Rheinische Friedrich-Wilhelms-Universität Bonn
Institut für Geodäsie und Geoinformation

Analyzing Crowd-Sourced Trajectories to Infer Routing Preferences of Cyclists

Dissertation
zur Erlangung des Grades

Doktor der Ingenieurwissenschaften (Dr.-Ing.)

der Agrar-, Ernährungs- und Ingenieurwissenschaftlichen Fakultät
der Rheinischen Friedrich-Wilhelms-Universität Bonn

vorgelegt von
Axel Forsch

aus
Köln

Bonn 2025

Referent:	Prof. Dr. Jan-Henrik Haunert
Korreferentin:	Prof. Dr. Sabine Storandt
Tag der mündlichen Prüfung:	15. November 2024

Angefertigt mit Genehmigung der Agrar-, Ernährungs- und Ingenieurwissenschaftlichen Fakultät der Universität Bonn.

Kurzfassung

Im letzten Jahrzent haben die Aktivitäten vieler Freiwilliger zu fast vollständigen Darstellungen von Straßennetzen und großen Mengen von Trajektorien geführt. Letztere wurden vor allem von Freizeitsportlern gesammelt, die zum Beispiel Fahrradtouren oder Wanderungen aufgezeichnet haben. Diese neue Quelle von Geodaten, die im Bereich der freiwilligen geografischen Informationen (Volunteered Geographic Information, VGI) gesammelt werden, hat zum Aufkommen von Methoden zur Auswertung von Bewegungsdaten beigetragen.

Auf der einen Seite bietet die Nutzung von nutzergenerierten Daten aufgrund ihrer großen Menge und Verfügbarkeit sowie der intrinsischen Nutzersicht zahlreiche Vorteile. Auf der anderen Seite stellt die große Daten-Heterogenität eine Herausforderung dar. Diese Heterogenität tritt in verschiedenen Aspekten der Daten auf, was zu vier Herausforderung führt: (1) unbekannte räumliche Genauigkeit, (2) fehlende Metadaten, (3) ungleichmäßige Beteiligung und (4) ethische Aspekte der Datennutzung.

Diese Arbeit stellt eine methodische Pipeline vor, mit der sich die Routingpräferenzen der Nutzer aus ihren Bewegungsdaten ableiten lassen. Die Pipeline gliedert sich in Vorverarbeitung, Analyse und Visualisierung. Jeder Teil ist darauf ausgerichtet, die oben genannten Herausforderungen zu bewältigen. Die zugrundeliegenden Probleme sind als Optimierungsprobleme modelliert, und es werden effiziente Algorithmen zu ihrer Lösung entwickelt.

Im Vorverarbeitungsteil wird ein Algorithmus zum Kartenabgleich vorgestellt, der mögliche Bewegungen abseits des Straßennetzes explizit modelliert. Außerdem werden Techniken zum Schutz der Privatsphäre der Nutzer diskutiert.

Im Analyseteil wird ein Algorithmus vorgestellt, mit dem aus Bewegungsdaten Routingpräferenzen abgeleitet werden können. Der Algorithmus funktioniert auch mit wenigen Eingabedaten und kann selbst Bewegungungen, die nicht durch ein einziges (kombiniertes) Optimierungskriterium erklärt werden können, wie z.B. Rundfahrten, auswerten. Damit wird auf unterschiedliche Bewegungsintentionen und Ungleichheit der Beteiligung eingegangen.

Der Visualisierungsteil führt eine Methode zur Berechnung von kartographischen Symbolen ein, die die Fortsetzung der Bewegungsdaten außerhalb des Bildschirms visualisieren und so die Erkundung großer Bewegungsdatensätze erleichtern. Weiterhin werden Ansätze zur Visualisierung von Reisezeiten durch Isolinien vorgestellt. Diese Visualisierungen nutzen schematische Darstellungen, um einfache, schnell zu begreifende Abbildungen zu erzeugen.

Insgesamt trägt diese Arbeit zu effizienten Algorithmen bei, die Einblicke in das Routing-Verhalten von Radfahrern auf der Grundlage von Nutzerdaten bieten. Die Ergebnisse können zur Unterstützung der Stadt- und Verkehrsplanung eingesetzt werden.

Abstract

In the last decade, the activities of many volunteers have resulted in almost complete representations of street networks and large sets of trajectories. The latter have primarily been collected by recreational sportspeople who, for example, have recorded bicycle tours or hikes with Global Navigation Satellite System (GNSS) receivers. This new source of geospatial data collected within the realm of Volunteered Geographic Information (VGI) contributed to the rise of trajectory data mining methods that analyze movement based on historical movement data.

Using crowd-sourced data has multiple benefits due to its availability and volume, particularly in domains where administrative data is scarce. Moreover, VGI data is mainly collected by local users. This local perspective is advantageous when using the data to gain insights into user behavior, as the data can be assumed to reflect the users' views. However, working with VGI data presents challenges due to its high heterogeneity. This heterogeneity occurs in different aspects of the data, resulting in four main challenges: (1) unknown spatial accuracy, (2) lack of metadata, (3) participation bias, and (4) ethics in data usage.

This thesis contributes to the analysis of crowd-sourced trajectory data by presenting a methodic pipeline for inferring the users' routing preferences from their past movements. The pipeline is structured into preprocessing, analysis, and visualization. Each part is specifically designed to address the aforementioned challenges. The underlying problems are modeled as optimization problems, and efficient algorithms to solve them are developed.

The preprocessing part includes a map-matching algorithm that adapts to varying spatial data quality and explicitly models potential off-road movement. Additionally, a discussion on ethical considerations and privacy preservation techniques is performed.

The subsequent analysis part presents an algorithm to infer routing preferences from trajectory recordings. The algorithm accommodates diverse user intents and addresses participation inequality by working even with sparse input data and trajectories that cannot be explained by a single (combined) optimization criterion, such as round trips.

The visualization part introduces a method for computing glyphs to visualize the off-screen parts of trajectories to facilitate the exploration of large trajectory datasets. Additionally, novel approaches to communicate travel times and distances through schematic isolines are presented. These visualizations utilize schematic representations to convey potential inaccuracies, enhancing understanding while acknowledging data limitations.

Overall, this thesis contributes efficient algorithms that offer insights into the routing behavior of cyclists based on crowd-sourced data. The results can be applied to support decision-making in urban planning and transportation.

Contents

Chapter 1

Introduction

In the last decade, volunteers have impressively contributed to acquiring spatial data. Some geographic information scientists – among them Goodchild (2007) – have early understood that volunteered geographic information (VGI) will have a tremendous impact on their discipline since it is driven by citizens who act as a vast network of sensors. A standout example of VGI's impact is the OpenStreetMap[1] project, where volunteers contribute to a free map of the world. This map surpasses many regions' traditional sources in detail, completeness, and currency. Beyond OpenStreetMap, VGI refers to other georeferenced data volunteers share, including geo-tagged social media posts and images. Furthermore, technological advancement and the resulting increase in consumer-grade, location-enabled devices, such as smartphones and smartwatches, have empowered individuals to record and share their activities online effortlessly. This includes activities such as cycling and walking, generating large repositories of crowd-sourced trajectories.

These datasets contributed to the rise of trajectory data mining methods that analyze movement based on historical movement data (Andrienko and Andrienko, 2013; Laube, 2014; Mazimpaka and Timpf, 2016). Problems of interest include travel mode detection (Bohte and Maat, 2009; Gong et al., 2012; Zhang et al., 2013), trajectory clustering (Morris and Trivedi, 2009; Yuan et al., 2016), trajectory segementation (Aronov et al., 2016; Alewijnse et al., 2018), the identification of places of interest based on trajectories (Feuerhake et al., 2011), the detection of anomalous traffic patterns (Barragana et al., 2017; Huang, 2015), the manipulation of trajectories to preserve privacy (Brauer et al., 2022; Seidl et al., 2016), and predicting traffic flow from trajectories (Li et al., 2020; Brauer et al., 2021).

Using crowd-sourced data has been shown to provide benefits but also challenges. Most importantly, the availability and volume of the data have immense benefits, particularly in domains like movement data, where traditional administrative data is scarce. Figure 1.1 shows an example for this in form of the GeoPrivacy dataset (National Land Survey of Finland, 2023), featuring a dense coverage of the target region. Additionally, VGI data is predominantly collected by local users, thereby representing local knowledge, often characterized by a high (semantic) precision (Antoniou et al., 2017). This inherent local perspective is advantageous when using the data to gain insights into user behavior, as the data can be assumed to reflect

[1] https://www.openstreetmap.org/

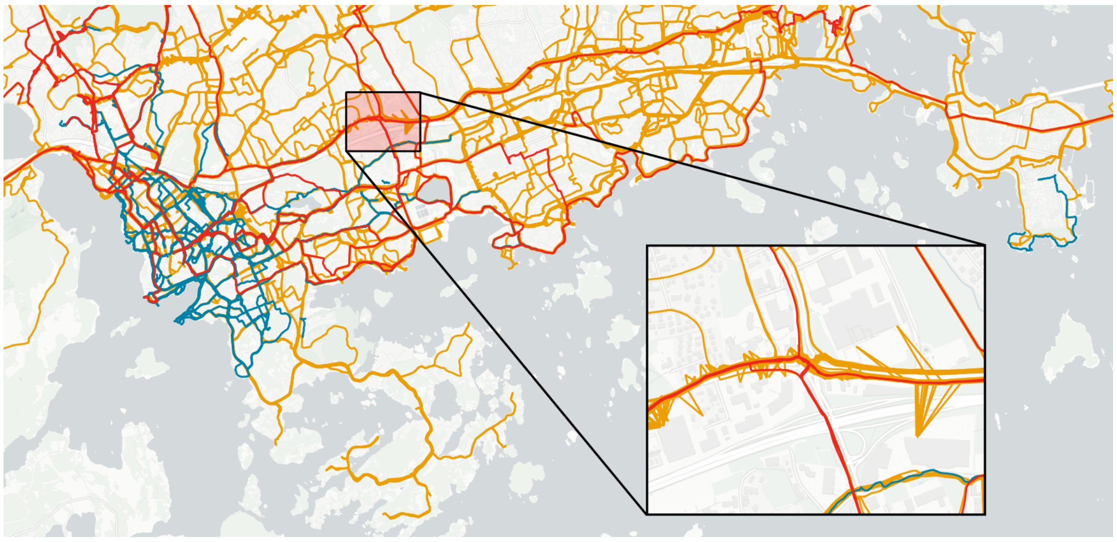


Figure 1.1: Excerpt of the GeoPrivacy dataset for the region of Espoo, Finland (National Land Survey of Finland, 2023). The dataset contains crowd-sourced trajectories of cycling (orange), running (red), and walking (blue). Visible are the benefits of VGI data, such as good coverage and a large amount of data, but also challenges, such as the data accuracy (see outliners in focus region). Background map: ©OpenStreetMap contributors ©CARTO

the users' views. However, VGI data presents challenges because of its high heterogeneity. This heterogeneity occurs in different aspects of the data, including the underlying user intent during contribution, spatial data quality, and the demographic characteristics of contributing users (Anahid Basiri and Mooney, 2019). An example of the varying spatial data quality can be seen in the focus region of Figure 1.1. Addressing these challenges requires robust data processing techniques capable of dealing with the varied nature of VGI data.

