user-friendly and efficient direction. In the future, we want to apply the developed methods to other problems. For instance, applying the techniques presented in Chapter 8 to the problem of mining driving preference in multicriteria networks (cf. Chapter 16). Another interesting aspect is the effect of time-dependent cost criteria on the presented methods.

Network Enrichment and Paths in Enriched Networks Part III of this thesis investigates the methodology for using crowdsourced data as a source for alternative cost criteria and introduces the related Twofold Time-Dependent Arc Orienteering Problem (2TD-AOP). Most established navigation services or systems rely on "hard" criteria such as distance, or travel time which are minimized in order to compute optimal paths. Existing approaches to capture "soft" criteria such as scenicness or excitement are rare. Some approaches are slowly finding their way into commercial solutions. In these cases, the content is curated by experts and the routing suggestions correspond to a particular scope of application. In contrast to hard criteria where the optimal path minimizes the cost, this does not hold for soft criteria. The most winding route need not be most appealing to a motorcyclist. This has two major implications. First, there is not strict notion of optimality for such criteria. Instead, trade-offs between hard and soft criteria are likely to be preferred (cf. Part II). Second, quality is evidently highly subjective. Different knowledge will infer different criteria and ideally in turn reflect different subjective interpretations of quality. Chapter 11 investigates means for widely automated extraction of such qualitative knowledge for various applications. Mining crowdsourced data sets, we extract qualitative information reflecting different notions of popularity according to the crowd. For instance, from the quantity and ratings of Flickr pictures at a certain location, one may infer a particular aesthetic appeal. Also, if two sights are often mentioned in conjunction in travel reports, this implies they go well together and might qualify for collective recommendation. These and other examples are studied in Chapter 11 theoretically and practically. Different types of data sources are reviewed, various implications are illuminated and several real-world data sets substantiate the claim that extensive and diverse knowledge can be mined from crowdsourced data. As with curated approaches, domain knowledge is required to achieve a certain result quality. However, the work shows that the abundance of (often freely available) crowdsourced data may be used for automated knowledge extraction purposes. Furthermore, the extracted knowledge is used to enrich the underlying road networks, yielding adjusted cost functions. The optimal paths within these enriched networks differ from conventional shortest paths. It is shown that these paths reflect qualitative knowledge without sacrificing their quantitative aspect. In Chapter 13, the 2TD-AOP is introduced, an extension of the family of NP-hard Orienteering Problems (OP). For a given budget and start and target locations, the solution to an OP is the path with maximal value while abiding by the budget. The OP and its variations often find application in tourist route recommendation. In these applications, the value often corresponds to scenicness or attractiveness. Hence, the findings of Chapter 11 may be utilized in these problems. We propose to extend the family of OPs by introducing twofold time-dependence. The

2TD-AOP allows for time-dependent travel time and value functions. The approach is the first time-dependent AOP to be evaluated on a large scale, real-world road network. The experiments establish that incorporating twofold time-dependence substantially improves results. Furthermore, it is shown that computing optimal results is infeasible. Thus, the proposed algorithm is unrivaled w.r.t. result quality and performance. In conclusion, the research in Part III points out methods to mine the rich source of crowdsourced data for qualitative knowledge. Data sources are reviewed, different means for knowledge extraction are surveyed and evaluated. The 2TD-AOP is a direct application for the attained knowledge scores. Albeit NP-hard and particularly complex, an efficient solution is proposed which yields valid results, even on large road networks as, for instance, used by commercial map providers. For future research, we want to put knowledge mined from different data sources to use in the 2TD-AOP. For this purpose, the dimension of time-dependence has to be incorporated into the mining techniques. Taking multiple value functions into account, computing the path skyline of 2TD-AOP results is another possibility for future research.

Sensor Data Applications and Query Processing Part IV of this thesis presents new queries and solutions pertaining to ubiquitous traffic problems based on sensor data. Recent advances in sensor technology and wireless connectivity have facilitated data collection and communication between entities significantly. Ideas to aggregate, mine and exploit this data to optimize traffic are manifold. They reach from smart road blocks to intervehicle warning systems to autonomous driving. Part IV investigates two applications with implications to everyday traffic. First, in Chapter 16, the application of personalized routing suggestions is examined. Given a set of historic trajectories and an incoming path query (by the same driver), the aim is to generate routing suggestions which correspond to the driver's preferences. There exist solutions to this task. However, most of these solutions derive location-dependent preferences, i.e., they infer favorite street segments. Thus, the most precise routing suggestions are made where most data has been collected. This is usually the home location of the driver where least direction is needed. In contrast, we propose a method which mines preferences according to the cost vector representation of a particular path. The cost vector of a path is independent of its location. Instead, it captures the particular trade-off between criteria the driver has chosen. We infer a preference distribution which may in turn be used to generate routing suggestions according to previous decisions. In Chapter 17, we address an abstract routing problem with several applications. The most prominent example is the search for a vacant parking space. Although it might be known where parking spaces exist, it is generally not known where vacant spaces are. However, recent advances in sensor technology allow for stationary as well as mobile system which may detect vacant parking spaces and feed such information into a cloud service. Based on this kind of information, the probability that a particular space is vacant may be deduced. Other examples of resources include charging stations for electric vehicles or taxi customers looking for a pick-up. We propose a probabilistic model which describes the volatility of such resources in a road network. Furthermore, we provide algorithmic solutions to this problem that handle its potentially infinite search

space. In conclusion, the research in Part IV shows that employing sensor data can indeed solve ubiquitous traffic problems while increasing convenience for the driver. Smart traffic shapes up to be inevitable in order to cope with environmental, urban and traffic issues. In the future, we want to investigate the possibilities of smart traffic solutions further. Rather than looking at solutions for individual drivers, we want examine possibilities for collaborative or collective optimization, in order to reduce overall drawbacks rather than support a single driver.

List of Figures

List of Tables

Bibliography

[1] Alvares, L. O., Bogorny, V., Kuijpers, B., de Macedo, J. A. F., Moelans, B., and Vaisman, A. A model for enriching trajectories with semantic geographical information. In *Proc. of the 15th Annual ACM Int'l Symp. on Advances in Geographic Information Systems* (2007), ACM, pp. 22:1–22:8.

[2] Aráoz, J., Fernández, E., and Zoltan, C. Privatized rural postman problems. *Computers & operations research 33*, 12 (2006), 3432–3449.

[3] Archetti, C., and Speranza, M. G. Arc routing problems with profits. *Arc Routing: Problems, Methods, and Applications, MOS-SIAM Series on Optimization* (2013), 257–284.

[4] Balteanu, A., Jossé, G., and Schubert, M. Mining driving preferences in multi-cost networks. In *Advances in Spatial and Temporal Databases*. Springer, 2013, pp. 74–91.

[5] Barber, C. B., Dobkin, D. P., and Huhdanpaa, H. The quickhull algorithm for convex hulls. *ACM Transactions on Mathematical Software (TOMS) 22*, 4 (1996), 469–483.

[6] Bellman, R. On a routing problem. *Quarterly of applied mathematics* (1958), 87–90.

[7] Bishop, C. M. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[8] Borzsony, S., Kossmann, D., and Stocker, K. The skyline operator. In *Proc. of the 17th Int'l Conf. on Data Engineering* (2001), pp. 421–430.

[9] Brilhante, I., Macedo, J. A., Nardini, F. M., Perego, R., and Renso, C. Where shall we go today?: Planning touristic tours with tripbuilder. In *Proceedings of the 22Nd ACM International Conference on Information & Knowledge Management* (New York, NY, USA, 2013), CIKM '13, ACM, pp. 757–762.

[10] Brumbaugh-Smith, J., and Shier, D. An empirical investigation of some bicriterion shortest path algorithms. *European Journal of Operational Research 43*, 2 (1989), 216–224.

[11] Chan, C. Y., Jagadish, H. V., Tan, K.-L., Tung, A. K. H., and Zhang, Z. On high dimensional skylines. In *Proceedings of the 10th International Conference on Extending Database Technology (EDBT), Munich, Germany* (2006), pp. 478–495.

[12] Chao, I.-M., Golden, B. L., and Wasil, E. A. A fast and effective heuristic for the orienteering problem. *European Journal of Operational Research 88*, 3 (1996), 475–489.

[13] Chen, C., Zhang, D., Guo, B., Ma, X., Pan, G., and Wu, Z. Tripplanner: Personalized trip planning leveraging heterogeneous crowdsourced digital footprints. *IEEE Transactions on Intelligent Transportation Systems 16*, 3 (2015), 1259–1273.

[14] Chen, H., Ku, W.-S., Sun, M.-T., and Zimmermann, R. The multi-rule partial sequenced route query. In *Proceedings of the 16th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems (ACMGIS'08)* (2008), pp. 10:1–10:10.

[15] Chen, H., Ku, W.-S., Sun, M.-T., and Zimmermann, R. The partial sequenced route query with traveling rules in road networks. *Geoinformatica 15*, 3 (2011), 541–569.

[16] Chen, L., Lv, M., Ye, Q., Chen, G., and Woodward, J. A personal route prediction system based on trajectory data mining. *Inf. Sci. 181*, 7 (Apr. 2011), 1264–1284.

[17] Chen, Z., Fu, R., Zhao, Z., Liu, Z., Xia, L., Chen, L., Cheng, P., Cao, C. C., Tong, Y., and Zhang, C. J. gmission: A general spatial crowdsourcing platform. *Proceedings of the VLDB Endowment 7*, 13 (2014), 1629–1632.

[18] Chen, Z., Shen, H. T., and Zhou, X. Discovering popular routes from trajectories. In *ICDE* (2011), pp. 900–911.

[19] Cheng, D., Hadjieleftheriou, M., Kollios, G., Li, F., and Teng, S.-H. On trip planning queries in spatial databases. In *Proceedings of the 9th International Conference on Advances in Spatial and Temporal Databases (SSTD'05)* (2005), pp. 273–290.

[20] Cherkassky, B. V., Goldberg, A. V., and Radzik, T. Shortest paths algorithms: Theory and experimental evaluation. *Mathematical programming 73*, 2 (1996), 129–174.

[21] Commission to the European Parliament, the Council, the European Committee and the Committee of the Regions. A roadmap for moving to a competitive low carbon economy in 2050, 2011.