This thesis contributes to the analysis of crowd-sourced trajectory data by presenting an algorithmic pipeline for inferring the users' routing preferences from their past movements. The pipeline is structured into three parts: preprocessing, analysis, and visualization. Each part is specifically designed to address the high data heterogeneity of VGI data. We envision two ways the results of the analysis can improve the cycling experience. Firstly, they offer the potential to improve route recommendations for individual users, thereby enhancing their navigation experience. Secondly, these insights can provide valuable support to decision-makers in urban planning, facilitating informed decisions regarding the development of new infrastructure.

1.1 Crowd-sourced trajectory data

This chapter serves as an introduction to crowd-sourced trajectory data, as the characteristics of these data sets form a crucial motivation for developing the algorithms presented in this thesis. Volunteered geographic information (VGI) can be classified into actively generated data, i.e., through a user study, or passively collected data through location-enabled mobile devices (Burghardt et al., 2024). Actively generated VGI offers a controlled environment for data collection, potentially leading to higher data accuracy. However, significant effort is required to set up and execute such studies. In contrast, passively generated VGI is generated at a much larger scale and at lower cost but is more susceptible to inaccuracies and errors. To harness the potential of the large datasets generated passively, this thesis focuses on this category of VGI.

Challenges. One of the fundamental challenges when working with VGI trajectory data is its heterogeneous nature (Forsch et al., 2024). According to Senaratne et al. (2017), "data is produced by heterogeneous contributors, using various technologies and tools, having different levels of details and precision, serving heterogeneous purposes, and a lack of gatekeepers."

The first challenge concerns the **unknown spatial quality** of the data. A central quality measure for spatial data is the positional accuracy (Senaratne et al., 2017). While nowadays most VGI trajectory data is recorded by modern Global Navigation Satelite System (GNSS) receivers, which may offer accuracies of up to 5 to 10 meters (Kealy and Moore, 2017), outliers may still arise due to external factors such as topography or canopy cover, requiring appropriate handling (Ivanovic et al., 2016). A factor of spatial quality that is more challenging than positional accuracy is the completeness of the data. For instance, the absence of existing roads in the street network representation can lead to erroneous assumptions regarding users' routes.

One significant challenge stems from the **lack of metadata**, which is either available to a very limited extent or not at all. This particularly holds for the users' intent to record and share data, or their transportation mode. This lack of context can result in datasets containing a mixture of commuting and recreational trips or a mixture of different modes of transportation, each with distinct characteristics impacting subsequent analysis. An example of the diverse behavior of VGI contributors has recently been provided by Korpilo et al. (2017), who discovered that mountain bikers are more likely to move off-road than pedestrians.

While theoretically open to all, VGI contributions are often skewed toward certain demographic groups, introducing a social bias (Zhang and Zhu, 2018). Consequently, the data may not be representative of the broader population. Additionally, VGI contributions are higher in areas with high population density than in sparsely populated areas, resulting in a spatial bias (Haklay, 2010). These biases, also referred to as **participation inequality**, need to be accounted for when working with VGI (Nielsen, 2006; Haklay, 2016). For instance, a road that no trajectory has traversed may not necessarily be unfavorable to use, but maybe VGI participation was too low in the corresponding area.

Another critical challenge in VGI is the **ethics** in handling the data. Contributors often share their data for purposes unrelated to academic research, raising concerns about consent and privacy (Ghermandi and Sinclair, 2019). For example, a cyclist might share their trajectory for social reasons rather than to be analyzed towards their routing preferences. Moreover, trajectories are personal data that can give significant insight into a user's life (Primault et al., 2019). Dunkel et al. (2020) observe that the users' privacy can be compromised whether the data is contributed with explicit knowledge of how the data is used or not. Therefore, privacy-preserving mechanisms are essential when working with VGI trajectories.

In this thesis, we present algorithms for analyzing trajectories specifically designed with these challenges in mind.

Data sources. There are many different sources for crowd-sourced trajectory data. One of the largest repositories is the OpenStreetMap (OSM) traces[2], containing over 11 million trajectories totaling more than 28 billion location measurements worldwide (OpenStreetMap, 2024). These trajectories primarily support mapping efforts within OSM and typically lack additional information, such as transportation mode or temporal details, depending on the user's chosen visibility settings.

In the context of urban mobility, two prominent datasets are the *Geolife* (Zheng et al., 2011) and the *T-Drive* (Zheng, 2011) datasets. The Geolife dataset contains 17,621 trajectories of 182 users recorded between 2007 and 2012, featuring different transportation modes such as walking, cycling, driving, and public transportation. In contrast, the T-Drive dataset contains 10,357 trajectories exclusively recorded by taxis in Beijing, China, over three months in 2012.

In the domain of sports activities, community-driven social networks collect large trajectory datasets. For instance, on the Strava[3] platform alone, more than 120 million athletes recorded over 10 billion sports activities in 2023 (Strava Inc., 2024). These platforms allow querying the database for all trajectories inside a target area marked as public by the corresponding user. In the past, datasets generated by bulk downloading the query results have been used in several publications, e.g., Sports Tracker data[4] (Ferrari and Mamei, 2013; Oksanen et al., 2015; Brauer et al., 2021) and GPSies data[5] (Oehrlein et al., 2017; Filomena and Verstegen, 2021). However, recent legal frameworks like the General Data Protection Regulation (GDPR) and stricter terms for using these platforms have limited the creation of new datasets from these sources. While platforms such as Strava Metro (Strava Inc., 2023) provide access to their datasets, the access is limited to actors involved in transportation planning, and these datasets only contain aggregated data (Jokinen et al., 2021). The *GeoPrivacy* platform (Mäkinen et al., 2023) bridges the gap between volunteers and scientists by providing a GDPR-compliant trajectory dataset (National Land Survey of Finland, 2023) explicitly targeted for academic use, see Figure 1.1.

[2] www.openstreetmap.org/traces [3] www.strava.com [4] www.sports-tracker.com [5] www.gpsies.com

1.2 Objective and contribution

This thesis aims to utilize the large amount of trajectory data available in the context of VGI to infer the routing behavior of users, with a particular focus on cyclists and pedestrians. Developing methods that can deal with the high heterogeneity of the input data is a central requirement to achieve this objective.

Scientific contribution. The primary contribution of this thesis is a methodological pipeline for analyzing the routing behavior of individuals based on crowd-sourced trajectory data. The pipeline integrates methods for preprocessing, analyzing, and visualizing the data to address the considerable heterogeneity of crowd-sourced data. The methods are formulated as optimization problems for which exact algorithms are presented. Specifically, the methods are designed to address the challenges identified when working with crowd-sourced data: unknown spatial quality, lack of metadata, participation inequality, and ethics.

In addition to the algorithms targeted at the analysis of trajectories, this thesis also contributes an approach to create schematic isochrones, which are lines that connect points of equal travel time. The versatility of the approach is highlighted by demonstrating its applicability in visualizing (a) routing preferences, (b) travel times within public transportation networks, and (c) fare zones in schematic metro maps.

Technical contribution. Next to the theoretical contributions of this thesis, prototypical implementations for all presented algorithms are provided as open-source code. The implementations are written in the Java[6] programming language and can be accessed at:

• Map Matching	https://gitlab.igg.uni-bonn.de/graphlibrary
• Inferring Routing Preferences	https://gitlab.vgiscience.de/forsch/routing-preferences
• Off-Screen Evolution	https://github.com/GeoinfoBonn/offscreen-evolution
• Schematic Isochrones	https://github.com/GeoinfoBonn/travel-time-maps
• Metrochrones	https://gitlab.igg.uni-bonn.de/forsch/metrochrones

[6] www.java.com/

1.3 Outline

In the following thesis, algorithms for processing crowd-sourced trajectory data are presented. The algorithms can be divided into three main processing steps, each represented in one part of this thesis. Figure 1.2 provides a visual overview of the three-part structure of this thesis, highlighting the content of the individual chapters. Part I focuses on algorithms for trajectory preprocessing that prepare the raw input data for subsequent analysis. The outcome of this part is the representation of trajectories as *paths* within the underlying road network. In Part II, the analysis of these paths regarding the routing preferences of the recording user is investigated. For this, a compression-based algorithm is introduced to correlate information about the road network with the path. Finally, in Part III, the geographic visualization of trajectory datasets is discussed. Moreover, visualizations aimed at communicating the results of the routing preference inference are explored. Specifically, isochrone visualizations for public transportation networks are developed, and their applicability to visualizing routing preferences is shown.

The algorithms have been published in journals or were presented at conferences and published in the conferences' proceedings. In specific, the publications highlighted in this dissertation are the following:

Part I — Timon Behr, Thomas C. van Dijk, Axel Forsch, Jan-Henrik Haunert, and Sabine Storandt. **Map matching for semi-restricted trajectories.** In Krzysztof Janowicz and Judith A. Verstegen, editors, *11th International Conference on Geographic Information Science (GI-Science 2021) - Part II, volume 208 of Leibniz International Proceedings in Informatics (LIPIcs)*, pages 12:1–12:16, Dagstuhl, Germany, 2021. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. 10.4230/LIPIcs.GIScience.2021.II.12

Part II — Axel Forsch, Johannes Oehrlein, Benjamin Niedermann, and Jan-Henrik Haunert. **Inferring routing preferences from user-generated trajectories using a compression criterion.** *Journal of Spatial Information Science*, 26(5):99–124, 2023. 10.5311/JOSIS.2023.26.256

Part III — Axel Forsch, Friederike Amann, and Jan-Henrik Haunert. **Visualizing the off-screen evolution of trajectories.** *KN - Journal of Cartography and Geographic Information*, 72(3): 201–212, 2022. 10.1007/s42489-022-00106-6

Axel Forsch, Youness Dehbi, Benjamin Niedermann, Johannes Oehrlein, Peter Rottmann, and Jan-Henrik Haunert. **Multimodal travel-time maps with formally correct and schematic isochrones.** *Transactions in GIS*, 25(6):3233–3256, 2021. 10.1111/tgis.12821

Axel Forsch and Jan-Henrik Haunert. **Metrochrones: Schematic isochrones for schematic metro maps.** *The Cartographic Journal*, 2023. 10.1080/00087041.2023.2284436