[22] Dantzig, G. B. *Linear programming and extensions*. Princeton university press, 1998.

[23] De Choudhury, M., Feldman, M., Amer-Yahia, S., Golbandi, N., Lempel, R., and Yu, C. Automatic construction of travel itineraries using social breadcrumbs. In *Proceedings of the 21st ACM Conference on Hypertext and Hypermedia* (New York, NY, USA, 2010), HT '10, ACM, pp. 35–44.

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

[25] Delis, A., Efstathiou, V., and Verroios, V. Reaching available public parking spaces in urban environments using ad hoc networking. In *Proceedings of the 12th International Conference on Mobile Data Management (MDM'11)* (2011), pp. 141–151.

[26] Demiryurek, U., Banaei-Kashani, F., Shahabi, C., and Ranganathan, A. Online computation of fastest path in time-dependent spatial networks. In *International Symposium on Spatial and Temporal Databases* (2011), Springer, pp. 92–111.

[27] Demiryurek, U., Kashani, F. B., and Shahabi, C. A case for time-dependent shortest path computation in spatial networks. In *18th ACM SIGSPATIAL International Symposium on Advances in Geographic Information Systems, ACM-GIS 2010, November 3-5, 2010, San Jose, CA, USA, Proceedings* (2010), pp. 474–477.

[28] Dempster, A. P., Laird, N. M., and Rubin, D. B. Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society, Series B 39*, 1 (1977), 1–38.

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

[30] Dolev, N., Doytsher, Y., Kanza, Y., Safra, E., and Sagiv, Y. Computing a k-route over uncertain geographical data. In *Proceedings of the 10th International Conference on Advances in Spatial and Temporal Databases (SSTD'07)* (2007), pp. 276–293.

[31] Duckham, M., and Kulik, L. 169–185.

[32] Duckham, M., Winter, S., and Robinson, M. Including landmarks in routing instructions. *Journal on Location Based Services Vol. 4 4* (2010), 28–52.

[33] Efentakis, A., and Pfoser, D. Optimizing landmark-based routing and preprocessing. In *Proceedings of the 6th ACM SIGSPATIAL International Workshop on Computational Transportation Science* (2013), IWCTS '13, pp. 25:25–25:30.

[34] Ehrgott, M., and Gandibleux, X. A survey and annotated bibliography of multiobjective combinatorial optimization. *OR-Spektrum 22*, 4 (2000), 425–460.

[35] European Environmental Agency. Laying the foundations for greener transport, term 2011: transport indicators tracking progress towards environmental targets in europe, 2011.

[36] Feillet, D., Dejax, P., and Gendreau, M. Traveling salesman problems with profits. *Transportation science 39*, 2 (2005), 188–205.

[37] Feldman, D., Sugaya, A., Sung, C., and Rus, D. idiary: From gps signals to a text-searchable diary. In *Proc. of the 11th ACM Conf. on Embedded Networked Sensor Systems* (2013), ACM, pp. 6:1–6:12.

[38] Floyd, R. W. Algorithm 97: shortest path. *Communications of the ACM 5*, 6 (1962), 345.

[39] Fomin, F. V., and Lingas, A. Approximation algorithms for time-dependent orienteering. *Information Processing Letters 83*, 2 (2002), 57–62.

[40] Ford Jr, L. R. Network flow theory. Tech. rep., DTIC Document, 1956.

[41] Fredman, M., and Tarjan, R. Fibonacci heaps and their uses in improved network optimization algorithms. *2013 IEEE 54th Annual Symposium on Foundations of Computer Science 0* (1984), 338–346.

[42] Froehlich, J., and Krumm, J. Route prediction from trip observations. In *Society of Automotive Engineers (SAE) 2008 World Congress* (2008).

[43] Garcia, A., Arbelaitz, O., Linaza, M. T., Vansteenwegen, P., and Souffriau, W. *Personalized tourist route generation*. Springer, 2010.

[44] García, A., Arbelaitz, O., Linaza, M. T., Vansteenwegen, P., and Souffriau, W. Personalized tourist route generation. In *Current Trends in Web Engineering - 10th International Conference on Web Engineering, ICWE 2010 Workshops, Vienna, Austria, July 2010, Revised Selected Papers* (2010), pp. 486–497.

[45] Garcia, A., Linaza, M. T., Arbelaitz, O., and Vansteenwegen, P. Intelligent routing system for a personalised electronic tourist guide. *Information and Communication Technologies in Tourism 2009* (2009), 185–197.

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

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

[48] Gavalas, D., Konstantopoulos, C., Mastakas, K., Pantziou, G., and Vathis, N. Efficient heuristics for the time dependent team orienteering problem with time windows. In *Proceedings of the First International Conference on Applied Algorithms - Volume 8321* (New York, NY, USA, 2014), ICAA 2014, Springer-Verlag New York, Inc., pp. 152–163.

[49] Gavalas, D., Konstantopoulos, C., Mastakas, K., Pantziou, G., and Vathis, N. Approximation algorithms for the arc orienteering problem. *Information Processing Letters 115*, 2 (2015), 313–315.