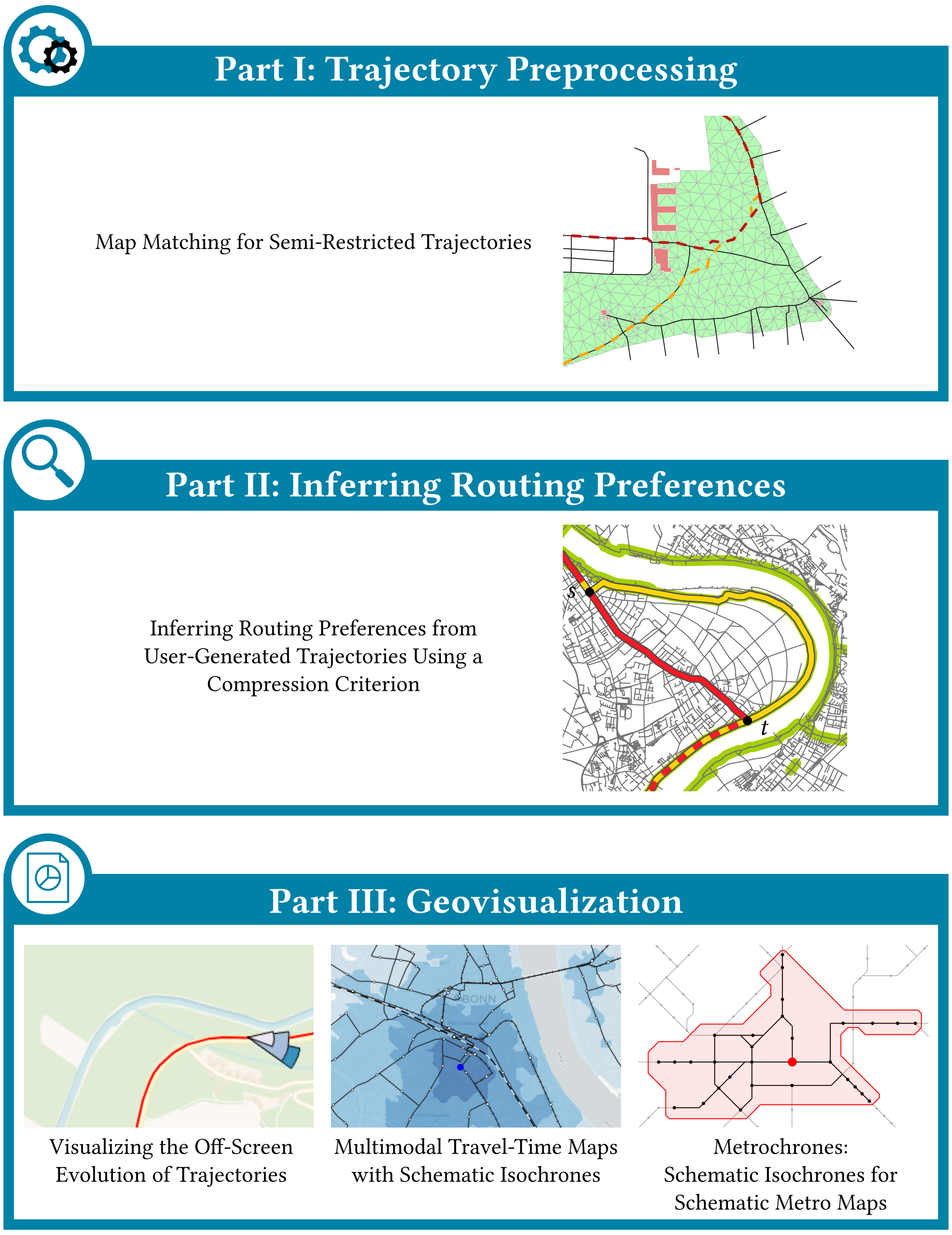

Part I: Trajectory Preprocessing
Map Matching for Semi-Restricted Trajectories
Part II: Inferring Routing Preferences
Inferring Routing Preferences from User-Generated Trajectories Using a Compression Criterion
s
t
Part III: Geovisualization
BONN
Visualizing the Off-Screen Evolution of Trajectories
Multimodal Travel-Time Maps with Schematic Isochrones
Metrochrones: Schematic Isochrones for Schematic Metro Maps

Figure 1.2: The three-part structure of this thesis. Part I deals with preprocessing methods addressing the data heterogeneity of the input. In Part II, inferring routing preferences from trajectories is discussed. Part III presents methods to visualize trajectories and analysis results.

Chapter 2

Methodological Background

In this chapter, the fundamental methods used throughout this thesis are introduced. In Section 2.1, computational complexity is introduced as a measure of an algorithm's efficiency. In Section 2.2, graphs as a data structure to store and analyze networks are discussed.

2.1 Computational complexity

Throughout this thesis, algorithms for analyzing geographical data are presented. The processing demands of these algorithms are contingent on the geographic extent of the input data, and as the scale of data increases, so does the computational workload. Consequently, to assess the efficiency of an algorithm, it becomes crucial to understand the computing resources it requires.

The *computatinal complexity* of an algorithm is a means for detailing its time and space requirements in relation to the input size. In this section, we introduce the notation for computational complexity used within the context of this thesis. For simplicity, we focus the description on the analysis of time complexity, with the understanding that the introduced notation can also be extended to address space requirements. Additional details can be found in standard algorithms textbooks, such as Cormen et al. (2009).

Asymptotic efficiency of algorithms. The time complexity of an algorithm gives information on the algorithm's *running time* in relation to the *input size*. The running time is estimated by counting the number of primitive operations being performed, assuming that each primitive operation takes a fixed amount of time. The measure for the input size varies depending on the specific problem; most commonly, the number of items in the input is used.

In many algorithms, even when the input size remains constant, some inputs may be processed faster than others. A classic example is that of sorting algorithms, where already sorted inputs lead to fast termination, while unsorted inputs demand more time. Consequently, it is valuable to distinguish between best-case and worst-case running time. In particular, the worst-case running time holds significance as it provides an upper bound on the algorithm's complexity, even for complex input.

While it is feasible to compute the exact running time for simple algorithms, in general, this task is often challenging. Therefore, we only consider the *asymptotic efficiency* of the algorithm. The asymptotic efficiency expresses how the running time scales with the input size as it approaches infinity. For sufficiently large input sizes, the impact of multiplicative constants and lower-order terms becomes negligible compared to the effects of the input itself. In this context, what matters most is the *rate of growth*, i.e., the highest-order term, which provides insights into the algorithm's scalability for increasingly large inputs. Therefore, it is common to express the running time with bounds instead of exact values.

Upper bound. The upper bound on the asymptotic behavior of a function expresses that the function grows no faster than a given rate. We express the upper bound using the O-notation, which is illustrated in Figure 2.1.

Definition 1 (O-notation)**.** Given two functions $f(n)$ and $g(n)$ defined on the real numbers, we say that f is big-oh of g as n approaches infinity (written as $f = O(g)$) if there exist constants c and n_0 such that $0 \leq f(n) \leq cg(n)$ for all $n \geq n_0$.

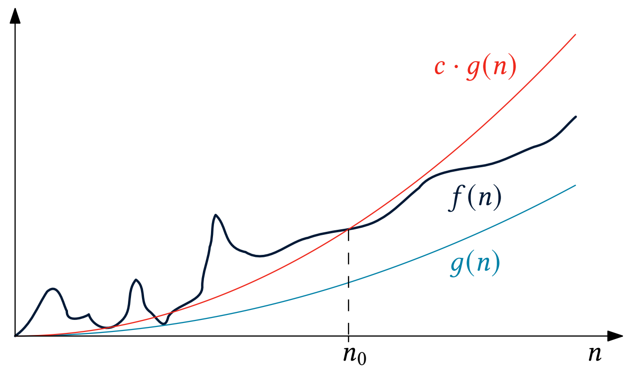


Figure 2.1: Illustration of the upper bound relationship denoted by the O-notation. The function f is $O(g)$ (here: $g = n^2$) as a constant multiple of g bounds its growth rate for $n > n_0$. The function f may represent the exact running time of an algorithm, which, in general, is not feasible to compute. The O-notation provides a tool for assessing and quantifying the algorithm's efficiency by characterizing its upper-bound behavior instead.

In the context of this thesis, we call an algorithm *efficient* if it qualifies as a *polynomial-time algorithm*, meaning its running time is bounded by $O(n^k)$ for some constant k. Noteworthy categories of polynomial-time algorithms include constant time ($O(1)$), logarithmic time ($O(\log n)$), and linear time ($O(n)$) algorithms. Note that the O-notation is versatile and applicable not only to running times but all mathematical functions. As such, the O-notation can also be used to express the space requirements of algorithms.

In the context of this thesis, most algorithms require a graph data structure as input, which consists of vertices and edges. Thus, to analyze the efficiency of these algorithms, it is suitable to express running times as functions of two input sizes: the number of vertices n and the number of edges m in the graph. An introduction to graphs follows in Section 2.2.

2.2 Graph theory

In this thesis, the analysis of networks plays an important role, with the most prominent example of networks encountered in this thesis being road networks. Typical analysis tasks in such networks include the computation of paths that are optimal with respect to a specific metric or connectivity analyses. Graphs are employed as a data structure to represent networks abstractly.

This section introduces the fundamental concepts of graph theory, providing a foundation for understanding both basic principles and advanced methods applied in the thesis. Comprehensive references can be found in textbooks, such as Cormen et al. (2009), for a more in-depth exploration of graph theory.

Basic terminology. A *graph* G is an ordered pair (V, E) where V represents the vertex set and E represents the edge set. The members of V are denoted as *vertices*, each representing a distinct element in the graph. At the same time, the members of E are termed *edges*, symbolizing the connections or relationships between the elements in the graph. Hence, each edge in E consists of two vertices in V. In the context of an *undirected graph*, an edge e between the vertices u and v is an unordered pair $\{u, v\}$, meaning that $\{u, v\} = \{v, u\}$. In a *directed graph*, an edge e' originating from u' and terminating in v' is represented as an ordered pair (u', v'). We call u' the *source* of e' and v' the *target* of e'. Figure 2.2 shows an example for a directed graph.

In the context of this thesis, directed graphs are more relevant than undirected ones. For ease of notation, we focus on concepts for directed graphs in the remainder of this section. Most of these concepts can be transferred to undirected graphs. We refer to the literature for a detailed description of undirected graphs.

Given an edge (u, v), we call the edge *incident* to vertex u and vertex v. Furthermore, vertex u is said to be *adjacent* to vertex v. Note that the adjacency relation is, in general, not symmetric. The *in-degree* of a vertex v is the number of incoming edges to v while the *out-degree* is the number of outgoing edges from v. The *degree* is the total number of incident edges to v, i.e., the sum of in-degree and out-degree. The maximum degree of a graph is the highest degree among all of its vertices. We speak of a graph G as a *bounded maximum degree graph* if its maximum degree is a constant $c \ll |V|$, meaning c is independent of the number of vertices in G.