[50] Geisberger, R., Sanders, P., Schultes, D., and Delling, D. Contraction hierarchies: Faster and simpler hierarchical routing in road networks. In *International Workshop on Experimental and Efficient Algorithms* (2008), Springer, pp. 319–333.

[51] Gendreau, M., Laporte, G., and Semet, F. A tabu search heuristic for the undirected selective travelling salesman problem. *European Journal of Operational Research 106*, 2 (1998), 539–545.

[52] Gidofalvi, G., Pedersen, T. B., Risch, T., and Zeitler, E. Highly scalable trip grouping for large-scale collective transportation systems. In *Proceedings of the 11th International Conference on Extending Database Technology: Advances in Database Technology* (New York, NY, USA, 2008), EDBT '08, ACM, pp. 678–689.

[53] Goldberg, A. V., and Harrelson, C. Computing the shortest path: $a^\star$ search meets graph theory. In *Proceedings of the 16th Annual ACM-SIAM Symposium on Discrete Algorithms* (2005), Society for Industrial and Applied Mathematics, pp. 156–165.

[54] Goldberg, A. V., Kaplan, H., and Werneck, R. F. Reach for A*: Efficient point-to-point shortest path algorithms. In *The Eighth Workshop on Algorithm Engineering and Experiments (ALENEX) SIAM, Miami, FL* (2006), pp. 129–143.

[55] Goldberg, A. V., and Werneck, R. F. Computing Point-to-Point Shortest Paths from External Memory. In *Proceedings of the 7th Workshop on Algorithm Engineering and Experiments* (2005), pp. 26–40.

[56] Golden, B. L., Levy, L., and Vohra, R. The orienteering problem. *Naval research logistics 34*, 3 (1987), 307–318.

[57] Golden, B. L., Wang, Q., and Liu, L. A multifaceted heuristic for the orienteering problem. *Naval Research Logistics (NRL) 35*, 3 (1988), 359–366.

[58] Graf, F., Kriegel, H.-P., Renz, M., and Schubert, M. Mario: Multi attribute routing in open street map. In *Proceedings of the 12th International Symposium on Spatial and Temporal Databases (SSTD), Minneapolis, MN* (2011).

[59] Gunawan, A., Lau, H. C., and Vansteenwegen, P. Orienteering problem: A survey of recent variants, solution approaches and applications. *European Journal of Operational Research* (2016).

[60] Gunawan, A., Yuan, Z., and Lau, H. C. A mathematical model and metaheuristics for time dependent orienteering problem. 202–217.

[61] Hansen, P. Bicriterion path problems. In *Multiple criteria decision making theory and application*. Springer, 1980, pp. 109–127.

[62] Hart, P. E., Nilsson, N. J., and Raphael, B. A formal basis for the heuristic determination of minimum cost paths. *Systems Science and Cybernetics, IEEE Transactions on 4*, 2 (1968), 100–107.

[63] Hsieh, H., and Li, C. Mining and planning time-aware routes from check-in data. In *Proceedings of the 23rd ACM International Conference on Conference on Information and Knowledge Management, CIKM 2014, Shanghai, China, November 3-7, 2014* (2014), pp. 481–490.

[64] Hsieh, H. P., Li, C. T., and Lin, S. D. Exploiting large-scale check-in data to recommend time-sensitive routes. In *Proceedings of the ACM SIGKDD International Workshop on Urban Computing* (2012), ACM.

[65] Jagadish, H., Gehrke, J., Labrinidis, A., Papakonstantinou, Y., Patel, J. M., Ramakrishnan, R., and Shahabi, C. Big data and its technical challenges. *Communications of the ACM 57*, 7 (2014), 86–94.

[66] Jossé, G., Franzke, M., Skoumas, G., Züfle, A., Nascimento, M. A., and Renz, M. A framework for computation of popular paths from crowdsourced data. In *31st IEEE International Conference on Data Engineering, ICDE 2015, Seoul, South Korea, April 13-17, 2015* (2015), pp. 1428–1431.

[67] Jossé, G., Schmid, K. A., and Schubert, M. Probabilistic resource route queries with reappearance. In *Proceedings of the 18th International Conference on Extending Database Technology (EDBT), Brussels, Belgium* (2015).

[68] Jossé, G., Schmid, K. A., Züfle, A., Skoumas, G., Schubert, M., and Pfoser, D. Tourismo: A user-preference tourist trip search engine. In *Advances in Spatial and Temporal Databases - 14th International Symposium, SSTD 2015, Hong Kong, China, August 26-28, 2015. Proceedings* (2015), pp. 514–519.

[69] Jossé, G., and Schubert, M. Towards resource route queries with reappearance. In *Proceedings of the 1st ACM SIGSPATIAL PhD Workshop, SIGSPATIAL PhD Workshop 2014, Dallas/Fort Worth, Texas, USA, November 4-7, 2014* (2014), pp. 4:1–4:5.

[70] Jossé, G., Schubert, M., and Kriegel, H.-P. Probabilistic parking queries using aging functions. In *Proceedings of the 21st ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems* (2013), SIGSPATIAL'13, ACM, pp. 452–455.

[71] Jossé, G., Schubert, M., and Zellner, L. Easyev: Monitoring and querying system for electric vehicle fleets using smart car data. In *Advances in Spatial and Temporal Databases - 14th International Symposium, SSTD 2015, Hong Kong, China, August 26-28, 2015. Proceedings* (2015), pp. 497–502.

[72] Kanza, Y., Levin, R., Safra, E., and Sagiv, Y. Interactive route search in the presence of order constraints. *The International Journal on Very Large Data Bases (VLDB'10)* (2010), 117–128.

[73] Kanza, Y., Safra, E., and Sagiv, Y. Route search over probabilistic geospatial data. In *Proceedings of the 11th International Symposium on Advances in Spatial and Temporal Databases (SSTD'09)* (2009), pp. 153–170.

[74] Kanza, Y., Safra, E., Sagiv, Y., and Doytsher, Y. Heuristic algorithms for route-search queries over geographical data. In *ACM SIGSPATIAL GIS* (2008), p. 11.

[75] Karbassi, A., and Barth, M. Vehicle route prediction and time of arrival estimation techniques for improved transportation system management. In *Intelligent Vehicles Symposium, 2003. Proceedings. IEEE* (2003), pp. 511–516.

[76] Kazemi, L., and Shahabi, C. Geocrowd: enabling query answering with spatial crowdsourcing. In *Proceedings of the 20th International Conference on Advances in Geographic Information Systems* (2012), ACM, pp. 189–198.

[77] Köhler, E., Möhring, R. H., and Schilling, H. Fast point-to-point shortest path computations with arc-flags. *9th Dimacs implementation challenge* (2006).

[78] Kolahdouzan, M., Shahabi, C., and Sharifzadeh, M. The optimal sequenced route query. *The International Journal on Very Large Data Bases (VLDB'08) 17*, 4 (2008), 765–787.

[79] Kossmann, D., Ramsak, F., and Rost, S. Shooting stars in the sky: an online algorithm for skyline queries. In *Proceedings of the 28th International Conference on Very Large Data Bases (VLDB), Hong Kong, China* (2002), pp. 275–286.

[80] Kriegel, H.-P., Renz, M., and Schubert, M. Route skyline queries: A multi-preference path planning approach. In *Data Engineering (ICDE), 2010 IEEE 26th International Conference on* (2010), IEEE, pp. 261–272.

[81] Laporte, G., and Martello, S. The selective travelling salesman problem. *Discrete applied mathematics 26*, 2-3 (1990), 193–207.

[82] Letchner, J., Krumm, J., and Horvitz, E. Trip router with individualized preferences (trip): Incorporating personalization into route planning. In *in proc. of 8th Conference on Innovative Applications of Artificial Intelligence(AAAI'06,* (2006), The AAAI Press.

[83] Levin, R., Kanza, Y., Safra, E., and Sagiv, Y. An interactive approach to route search. In *Proceedings of the 17th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems (ACMGIS'09)* (2009), pp. 408–411.

[84] Li, F., Cheng, D., Hadjieleftheriou, M., Kollios, G., and Teng, S.-H. On trip planning queries in spatial databases. In *Proceedings of the 9th International Conference on Advances in Spatial and Temporal Databases* (Berlin, Heidelberg, 2005), SSTD'05, Springer-Verlag, pp. 273–290.

[85] Li, J. Research on team orienteering problem with dynamic travel times. *Journal of Software 7*, 2 (2012), 249–255.

[86] Liang, Y.-C., Kulturel-Konak, S., and Smith, A. E. Meta heuristics for the orienteering problem. In *Evolutionary Computation, 2002. CEC'02. Proceedings of the 2002 Congress on* (2002), vol. 1, IEEE, pp. 384–389.

[87] Lin, X., Yuan, Y., Zhang, Q., and Zhang, Y. Selecting stars: The k most representative skyline operator. In *Proceedings of the 23rd International Conference on Data Engineering (ICDE), Istanbul, Turkey* (2007), pp. 86–95.

[88] Loper, E., and Bird, S. Nltk: The natural language toolkit. In *Proc. of the ACL-02 Workshop on Effective Tools and Methodologies for Teaching Natural Language Processing and Computational Linguistics - Volume 1* (2002), pp. 63–70.

[89] Lu, Y., and Shahabi, C. An arc orienteering algorithm to find the most scenic path on a large-scale road network. In *Proceedings of the 23rd SIGSPATIAL International Conference on Advances in Geographic Information Systems* (New York, NY, USA, 2015), GIS '15, ACM, pp. 46:1–46:10.

[90] Lu, Y., and Shahabi, C. An arc orienteering algorithm to find the most scenic path on a large-scale road network. In *Proceedings of the 23rd SIGSPATIAL International Conference on Advances in Geographic Information Systems* (2015), ACM, p. 46.

[91] Lv, M., Chen, L., and Chen, G. Discovering personally semantic places from gps trajectories. In *Proc. of the 21st ACM Int'l Conf. on Information and Knowledge Management* (2012), pp. 1552–1556.

[92] Ma, S., Wolfson, O., and Zheng, Y. T-share: A large-scale dynamic taxi ridesharing service. In *Proceeding of International Conference on Data Engineering (ICDE'13)* (2013), pp. 410 – 421.