Graphs can be categorized into dense and sparse graphs. In a *dense graph*, the number of edges is close to the maximum possible number of edges ($|E| = \frac{n \cdot (n-1)}{2}$, with $n = |V|$), while *sparse graphs* have relatively few edges compared to this maximum.

Paths. A path $P = \langle v_0, v_1, \ldots, v_k \rangle$ is a sequence of vertices such that $(v_{i-1}, v_i) \in E$ for $i = 1, 2, \ldots, k$, indicating pairwise connections between consecutive vertices through edges. Alongside the vertices, the path inherently *contains* the edges $e_1 = (v_0, v_1), e_2 = (v_1, v_2), \ldots, e_k = (v_{k-1}, v_k)$. Thus, an equivalent representation for P using its edges is $\langle e_1, e_2, \ldots, e_k \rangle$. Following the nomenclature of edges, we call $v_0 = s$ the source of P and $v_k = t$ its target. If the source and target vertices hold particular significance, we call P an s-t path and denote it with P_{st}. A path is called a *cycle*, if $s = t$. A *subpath* of a path is a contiguous subsequence of its vertices.

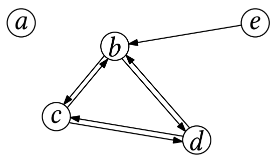


Figure 2.2: A directed graph with five vertices and seven directed edges.

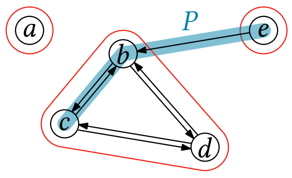


Figure 2.3: Vertex c is reachable from e through path P. The strongly connected components are marked in red.

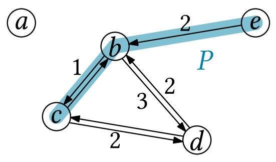


Figure 2.4: A weighted graph with edge weight function w. Path P has a weight of $w(P) = 3$ according to these weights.

A path's *length* is the number of edges on the path. In the general case, vertices and edges along a path may reoccur. However, a path is called *simple* if all its vertices (and consequently, edges) are unique. Figure 2.3 shows a simple e-c-path P of length two for the directed graph introduced in Figure 2.2.

Reachability and connectivity. A vertex v is called *reachable* from a vertex u via P if a u-v path P exists. A directed graph is called *strongly connected* if all of its vertices are mutually reachable from each other. Conversely, a directed graph is called *weakly connected* if all vertices are mutually reachable in the graph's undirected counterpart (formed by replacing all directed edges with undirected edges). The strongly connected components are marked in red in Figure 2.3. Most notably, vertices b, c, and d form a strongly connected component of size 3, while vertices a and e form a strongly connected component of size 1. In the following, unless stated otherwise, references to connectedness imply weak connectivity.

Edge weighting. A *weighted graph* is characterized by the inclusion of an *edge-weight function*. This function, denoted as $w: E \longrightarrow \mathbb{R}$, assigns numeric weight $w(e)$ to each edge e in the graph. By applying the definition of weights to paths, we can define the weight of a path P as the sum of all its edges' weights: $w(P) = \sum_{e \in P} w(e)$. An s-t path with the lowest weight, i.e., no other path between s and t with a lower weight exists, is called *minimum-weight path* with respect to w. Figure 2.4 shows a weighted variant of the graph introduced in Figure 2.2 with the unique minimum-weight path P between vertices e and c highlighted in blue.

The assigned weights offer additional insights into the relationships between adjacent vertices and can represent factors such as traversal costs, capacities, or distances. Depending on the context, the edge-weight function is sometimes referred to as *edge-cost function*, and the weights themselves as *costs*. In this thesis, the term "weight" is employed when maximizing a value is favored, whereas "cost" is used when minimization is the objective.

Planar graphs and embeddings. Graphs are generally location-agnostic, i.e., vertices have no fixed location. Nonetheless, it is required to draw a graph for many applications, where a vertex is represented at a location and each edge is represented as a simple arc. Such a drawing is called *embedding*.

A *planar graph* is a graph that can be drawn in the Euclidean plane without edge intersections. We call such a drawing *planar embedding* of the graph or simply *plane graph*. A planar embedding divides the plane into a set F of regions called *faces*. Analogously to edges and vertices, we call an edge e and a face F *incident* to each other if e borders F. According to Euler's formula, in a connected graph $G = (V, E)$ with planar embedding, $|V| - |E| + |F| = 2$ holds, i.e., $O(|V|) = O(|E|) = O(|F|)$. A *planar straight-line graph* is an embedding of a planar graph whose edges are drawn as straight lines.

In some contexts, the location of the vertices (and sometimes also the edges) is given. Graphs with given vertex locations are called *geometric graph*, and the graph and its embedding can be treated as equivalent.

Graph representations. There are two common ways to represent a graph $G = (V, E)$: the adjacency-matrix and the adjacency-list representation. In the *adjacency-matrix representation*, a $|V| \times |V|$ matrix A is used to store the adjacency information. Each matrix entry $a_{i,j} \in A$ with $i, j \leq |V|$ is equal to 1 if an edge from the i-th vertex to the j-th vertex exists, otherwise it is 0. This representation allocates memory for all possible $|V|^2$ edges in the graph and thus is inefficient for *sparse* graphs, where $|E|$ is much smaller than $|V|^2$.

In the *adjacency-list representation*, for each vertex u, an adjacency list $\mathcal{A}(u)$ is stored. For each outgoing edge (u, v) of u, the target vertex v of the edge is added to $\mathcal{A}(u)$. The memory requirement of this representation scales with the complexity of the graph and is therefore suitable for sparse graphs. This advantage in terms of memory usage comes at the disadvantage that looking up specific edges cannot be done in constant time anymore.

Most algorithms that work with graphs require the management of attributes for both vertices and edges. For instance, the edge weight or other information, such as the semantics of the abstracted elements, can be considered an attribute. Both graph representations can easily be extended to store these attributes by replacing the entries in the adjacency matrix (respectively, the adjacency list) with composite objects holding the attributes. The specifics of implementing these objects depend on the programming language being used.

In the following thesis, almost all considered graphs are sparse. Therefore, if not stated otherwise, we assume the adjacency-matrix representation for graphs.

Doubly connected edge list. Next to the two aforementioned generic graph representations, specialized representations are used to address specific tasks. One such representation is the *doubly-connected edge list* (DCEL), designed to represent plane graphs. This representation provides efficient access to the topological relationships among graph elements. Due to its ability to handle geometric information within graphs, it is often used in computational geometry (Berg et al., 2008, pp. 29ff).

Each (undirected) edge in the plane graph is considered as two *half edges*, one for each side of the edge. The two half edges for a given edge are called *twins*. Half edges are directed such that they are oriented anti-clockwise around their incident face. In the DCEL, one record is stored for each face, half edge, and vertex of the plane graph. Each record stores geometric information about the element it represents and topological information about neighboring graph elements. Additionally, supplemental semantic information can be stored in the record.

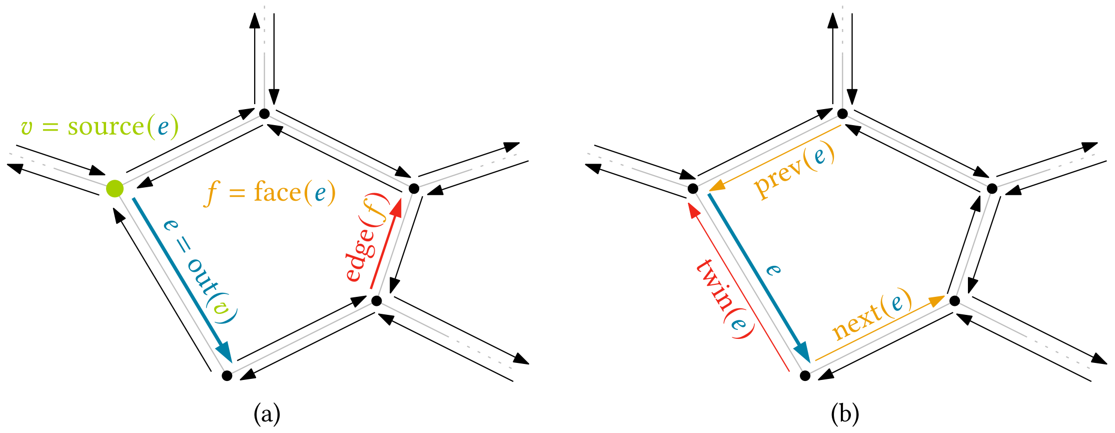


Figure 2.5: A doubly connected edge representation of the graph is shown in gray. (a) Pointers between the DCEL elements (vertices, half edges, and faces) store their topological relationship. (b) Each half edge stores pointers to neighboring half edges to allow navigating the graph.

The topological information is stored in the following way:

- Each vertex v stores a reference to an arbitrary outgoing half-edge out(v)
- Each face f stores a reference to an arbitrary incident half edge edge(f).
- Each half edge e stores references to its source vertex source(e) and its incident face face(e).

For effective graph navigation, each half-edge e additionally stores references to the following:

- next(e) and prev(e): The respective next and previous half edge to e in an anti-clockwise sequence around the incident face of e.
- twin(e): The twin half edge associated with e.

Figure 2.5 gives an overview of the topological information stored in a DCEL for an example plane graph. Common tasks efficiently addressed by a DCEL include (a) finding the outline of a face f, achieved by collecting the half edges found by iteratively following next($\cdot$) on the incident half edge edge(f) and (b) finding a neighboring face of a face f, accomplished by selecting the incident face of the twin half edge associated with f's incident half edge: face(twin(edge(f))). The inherent structure of the DCEL facilitates these operations, making it a powerful tool for topological analysis and navigation within plane graphs.

In the context of this thesis, the DCEL is frequently used in Part III to allow us to trace the outline of a so-called *reachable component* of a graph representing transit networks.

2.3 Optimal paths

One frequently encountered problem in this thesis is finding an optimal path between two locations in a network. A typical scenario is routing in a road network, where the objective is to find the most efficient route from point A to point B. The definition of *optimal* can vary depending on the context, such as minimizing geometric distance, expenditure of time, or fuel consumption. Therefore, when referring to an optimal path, it is crucial to specify the *opti-*

mization goal to which the path is optimal. In Chapter 6 we focus on finding the optimization goal from a given set of paths. Due to the importance of the optimization goal for this thesis, we speak of *optimal paths* instead of *shortest paths*, a term frequently used in literature (Cormen et al., 2009). This distinction helps prevent potential confusion, as “shortest paths” may imply a focus on geometric shortest paths, contrasting the broader optimization goals discussed in this thesis.

Problem definition. The scenario introduced is formalized in the *optimal-path problem*, where the goal is to find a path that is optimal with respect to a specified cost metric:

Definition 2 (Optimal path problem)**.** Given a (directed) graph $G = (V, E)$ with an edge cost function $c\colon E \longrightarrow \mathbb{R}$, a *source vertex* $s \in V$ and *sink vertex* $t \in V$, find an s-t path P with the lowest cost $c(P)$ in G. That is, there is no other s-t path P^* such that $c(P^*) < c(P)$.

As there is exactly one source and one target in the optimal path problem, we also refer to it as the *single-pair optimal path problem.* A generalized variant of this problem is the *single-source optimal path problem*, where optimal paths from a given source vertex to every other vertex in the graph are sought. Solving this problem also provides a solution to the single-pair optimal path problem. One naive approach to solving this problem could be enumerating all possible paths between the source vertex and the individual target vertices. However, even if we exclude paths that include cycles, the number of paths would be prohibitively large. In this section, we introduce Dijkstra’s algorithm (Dijkstra, 1959), which efficiently solves single-source optimal-path problems.

Dijkstra’s algorithm. Dijkstra’s algorithm solves the single-source optimal-path problem for a directed graph $G = (V, E)$ with a non-negative edge cost function c. That is, for every edge $e \in E$, $c(e) \geq 0$ holds.

The core concept of Dijkstra’s algorithm is, starting at the source vertex, to iteratively explore the neighborhood of parts that have already been visited while keeping track of the shortest paths already discovered, see Algorithm 1. Throughout the algorithm’s execution, a set $\mathcal{S}$ of vertices is maintained for which the minimum-cost path is already known. These vertices are termed *settled.* For all vertices adjacent to a settled vertex, a minimum-cost estimate, called *tentative cost*, is tracked, and these vertices are labeled as *visited.* Due to the distinct costs associated with the edges, the tentative cost for vertices may change during the algorithm (line 11). In each iteration, only the path to the visited vertex with the lowest tentative cost is guaranteed to be optimal. Thus, the corresponding vertex can be settled in that iteration (line 15). Figure 2.6 gives an example of an execution of Dijkstra’s algorithm on a directed, weighted graph. Each subfigure displays the state after one step of the exploration loop (line 7).

A *min-priority queue* is used to determine the vertex with the minimum tentative cost. This data structure allows storing and manipulating elements associated with an orderable key. More specifically, it provides two fundamental operations:

- ExtractMin(): This operation removes and returns the element associated with the lowest key.
- DecreaseKey(x, k): This operation reduces the key value for the element x to k.

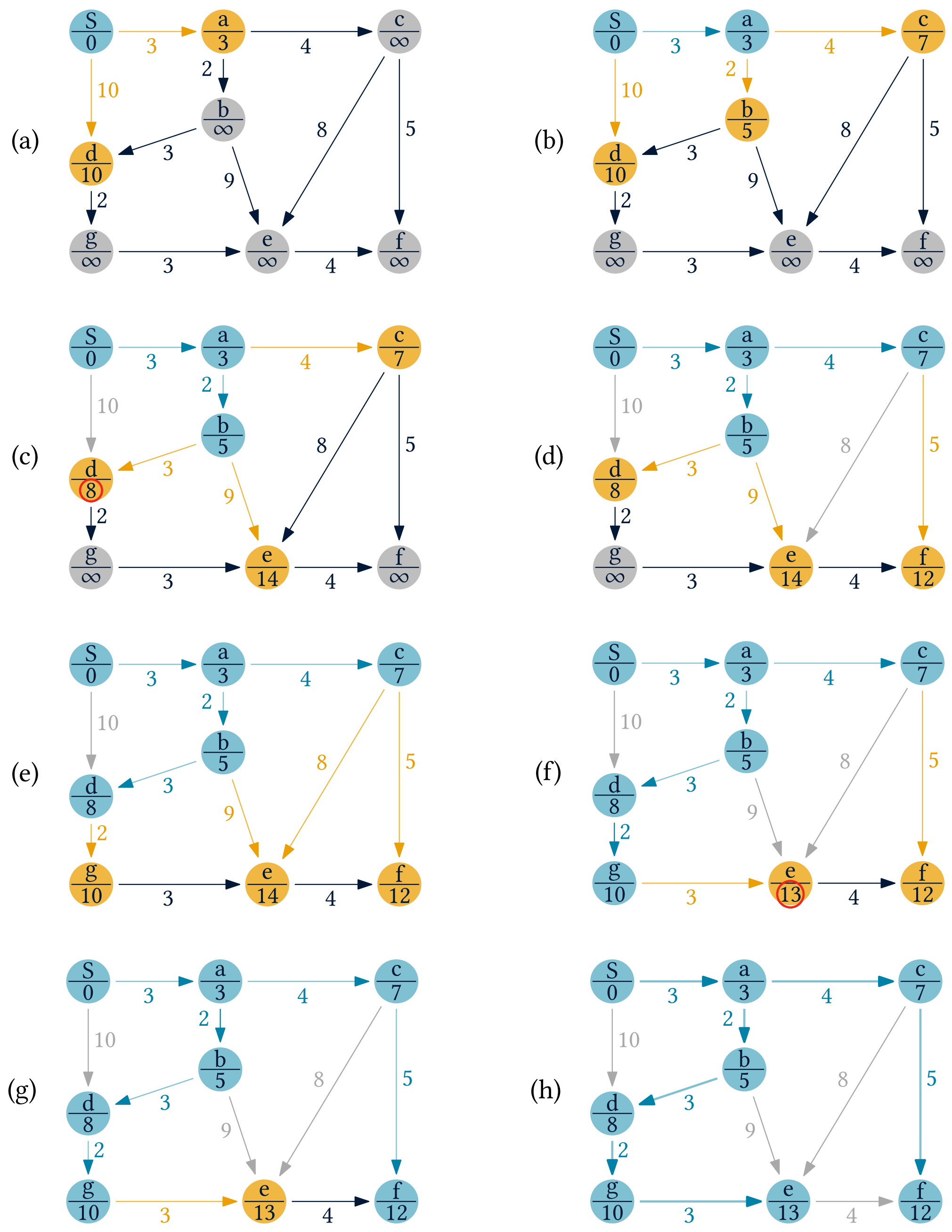


Figure 2.6: Dijkstra's algorithm starting at vertex S. Vertices in blue are settled, i.e. the shortest distance from S is already known. For vertices in orange, a path from S is found, but a shorter path might exist. Colored incoming edges represent the final (blue) or current best (orange) predecessor on the shortest path. The tentative distance to each vertex is displayed below it. (c) Settling vertex b reveals a shorter path to d, prompting an update in d's tentative distance and predecessor. (h) On completion, the predecessor edges form the shortest-path tree from S.

Furthermore, an initialization, denoted as MinPriorityQueue(S, κ), is assumed to set up a min-priority queue from a set S with an associated key function $\kappa\colon S \to \mathbb{R}$.

Up to this point, we have only considered the minimum cost for reaching a specific vertex. However, in practice, we are often interested in the actual path associated with this cost. Dijkstra's algorithm reconstructs this path by taking advantage of the *optimal substructure* of shortest paths, stating that an optimal path is composed of optimal subpaths (Cormen et al., 2009). Thus, if $P = \langle v_0, \dots, v_{k-1}, v_k \rangle$ is an optimal v_0-v_k path, the subpath $P^* = \langle v_0, \dots, v_{k-1} \rangle$ of P is an optimal v_0-v_{k-1} path. Consequently, the optimal paths can be fully reconstructed if, for each vertex u, its predecessor $\pi(u)$ is known, which we keep track of during algorithm execution, see Algorithm 1, line 13.

Algorithm 1: Dijkstra's algorithm

Data: graph $G = (V, E)$ with non-negative edge cost function c, source vertex $s \in V$
Result: minimum costs in G from s to all vertices reachable from s, information on the optimal paths

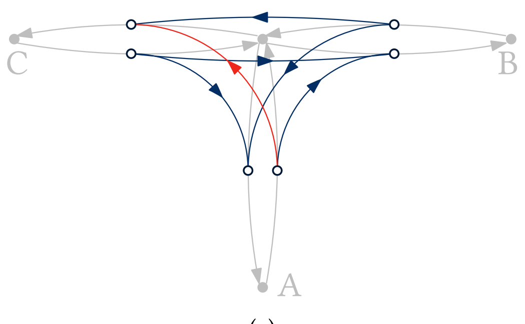
```
   // - Initialization - //
1  foreach vertex $u \in V$ do
2      $\gamma[u] \leftarrow \text{inf}$; // tenative costs
3      $\pi[u] \leftarrow$ NIL; // predecessors
4  $\gamma[s] \leftarrow 0$;
5  $Q \leftarrow$ MinPriorityQueue($V, \gamma$); // insert all vertices to the queue
6  $\mathcal{S} \leftarrow \varnothing$; // settled vertices
   // -- Exploration -- //
7  while $Q \neq \varnothing$ do
8      $u \leftarrow Q$.ExtractMin(); // extract vertex with minimum tentative cost
9      foreach outgoing edge $(u, v)$ of $u$ do
10         $\gamma^* \leftarrow \gamma[u] + c(u, v)$; // cost to $v$ over $u$
11         if $\gamma^* < \gamma[v]$ then // path to $v$ over $u$ has lower cost than previously found
12             $\gamma[v] \leftarrow \gamma^*$; // update tentative cost for $v$
13             $\pi[v] \leftarrow u$; // set $u$ as predecessor of $v$
14             $Q$.DecreaseKey($v, \gamma^*$);
15     $\mathcal{S} \leftarrow \mathcal{S} \cup \{u\}$; // mark $u$ settled
```

Examining the efficiency of Algorithm 1, we notice that each vertex gets extracted exactly once from the priority queue; see line 8. Furthermore, every edge in the graph is visited exactly once when its source vertex is extracted from Q; see line 9. The DecreaseKey function potentially gets executed once for each edge. Therefore, the running time of the algorithm is $O(T_e|V| + T_d|E|)$, where T_e is the time needed for ExtractMin and T_d is the time needed for DecreaseKey. Hence, The running time of Dijksta's algorithm depends on the implementation of the MinPriorityQueue. The *Fibonacci heap* (Fredman and Tarjan, 1987) is an implementation of a priority queue that allows ExtractMin to be performed in amortized logarithmic time and DecreaseKey to be performed in amortized constant time.[1] Using a Fibonacci heap, the running time of Dijkstra's algorithm thus is $O(|V| \log |V| + |E|)$.

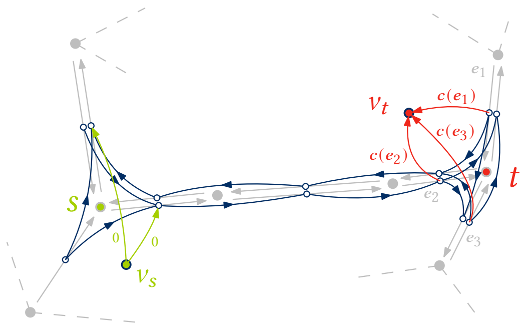
In general, $|E| = O(|V|^2)$ holds, i.e., the number of edges is quadratic in the number of vertices. Nonetheless, for planar graphs $|E| = O(|V|)$ holds, as discussed in Section 2.2. Many graphs to which we apply Dijksta's algorithm in this thesis are planar or at least close to planar. For these graphs, we can simplify the equation for the running time to $O(|V| \log |V|)$.