[93] Machuca, E., and Mandow, L. Multiobjective route planning with precalculated heuristics. In *Proc. of the 15th Portuguese Conference on Artificial Intelligence (EPIA 2011)* (2011), pp. 98–107.

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

[95] Matos, L., Nune, J., and Trigo, A. Taxi pick-ups route optimization using genetic algorithms. In *Proceedings of the 10th International Conference on Adaptive and Natural Computing Algorithms (ICANNGA'11)* (2011), pp. 410–419.

[96] McKinsey Global Institute. Big data: The next frontier for innovation, competition, and productivity, 2011.

[97] McMullen, P. The maximum numbers of faces of a convex polytope. *Mathematika 17*, 02 (1970), 179–184.

[98] Möhring, R. H., Schilling, H., Schütz, B., Wagner, D., and Willhalm, T. Partitioning graphs to speed up dijkstras algorithm. In *International Workshop on Experimental and Efficient Algorithms* (2005), Springer, pp. 189–202.

[99] Mote, J., Murthy, I., and Olson, D. L. A parametric approach to solving bicriterion shortest path problems. *European Journal of Operational Research 53*, 1 (1991), 81–92.

[100] Mouratidis, K., Lin, Y., and Yiu, M. L. Preference queries in large multicost transportation networks. In *2010 IEEE 26th International Conference on Data Engineering (ICDE 2010)* (2010), IEEE, pp. 533–544.

[101] Mousselly-Sergieh, H., Watzinger, D., Huber, B., Döller, M., Egyed-Zsigmond, E., and Kosch, H. World-wide scale geotagged image dataset for automatic image annotation and reverse geotagging. In *Proc. of the 5th ACM Multimedia Systems Conf.* (2014), pp. 47–52.

[102] Mousselly-Sergieh, H., Watzinger, D., Huber, B., Döller, M., Egyed-Zsigmond, E., and Kosch, H. World-wide scale geotagged image dataset for automatic image annotation and reverse geotagging. In *ACM Multimedia Systems* (2014), pp. 47–52.

[103] Müller-Hannemann, M., and Weihe, K. Pareto shortest paths is often feasible in practice. In *Algorithm Engineering*. Springer, 2001, pp. 185–197.

[104] Newson, P., and Krumm, J. Hidden markov map matching through noise and sparseness. In *ACM SIGSPATIAL GIS* (2009), pp. 336–343.

[105] Newson, P., and Krumm, J. Hidden markov map matching through noise and sparseness. In *Proceedings of the 17th ACM SIGSPATIAL international conference on advances in geographic information systems* (2009), ACM, pp. 336–343.

[106] Nicholson, T. A. J. Finding the shortest route between two points in a network. *The computer journal 9*, 3 (1966), 275–280.

[107] Norris, J. *Markov Chains*. Cambridge Univ. Press, Cambridge, 1998.

[108] Özpeynirci, Ö., and Köksalan, M. An exact algorithm for finding extreme supported nondominated points of multiobjective mixed integer programs. *Management Science 56*, 12 (2010), 2302–2315.

[109] Palma, A. T., Bogorny, V., Kuijpers, B., and Alvares, L. O. A clustering-based approach for discovering interesting places in trajectories. In *Proc. of the ACM Symp. on Applied Computing* (2008), pp. 863–868.

[110] Papadias, D., Tao, Y., Fu, G., and Seeger, B. An optimal and progressive algorithm for skyline queries. In *Proceedings of the ACM International Conference on Management of Data (SIGMOD), San Diego, CA* (2003), pp. 467–478.

[111] Parent, C., Spaccapietra, S., Renso, C., Andrienko, G., Andrienko, N., Bogorny, V., Damiani, M. L., Gkoulalas-Divanis, A., Macedo, J., Pelekis, N., Theodoridis, Y., and Yan, Z. Semantic trajectories modeling and analysis. *ACM Comput. Surv. '13 45*, 4 (Aug. 2013), 42:1–42:32.

[112] Pei, J., Jin, W., Ester, M., and Tao, Y. Catching the best views of skyline: a semantic approach based on decisive subspaces. In *Proceedings of the 31st International Conference on Very Large Data Bases (VLDB), Trondheim, Norway* (2005), pp. 253–264.

[113] Preparatat, F. P., and Shamos, M. I. Computational geometry: An introduction.

[114] Quercia, D., Schifanella, R., and Aiello, L. M. The shortest path to happiness: Recommending beautiful, quiet, and happy routes in the city. *CoRR (abs/1407.1031)* (2014).

[115] Raith, A., and Ehrgott, M. A comparison of solution strategies for biobjective shortest path problems. *Computers & Operations Research 36*, 4 (2009), 1299–1331.

[116] Raith, A., and Ehrgott, M. A comparison of solution strategies for biobjective shortest path problems. *Computers & Operations Research 36*, 4 (2009), 1299–1331.

[117] Ramesh, R., and Brown, K. M. An efficient four-phase heuristic for the generalized orienteering problem. *Computers & Operations Research 18*, 2 (1991), 151–165.

[118] Richter, K.-F., and Duckham, M. Simplest instructions: Finding easy-to-describe routes for navigation. 274–289.

[119] Sacharidis, D., and Bouros, P. Routing directions: Keeping it fast and simple. In *Proc. of the 21st ACM Int'l Conf. on Advances in Geographic Information Systems* (2013), pp. 164–173.

[120] Sanders, P., and Schultes, D. Engineering highway hierarchies. In *European Symposium on Algorithms* (2006), Springer, pp. 804–816.

[121] Santos, D. O., and Xavier, E. C. Dynamic taxi and ridesharing: A framework and heuristics for the optimization problem. In *Proceedings of the 23rd International Joint Conference on Artificial Intelligence (IJCAI'13)* (2013), pp. 2885–2891.

[122] Schultes, D., and Sanders, P. Dynamic highway-node routing. In *International Workshop on Experimental and Efficient Algorithms* (2007), Springer, pp. 66–79.

[123] Schulz, F., Wagner, D., and Weihe, K. Dijkstras algorithm on-line: An empirical case study from public railroad transport. In *International Workshop on Algorithm Engineering* (1999), Springer, pp. 110–123.

[124] Seidel, R. The upper bound theorem for polytopes: an easy proof of its asymptotic version. *Computational Geometry 5*, 2 (1995), 115–116.

[125] Shekelyan, M., Jossé, G., and Schubert, M. Linear path skylines in multicriteria networks. In *31st IEEE International Conference on Data Engineering, ICDE 2015, Seoul, South Korea, April 13-17, 2015* (2015), pp. 459–470.

[126] Shekelyan, M., Jossé, G., and Schubert, M. Linear path skylines in multicriteria networks. In *31st IEEE International Conference on Data Engineering, ICDE 2015, Seoul, South Korea, April 13-17, 2015* (2015), pp. 459–470.

[127] Shekelyan, M., Jossé, G., and Schubert, M. Linear path skylines in multicriteria networks. In *ICDE15* (2015), pp. 459–470.

[128] Shekelyan, M., Jossé, G., and Schubert, M. Paretoprep: Efficient lower bounds for path skylines and fast path computation. In *Advances in Spatial and Temporal Databases - 14th International Symposium, SSTD 2015, Hong Kong, China, August 26-28, 2015. Proceedings* (2015), pp. 40–58.

[129] Shekelyan, M., Jossé, G., and Schubert, M. Paretoprep: Efficient lower bounds for path skylines and fast path computation. In *Advances in Spatial and Temporal Databases - 14th International Symposium, SSTD 2015, Hong Kong, China, August 26-28, 2015. Proceedings* (2015), pp. 40–58.

[130] Shekelyan, M., Jossé, G., Schubert, M., and Kriegel, H. Linear path skyline computation in bicriteria networks. In *Database Systems for Advanced Applications - 19th International Conference, DASFAA 2014, Bali, Indonesia, April 21-24, 2014. Proceedings, Part I* (2014), pp. 173–187.

[131] Shekelyan, M., Jossé, G., Schubert, M., and Kriegel, H. Linear path skyline computation in bicriteria networks. In *Database Systems for Advanced Applications - 19th International Conference, DASFAA 2014, Bali, Indonesia, April 21-24, 2014. Proceedings, Part I* (2014), pp. 173–187.

[132] Simmons, R., Browning, B., Zhang, Y., and Sadekar, V. Learning to predict driver route and destination intent. In *in proc. of IEEE Intelligent Transportation Systems Conference (ITSC '06)* (2006), pp. 127–132.

[133] Skoumas, G., Pfoser, D., and Kyrillidis, A. On quantifying qualitative geospatial data: A probabilistic approach. In *Proc. of the Second ACM Int'l Workshop on Crowdsourced and Volunteered Geographic Information* (2013), pp. 71–78.

[134] Skoumas, G., Pfoser, D., and Kyrillidis, A. T. Location estimation using crowdsourced geospatial narratives. *CoRR abs/1408.5894* (2014).

[135] Skoumas, G., Schmid, K. A., Jossé, G., Schubert, M., Nascimento, M. A., Züfle, A., Renz, M., and Pfoser, D. Knowledge-enriched route computation. In *Advances in Spatial and Temporal Databases - 14th International Symposium, SSTD 2015, Hong Kong, China, August 26-28, 2015. Proceedings* (2015), pp. 157–176.

[136] Skoumas, G., Schmid, K. A., Jossé, G., Züfle, A., Nascimento, M. A., Renz, M., and Pfoser, D. Towards knowledge-enriched path computation. In *Proceedings of the 22nd ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, Dallas/Fort Worth, TX, USA, November 4-7, 2014* (2014), pp. 485–488.

[137] Skriver, A. J. A classification of bicriterion shortest path (bsp) algorithms. *Asia Pacific Journal of Operational Research 17*, 2 (2000), 199–212.

[138] Skriver, A. J., and Andersen, K. A. A label correcting approach for solving bicriterion shortest-path problems. *Computers & Operations Research 27*, 6 (2000), 507–524.

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

[140] Spaccapietra, S., and Parent, C. Adding meaning to your steps. In *Proc. of the 30th Int'l Conf. on Conceptual Modeling* (2011), pp. 13–31.

[141] Stewart, B., and White III., C. Multiobjective $a^\star$. *Journal of the ACM (JACM) 38*, 4 (1991), 775–814.