Pseudo-dual graph. In the preceding paragraph, we have introduced Dijkstra's algorithm for computing optimal paths in a graph. The edge-cost function c of G allows distinct costs to be applied to the edges of G. In some scenarios, however, costs are associated with the graph's vertices. Consider the graph representing the road network in a right-hand driving country. Due to traffic regulations, taking a right turn is often much easier than taking a left turn. Therefore, assigning a higher cost for turning left than for turning right might be appropriate. However, these costs are not associated with a road (i.e., edge) but with an intersection (i.e., vertex) of the network. Similarly, turn restrictions, i.e., forbidden edge-to-edge transitions, cannot be modelled in our current graph.

The *pseudo-dual graph* introduced by Winter (2002) can be used to address these issues. This model represents edge-to-edge transitions as explicit edges, enabling the reuse of an edge-cost function and Dijkstra's algorithm for the specific problem.



(a)



(b)

Figure 2.7: Modeling turn weights and turn restrictions with a pseudo-dual graph. (a) The pseudo-dual graph D (blue) of a primal graph (gray) representing a T-junction. If turning left (from A to C) is forbidden, the corresponding edge (red) can be removed from D. (b) To allow finding optimal paths between vertices of the primal graph, dummy source (green) and target (red) vertices are added to the outgoing (resp. incoming) edges of the primal source and target.

Given a directed weighted graph $G = (V, E)$ – which we call the *primal graph* in the following –, its complete pseudo-dual graph $D = (V_D, E_D)$ is a directed weighted graph with the following structure, displayed in Figure 2.7:

[1] The term *amortized* means that, over a large number of operations, the average performance per operation is guaranteed to be, at worst, the one stated. While a single operation can be more costly, the amortized analysis provides a more balanced view of the overall efficiency. Details can be found in textbooks (Cormen et al., 2009).

- For every primal edge $e \in E$, there is a vertex $v = d(e)$ in V_{D}. Here, $d \colon E \to V_{\mathrm{D}}$ is a bijective, one-to-one mapping function between primal edges and pseudo-dual vertices, satisfying $d^{-1}(v) = e$.
- For each pair of consecutive primal edges $\{e_i = (u, v), e_j = (v, w)\}$ in E there is an edge $\epsilon = (\nu, \omega)$ in E_{D} between the corresponding vertices $\nu = d(e_i)$ and $\omega = d(e_j)$.

Analogously to paths in the primal graph, a path Π in the pseudo-dual graph can be represented as a sequence of vertices $\Pi = \langle \nu_0, \nu_1, \ldots, \nu_k \rangle$, which directly translates to the edge-based representation of the path P in the primal graph by $P = \langle d^{-1}(\nu_0), d^{-1}(\nu_1), \ldots, d^{-1}(\nu_k) \rangle$.

In the optimal path problem, see Definition 2, we are given a primal source vertex s and a primal sink vertex t. These two vertices have no explicit representation in the pseudo-dual graph D. However, s can only be left through one of its outgoing edges, which have an explicit representation in D in the form of vertices. Similarly, t can only be entered through one of its incoming edges, explicitly represented in D as vertices. Solving the optimal path problem in the pseudo-dual graph thus results in a *multi-pair optimal path problem*, where the vertices $V_{\mathrm{D}}^{\mathrm{s}}$ corresponding to the primal source's outgoing edges are the new sources, and the vertices $V_{\mathrm{D}}^{\mathrm{t}}$ corresponding to the primal sink's incoming edges are the new sinks. We can convert this problem to the single-pair optimal path problem by introducing two *dummy vertices* to the pseudo-dual graph. The source dummy ν_{s} is connected by edges to the vertices $V_{\mathrm{D}}^{\mathrm{s}}$ while the sink dummy ν_{t} is connected by edges to the vertices $V_{\mathrm{D}}^{\mathrm{t}}$.

Until now, we have not considered the cost function $\kappa \colon E_{\mathrm{D}} \to \mathbb{R}$ of D. We want to be able to apply a cost ρ for edge-to-edge transitions while still considering the costs c of the primal graph. For this, the pseudo-dual cost function integrates both of these costs: $\kappa(\epsilon) = \rho(\epsilon) + c(d^{-1}(\text{source}(\epsilon)))$. The edges from the source dummy get zero cost while the edges to the sink get a cost $\kappa(\epsilon) = c(d^{-1}(\text{source}(\epsilon)))$. In specific, the dummy edges receive no transition cost ρ. Assuming $\rho(\epsilon) = 0$ for all edges in E_{D}, this definition guarantees that an s-t path P in G has the same cost $c(P)$ as its equivalent path in D. This setup is visualized in Figure 2.7b.

Next to the weighting of edge-to-edge transitions, we want to model turn restrictions. Turn restrictions are forbidden edge-to-edge transitions in the primal graph. Modeling these restrictions is achieved by removing the corresponding edges from the complete pseudo-dual graph, resulting in a *restricted pseudo-dual graph*.

The complexity of the pseudo-dual graph D in relation to the number of graph elements compared to its primal G is as follows. The number of vertices in D equals the number of primal edges, resulting in $|V_{\mathrm{D}}| = O(|E|)$. For each edge-to-edge transition in G, there is one corresponding edge in D. For each vertex v in G, there are $\textsc{InDegree}(v) \cdot \textsc{OutDegree}(v)$ possible edge-to-edge transitions. Assuming that the maximum degree $\delta(G)$ over all vertices in G is a constant with $\delta(G) \ll |E|$, this leads to $|E_{\mathrm{D}}| = O(|V|)$. This assumption particularly holds for planar graphs, the common type of primal graph considered in this thesis. Utilizing the fact that $O(|V|) = O(|E|)$ in planar graphs (see Section 2.2), we can conclude that $|V_{\mathrm{D}}| = O(|V|)$ and $|E_{\mathrm{D}}| = O(|E|)$. Therefore, pseudo-dual graphs for planar primal graphs have the same asymptotic complexity as their primal counterparts.

In this thesis, the concept of pseudo-dual graphs is extensively used in Part III, where the primal graphs are geometric graphs representing candidates for visualization. Using pseudo-dual graphs allows penalizing bends, promoting simple visualizations over more complex ones.

2.4 Trajectories

The main focus of this thesis is the analysis of mobility data. Specifically, we want to investigate the routes people take between two locations, A and B, and the underlying reason for choosing these routes over alternatives. In the context of geographic data, the recording of routes primarily relies on location-tracking devices, such as GNSS receivers. The data generated by these devices is referred to as trajectories. Formally, a *trajectory* T is a sequence of timed points represented by $T = \langle p_0 = (x_0, y_0, \tau_0), p_1 = (x_1, y_1, \tau_1), \ldots, p_n = (x_n, y_n, \tau_n)\rangle$, where $x_i, y_i, \tau_i \in \mathbb{R}$ and $\tau_i < \tau_{i+1}$ for each $i \in \{0, \ldots, n\}$. Here, x_i and y_i correspond to the coordinates of points in the Euclidean plane, while τ_i represents the timestamp associated with each location. Although location information can be extended to include height in some contexts, this thesis focuses on the simpler 2D variant, as height information is not relevant to the core objectives. We denote the part between two points $p_i, p_j \in T$ with $i \leq j$ as T_i^j and call it a *subtrajectory* of T.

The time between two location measurements is referred to as *sampling interval*, typically remaining constant over a given trajectory. For example, a typical sampling interval for GNSS receivers in modern smartphones is one second. Between the two trajectory points, no information about the route is available. It is common to assume straight-line motion between consecutive points, a reasonable assumption if the sampling interval is sufficiently small. The line connecting two consecutive trajectory points is referred to as *trajectory segment*. Figure 2.8 shows a trajectory consisting of 12 trajectory points with a constant sampling interval of two seconds. Red arrows indicate the trajectory segments.

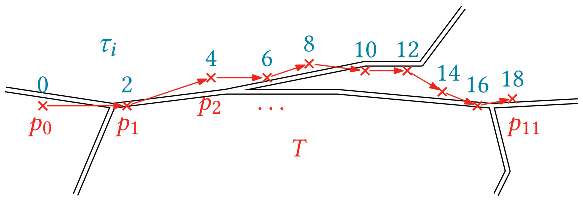


Figure 2.8: A trajectory with 12 points and the underlying road network. The timestamps (blue) are given as seconds after the initial location measurement. The trajectory has a constant sampling interval of two seconds.

Trajectory segmentation. Additional, derived information can be computed from the trajectory data that characterizes the movement. Examples are the velocity, the acceleration, the heading direction, or the sampling interval along the trajectory. The idea of *trajectory segmentation* is to split an input trajectory into subtrajectories to achieve relative uniformity in movement characteristics within each subtrajectory. This segmentation process facilitates a more nuanced comprehension of underlying movement patterns. For instance, analyzing distinctions between subtrajectories exhibiting high velocity and those with low velocity becomes feasible, providing insights into the dynamics of the observed movement. In this section, we introduce the discrete trajectory segmentation following the approaches from Aronov et al. (2016); Buchin et al. (2011) and Alewijnse et al. (2014), as it is closely related to our algorithm presented in Chapter 6. A comprehensive literature review on trajectory segmentation follows in Chapter 6.

Movement characteristics within a trajectory are defined by an *attribute function* that assigns attribute values to each point in time along the trajectory. The desired uniformity of a subtrajectory is defined through *criteria*, specifying the allowable differences in attribute values for them to be considered uniform. The *segmentation problem* entails determining the minimum number of subtrajectories such that all subtrajectories meet the specified criteria. The resulting segmentation is termed *optimal*. In the following, we approach segmentation as a *discrete* problem, meaning that a trajectory can only be split at trajectory points. For a more comprehensive understanding of the more general *continuous* variant, we direct readers to the literature, given its limited relevance within the scope of this thesis.

The segmentation criteria have distinct properties with respect to their behavior when the corresponding trajectory is extended or shortened. An important class of criteria are *monotone criteria*. If a monotone *decreasing* criterion holds for a specific trajectory T, it also holds for any subtrajectory of T. An illustrative example of a decreasing criterion is a constraint on the speed range, wherein the speed can vary by at most 5 km/h. Conversely, a monotone *increasing* criterion that holds for a subtrajectory T_i^j also holds for any trajectory containing T_i^j. An example of an increasing criterion is the trajectory length: if a subtrajectory exceeds a given length, any trajectory containing that subtrajectory also exceeds this length. In isolation, increasing criteria may not prove highly effective for segmentation, as they typically result in either segmenting the entire trajectory into a single subtrajectory or failing to identify a valid segmentation. Nevertheless, a Boolean combination of increasing and decreasing criteria can offer utility. For instance, in stay point detection, staying a minimum duration within a specified area qualifies as a stay point (Ye et al., 2009). Such combinations of increasing and decreasing criteria lose their monotonicity. Figure 2.9 shows a segmentation of the trajectory introduced in Figure 2.8 using the decreasing criterion of limited speed variation. The segmentation shown is a minimum segmentation, i.e., there is no way to segment the trajectory with fewer subtrajectories while still fulfilling the constraint.

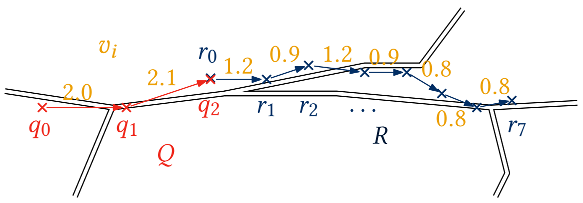


Figure 2.9: The trajectory depicted in Figure 2.8 with velocity per segment (orange). The trajectory is split into two subtrajectories Q (red) and R (blue), using the speed variance as the criterion. Within each subtrajectory, the speed varies by at most one unit. The displayed segmentation is a minimum segmentation for this criterion.

In the following, we present a framework for trajectory segmentation for non-monotone criteria as presented by Aronov et al. (2016). At the core of the segmentation framework lies a start-stop matrix $\mathcal{M}$, which encodes the relationship between a trajectory $T = \langle p_1, \ldots, p_n \rangle$ and a criterion C. The matrix, of dimension $n \times n$, comprises entries $\mathcal{M}[i, j]$ with $i, j \leq n$, where each entry is a Boolean value indicating whether the subtrajectory T_i^j of T satisfies criterion C. The entries where C is satisfied built up the *free space* while the other entries mark the *forbidden space* of $\mathcal{M}$. Subtrajectories are only defined for $i \leq j$, rendering $\mathcal{M}$ a right triangular matrix. The diagonal elements correspond to individual trajectory points. The segmentation problem is addressed through a two-step process utilizing the start-stop matrix. First, $\mathcal{M}$ must be populated with a value indicating the fulfillment of C. Subsequently, the optimal segmentation is determined within $\mathcal{M}$.

The entries of the start-stop matrix indicate whether a criterion is fulfilled or not. An auxiliary $n \times n$ matrix $\mathcal{A}$ holding the corresponding attribute values for the individual subtrajectories can be used to fill these entries. Transforming $\mathcal{A}$ to $\mathcal{M}$ then becomes a binary decision, i.e., $\mathcal{M}[i, j] = \textsc{True}$ if $\mathcal{A}[i, j]$ is in the allowed attribute range for criterion C, otherwise $\mathcal{M}[i, j] = \textsc{False}$. Using this approach, filling the start-stop matrix can be done in $O(T_\mathrm{e} \cdot n^2)$ time, where T_e is the time needed to evaluate the attribute value for a given subtrajectory of length m. For many common criteria, $T_\mathrm{e} = O(m)$ holds.

Any valid segmentation of T can be visualized as a staircase path through $\mathcal{M}$, connecting $\mathcal{M}[1, 1]$ to $\mathcal{M}[n, n]$. Each step in the path corresponds to one subtrajectory. Using dynamic programming, a minimal (discrete) segmentation of T can be found in $O(n^2)$ time (Aronov et al., 2016, Section 2.2), resulting in a combined running time of $O(T_\mathrm{e} \cdot n^2)$ for finding the optimal segmentation.

Map matching. A trajectory is a sequence of raw location measurements. In various applications, understanding the context of this trajectory is crucial. In the context of human mobility data, we are particularly interested in the streets a person traveled while recording the trajectory.

Like any measurement, trajectory data contains inaccuracies, and the abstraction of the street network into a graph model introduces an additional layer of complexity. Consequently, pinpointing the street corresponding to a trajectory segment is not straightforward. Finding the best trajectory representation within a real-world model of streets is what we call *map matching*.

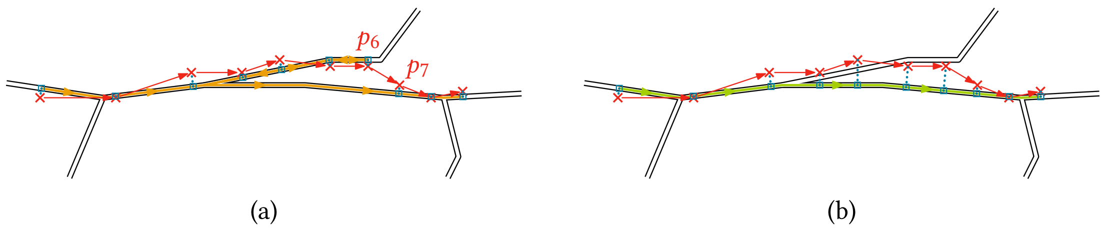


Figure 2.10: Map matching is demonstrated using the trajectory illustrated in Figure 2.8. Blue, dotted lines display the relation between the trajectory and matched points. (a) Naive approach by mapping each trajectory point to the closest road segment. The resulting path (shown in orange) in the network performs a detour deviating from the original movement. (b) An improved matching (shown in green) that closely resembles the original movement. Achieving such a solution requires advanced matching algorithms, such as the one presented in Chapter 4.

In map matching, a given trajectory is transformed into a path in a graph. This path, the *matched trajectory*, combines information on the street network with information on the mobility data and serves as the basis for many further analysis algorithms. Figure 2.10 gives two examples of different solutions of the matched trajectory for the (raw) trajectory introduced in Figure 2.8. The solution presented in Figure 2.10a stems from a simplistic map-matching approach where each trajectory point is mapped to the geometrically nearest point in the road network. However, the resulting matched trajectory takes an implausible route, making a left turn followed by a U-turn to return to the central road. This movement not only seems unnatural but would also imply an unrealistically high velocity between the matched points p_6 and p_7 (recall that the time between two trajectory points is two seconds).

Contrastingly, Figure 2.10b illustrates a matched trajectory that follows the central road. Although the distance between trajectory points and matched points is larger compared to the naive solution, this advanced approach better captures the overall shape of the input trajectory. Such refined solutions are achieved through the implementation of more sophisticated map-matching algorithms.

For a comprehensive understanding of map matching, including related work, readers are directed to Chapter 4. Within the same chapter, map matching specifically designed for trajectories that are not restricted to the modeled street network is discussed. For these trajectories, no suitable path can be found inside the network, hindering the applicability of subsequent analyses. The presented algorithm resolves this issue by extending the road network in areas where off-road movement is likely while ensuring that the extended parts of the network only serve as a fallback solution in case no suitable solution in the original network can be found.

Part I

Trajectory Preprocessing

Chapter 3

Trajectory Preprocessing: Motivation & Research Context

Trajectory data obtained from crowd-sourced databases is often not polished, e.g., it contains outliers or lacks sufficient metadata. It is often necessary to preprocess the data to apply existing tools to analyze and interpret such trajectory data. This chapter discusses preprocessing methods that address these challenges, aiming to enhance the utility and reliability of trajectory data.

Usually, trajectories are generated using location-enabled devices that can measure their location within a given reference system at a specific time. We call the trajectory data obtained directly from the device the *raw trajectory data*. Depending on the context, specific properties of the data are of interest. For instance, when assessing the accuracy of the recording device, the measurement noise is of interest. Conversely, in many other applications, the goal is to eliminate the noise from the data. In the following, four challenges related to raw trajectory data in the context of human mobility analysis are discussed:

1. **Noise and outliers:** Raw trajectories are prone to noise and outliers, stemming from inaccuracies in the recording devices or environmental factors. These inaccuracies can hinder further analysis or distort the analysis results.
2. **Periods of stagnation:** Raw trajectories often include intervals where the user remains stationary. These stay points can have a high impact on the analysis – depending on the application, it is desirable to remove them or focus only on these stay points.
3. **Privacy concerns:** Raw trajectories may contain sensitive information about the user, raising privacy concerns. Anonymizing or obfuscating the data while preserving its analytical value is important for ethical processing.
4. **Lack of context:** Raw trajectories lack contextual information about the surrounding environment, such as the road network or landmarks. Integrating spatial context opens new possibilities for trajectory analysis.

This chapter reviews literature dealing with the Challenges 1-3. Subsequently, Chapter 4 addresses Challenge 4 by introducing a novel algorithm for finding a trajectory representation in the underlying road network. The algorithm is specifically designed to be robust against off-road segments in the trajectory as well as missing roads in the road network data.

3.1 Trajectory smoothing

In the context of crowd-sourced (sports) trajectories, positioning using Global Navigation Satellite Systems (GNSS) is most relevant. Today, even low-cost GNSS receivers offer spatial accuracies ranging from 5 to 10 meters (Kealy and Moore, 2017), which may suffice for most applications. However, challenges arise when computing derived metrics such as velocity or acceleration, as taking these derivatives amplifies the inherent measurement noise. Additionally, external influences can cause erroneous measurements and outliers in the data. Trajectory filtering and outlier detection algorithms have been proposed to address these problems.

3.1.1 Trajectory filtering

The goal of trajectory filtering is to reduce the measurement noise. One basic approach is employing mean or median filters, which update each point measurement to the average (respective median) value of itself and a specified number $k-1$ of neighboring points. These filters are referred to as *sliding-window filters* due to their window-based processing mechanism, where the window size n determines the extent of data considered for filtering. Despite their simplicity and effectiveness in noise reduction, sliding window filters suffer inherent drawbacks. Notably, they introduce a lag in the filtered data, meaning that sudden changes in direction will only gradually be recovered. Larger window sizes – while generating a smoother output – aggravate this effect. Furthermore, these filters do not produce good estimates for velocity and acceleration (Lee and Krumm, 2011). To reduce the lag, Schüssler and Axhausen (2008) present a sliding-window filter based on a Gaussian Kernel. This approach increases the influence of temporally close measurements on the filter output compared to those further in time.

An advanced filtering method is the *Kalman filter*, which models the measurement and process model as well as their corresponding noise (Lee and Krumm, 2011). The Kalman filter has been shown to provide good estimates for speed and acceleration (Jun et al., 2006). However, its applicability in crowd-sourced data is limited due to the lack of information about measurement and process noise in this domain (Schüssler and Axhausen, 2008).