[142] Tan, K. L., Eng, P.-K., and Ooi, B. C. Efficient progressive skyline computation. In *Proceedings of the 27th International Conference on Very Large Data Bases (VLDB), Roma, Italy* (2001), pp. 301–310.

[143] Tarapata, Z. Selected multicriteria shortest path problems: An analysis of complexity, models and adaptation of standard algorithms. *Int. J. Appl. Math. Comput. Sci. 17*, 2 (June 2007), 269–287.

[144] Tasgetiren, M. F. A genetic algorithm with an adaptive penalty function for the orienteering problem. *Journal of Economic and Social Research 4*, 2 (2001), 1–26.

[145] Texas A&M Transportation Institute. Urban mobility scoreboard 2015, 2015.

[146] To, H., Shahabi, C., and Kazemi, L. A server-assigned spatial crowdsourcing framework. *ACM Transactions on Spatial Algorithms and Systems 1*, 1 (2015), 2.

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

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

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

[150] Verbeeck, C., Sörensen, K., Aghezzaf, E., and Vansteenwegen, P. A fast solution method for the time-dependent orienteering problem. *European Journal of Operational Research 236*, 2 (2014), 419–432.

[151] Verbeeck, C., Vansteenwegen, P., and Aghezzaf, E.-H. An extension of the arc orienteering problem and its application to cycle trip planning. *Transportation Research Part E: Logistics and Transportation Review 68* (2014), 64 – 78.

[152] Wagner, D., Willhalm, T., and Zaroliagis, C. Geometric containers for efficient shortest-path computation. *Journal of Experimental Algorithmics (JEA) 10* (2005), 1–3.

[153] Warshall, S. A theorem on boolean matrices. *Journal of the ACM (JACM) 9*, 1 (1962), 11–12.

[154] Westphal, M., and Renz, J. Evaluating and minimizing ambiguities in qualitative route instructions. In *Proc. of the 19th ACM Int'l Conf. on Advances in Geographic Information Systems* (2011), pp. 171–180.

[155] Yan, Z., Chakraborty, D., Parent, C., Spaccapietra, S., and Aberer, K. Semitri: A framework for semantic annotation of heterogeneous trajectories. In *Proc. of the 14th Int'l Conf. on Extending Database Technology* (2011), pp. 259–270.

[156] Yan, Z., Chakraborty, D., Parent, C., Spaccapietra, S., and Aberer, K. Semantic trajectories: Mobility data computation and annotation. *ACM Trans. Intell. Syst. Technol. 4*, 3 (July 2013), 49:1–49:38.

[157] Yan, Z., Spremic, L., Chakraborty, D., Parent, C., Spaccapietra, S., and Aberer, K. Automatic construction and multi-level visualization of semantic trajectories. In *Proc. of the 18th Int'l Conf. on Advances in Geographic Information Systems* (2010), pp. 524–525.

[158] Yang, B., Guo, C., Jensen, C. S., Kaul, M., and Shang, S. Multi-cost optimal route planning under time-varying uncertainty. In *Proceedings of the 30th International Conference on Data Engineering (ICDE),Chicago,IL, USA* (2014).

[159] Yang, D., Zhang, D., Chen, L., and Qu, B. Nationtelescope: Monitoring and visualizing large-scale collective behavior in lbsns. *Journal of Network and Computer Applications 55* (2015), 170–180.

[160] Yang, D., Zhang, D., and Qu, B. Participatory cultural mapping based on collective behavior in location based social networks. *ACM Transactions on Intelligent Systems and Technology* (2015). in press.

[161] Yuan, J., Zheng, Y., Xie, X., and Sun, G. Driving with knowledge from the physical world. In *Proceedings of the 17th ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD), San Diego, CA* (2011), pp. 316–324.

[162] Yuan, J., Zheng, Y., Zhang, C., Xie, W., Xie, X., Sun, G., and Huang, Y. T-drive: driving directions based on taxi trajectories. In *Proceedings of the 18th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems (ACM GIS), San Jose, CA* (2010), pp. 99–108.

[163] Zheng, Y., and Zhou, X., Eds. *Computing with Spatial Trajectories*. Springer, 2011.

Acknowledgements

This work would not have been possible without the professional and personal support of those around me, at work and everywhere else.

Particular gratitude goes to my supervisor, PD Dr. Matthias Schubert, to whom I owe greatly. Not only did he provide funding for my position, he also taught me the ropes, he worked with me on papers, and he steered me and my research. Without his commitment, this would not have been possible.

I am especially thankful to Prof. Kyriakos Mouratidis who agreed to act as a second reviewer of this thesis.

For making this job so productive and pleasant at the same time, I have to thank the great team at the chair of Prof. Seidl. You have been splendid colleagues, and it has been an honor to work with and alongside of you. Particular thanks go to Georgios Skoumas, Johannes Niedermayer, Tobias Emrich, Andreas Züfle, Felix Borutta, Markus Mauder, Klaus Schmid, Matthias Renz, Mario A. Nascimento and, again, Matthias Schubert. Not to forget the backbone of the chair, Susanne Grienberger and Franz Krojer.

Finally, I thank my wife, Lisa. You mean the world to me. You are my home. And I thank my mother. Your love and support are invaluable. Thank you for always being there for me.