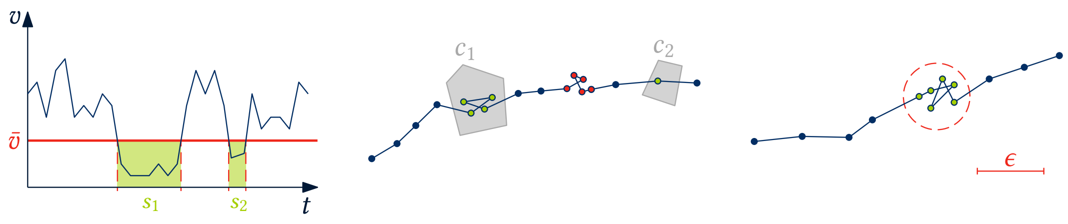
For the specific use case of map creation, Cao and Krumm (2009) propose an energy-based approach to smooth large sets of trajectories by pulling multiple nearby trajectories onto each other. This method offers a different perspective on trajectory filtering, focusing on the collective behavior of trajectories rather than individual point measurements.

3.1.2 Outlier detection

Many of the filtering methods mentioned above are susceptible to outlier points, that is, erroneous measurements beyond the expected measurement noise (Lee and Krumm, 2011). Thus, filtering effectiveness can be improved if these outliers are removed in advance. For instance, Ivanovic et al. (2016) propose an outlier detection based on changes in metrics and geometrics characteristics, such as speed and direction, of the measured points. More recently, Ivanovic et al. (2019) focus on outliers that stem from "secondary human behavior", which are sections where the contributor performed actions other than the main movement. Custers et al. (2019) present an outlier detection method motivated by physical properties of the tracked entity.

3.2 Stay-point detection

According to Spaccapietra et al. (2008), a trajectory can be conceptualized into a series of alternating stops and moves. The stops hold significant semantic information regarding the trajectory, indicating points of significance for the user. Conversely, individual trajectory points within a stop contribute relatively little information. Consequently, numerous approaches have been developed to detect these stops accurately. Stay-point detection is a crucial step when analyzing routing preferences to identify the origin and destination points of the route. It will, therefore, be reviewed in the following section. Figure 3.1 gives an overview of the approaches, categorized into velocity-based, point-of-interest-based, and density-based methods.



(a) velocity based (b) point-of-interest based (c) density based

Figure 3.1: Sketches of different stay-point-detection approaches. (a) In velocity-based approaches, trajectory points with velocities lower than a specified threshold are identified as stops. Slow movement, such as during s_2, may lead to false positives. (b) In point-of-interest-based approaches, stop candidates are modeled. Some stops, like the ones marked red, may not be identified if no candidate is present. (c) In density-based approaches, stops are detected where the trajectory remains within an area of extent ϵ for a predefined minimum duration.

3.2.1 Velocity based

Yan et al. (2010) introduce a computing platform for trajectory data processing, where stops are identified by applying a velocity threshold. Points with velocities lower than this threshold are classified as stops. Similarly, Stylianou (2017) use extrema in the velocity and acceleration to identify stay points. These velocity-based methods are sensitive to measurement noise and often require a constraint on a minimal stop duration to avoid false positives (Yan et al., 2010).

3.2.2 Point-of-interest based

Stops can be classified into *personalized stops*, relevant to individual users (such as their home location), and *shared stops*, relevant for many users (Tran et al., 2011) (such as restaurants). Existing points of interest (POIs) have been incorporated into stay-point-detection algorithms to identify shared stops. For instance, Alvares et al. (2007) model candidate stops as polygons around the given POIs and specify a minimum stay duration for each candidate. A stop is introduced for every trajectory that stays within a candidate stop for the minimum duration. A similar approach is proposed by Xie et al. (2009), who use a Voronoi-tesselation based on the POIs to generate the candidates.

3.2.3 Density based

Many popular stay-point detection methods rely on spatio-temporal point density. Palma et al. (2008) (CB-SMoT) and Tran et al. (2011) (TrajDBSCAN) introduce stay-point-detection algorithms based on modifications to the *DBSCAN* clustering algorithm (Ester et al., 1996). Both approaches define a stop as a subtrajectory within a spatial neighborhood of a given extent ϵ, with a minimum duration τ_{min}. The difference between the approaches lies in the definition of the neighborhood: CB-SMoT considers the trajectory, while TrajDBSCAN uses straight-line distance. Similarly, Gong et al. (2015) introduce a DBSCAN variant, which they denote C-DBSCAN, to extract stop locations. In contrast to modifying the neighborhood definition, the authors ensure temporal consistency of stops through constraint-based methods. DBSCAN-based algorithms are sensitive to the parameter choice for ϵ and τ_{min}, and inappropriate parameter sets can lead to a low performance of the algorithms. To address this issue, Kami et al. (2010) introduce a probabilistic point density estimate. However, their approach overlooks the temporal aspect of stops, which means that trajectories passing through the same location multiple times may result in false positive stop detections.

Stop detection can substantially improve the results of subsequent analysis depending on the research question. For instance, in the context of learning routing preferences, identifying the subtrajectories between stay points allows the separation of intent, such as reaching a specific location, from the routing behavior observed between the start and endpoint of the trajectory. However, revealing the stay points may also compromise the users' privacy. In the following section, methods to protect the users' privacy are discussed.

3.3 Privacy preservation

Trajectories are personal data and the users' privacy can be compromised when the data or subsequent analysis reveals sensitive information about the user. In response, Location-Privacy Preserving Methods (LPPMs) have been developed to mitigate this threat (Primault et al., 2019; Fiore et al., 2020), reviewed in the following. Then, this thesis gives particular attention to a truncation-based LPPM introduced by Brauer et al. (2022), which the author contributed to. This method holds particular importance within the thesis, as it maintains large parts of the trajectories unchanged, allowing for a representative analysis of the routing behavior while protecting user privacy. Additionally, this method is employed in processing the trajectory data of the crowd-sourced Geoprivacy platform (Mäkinen et al., 2023).

3.3.1 LPPMs providing formal privacy guarantees

Crowd-sourced trajectory datasets are often published in anonymized form, meaning they lack explicit user identifiers, such as their name. However, the threat of *re-identification attacks* remains. These attacks are designed to link anonymously published data to the individual they belong to (Henriksen-Bulmer and Jeary, 2016). A widely applied strategy to protect datasets from re-identification attacks is *k-anonymity* (Sweeney, 2002). In a k-anonymous dataset, each entry cannot be distinguished from at least $k-1$ other entries with respect to a set of attributes called *quasi-identifiers*. Thus, an attacker can only associate an individual to a group of at least k entries. In the context of location-based services, Bettini et al. (2005) define *location-based quasi-identifiers* (LBQID) as a sequence of spatio-temporal points that define a mobility pattern. Various techniques have been employed to achieve k-anonymity for trajectories, including spatial generalization (Yarovoy et al., 2009), clustering (Nergiz et al., 2008) and suppression (Terrovitis et al., 2017; Pensa et al., 2008). However, despite k-anonymity's protection against re-identification attacks, Ganta et al. (2008) demonstrated its susceptibility when information from multiple sources is combined. For instance, *attribute linkage attacks* aim to infer sensitive attributes from the data rather than specific records (Fiore et al., 2020). If all k trajectories associated with an individual lead to a hospital, the attacker can infer that the individual visited said hospital.

To address this issue, *ℓ-diversity* (Machanavajjhala et al., 2007) extends k-anonymity by ensuring that each anonymized group contains at least ℓ different sensitive attributes. Going further, *t-closeness* (Li et al., 2007) imposes a statistical criterion on the diversity of the sensitive attributes rather than a numerical one, as is the case with *ℓ-diversity*. Both concepts have been applied to trajectory data (Tu et al., 2019).

Differential privacy is a stricter privacy model than k-anonymity, requiring that the result of anonymization exhibits minimal variations depending on whether a specific entry is present or not (Dwork, 2008). While differential privacy has been applied to trajectories (Chen et al., 2012; Bonomi and Xiong, 2013; Gursoy et al., 2019), these methods are not suitable for trajectories with a high sampling rate.

3.3.2 LPPMs reducing the risk of privacy breaches

Location-Privacy Preserving Methods (LPPMs) providing a formal privacy guarantee often have the drawback that the data is altered to an extent that limits the utility of the data for subsequent analysis. In cases where this formal guarantee is not needed, heuristic solutions that reduce the risk of privacy breaches can yield higher utility of the anonymized data. According to Fiore et al. (2020), these methods include:

- **obfuscation**, where the trajectory is distorted by adding random noise to the location data (Seidl et al., 2016),
- **cloaking**, where the spatial and temporal granularity of the trajectory is reduced (Murakami et al., 2017),
- **segmentation**, where the trajectory is split into subtrajectories with the motivation that shorter trajectories are more ambiguous than longer ones (Song et al., 2014b), and
- **swapping**, where subtrajectories are exchanged between trajectories, such that each trajectory contains parts of multiple trajectories (Salas et al., 2018).

Additionally, some LPPMs focus on concealing sensitive locations (such as stay points) along the trajectory while leaving the remainder of the trajectory untouched (Nergiz et al., 2008; Huo et al., 2012). While these methods may not completely prevent re-identification attacks, they can mitigate the risk by removing crucial information like workplace or home location.

3.3.3 k-site-unidentifiability

Brauer et al. (2022) introduce *k-site-unidentifiability*, a privacy model to formalize the protection of sensitive locations by extending the concept of k-anonymity to such sites. In the model, given a set S of sites and a destination site $\tilde{s} \in S$, k-site-unidentifiability ensures that $\tilde{s}$ remains indistinguishable from at least $k-1$ other sites in S. This is achieved by concealing $\tilde{s}$ within a set $C(\tilde{s})$ of k sites, termed the *protection set*. Further, the authors define an attacker model and propose an algorithm to ensure k-site-unidentifiability with respect to this attacker model.

Attacker model. The attacker has access to a set of trajectories $\Theta = \{T_1, \ldots, T_l\}$ sharing a common destination $\tilde{s}$; for instance, this could be the contributor's home. Furthermore, the attacker knows a set of sites S that is guaranteed to contain $\tilde{s}$. S could, for example, be the set of all buildings present in a dataset. The attacker aims to identify the common destination based on the trajectories and the sites. The attacker employs several attack functions $f_1, \ldots, f_a$ to accomplish this. For each trajectory T, an attack function f yields a set of candidate sites $f(S, T) \subseteq S$. The attacker attempts to deduce $\tilde{s}$ by aggregating these candidates across all trajectories. For example, the attacker could assume that $\tilde{s}$ is the site that occurs most often in the joint set. The sites in the protection set $C(\tilde{s})$ are indistinguishable with respect to f if either all or none of the sites in $C(\tilde{s})$ are part of the attack's result for each trajectory in Θ. As the attacker knows all the sites in S, a mechanism that preserves k-site-unidentifiability cannot alter S. It can, however, alter the trajectories in Θ.

S-TT algorithm. The authors propose the Site-Dependent Trajectory Truncation (S-TT) algorithm, which selectively truncates trajectories to maintain k-site-unidentifiability. The S-TT algorithm truncates trajectories based on these attack functions, ensuring that k-site-unidentifiability is preserved. To that end, it iteratively shortens the trajectories until either all or none of the sites in $C(\tilde{s})$ remain viable candidates, thereby impeding the attacker's ability to distinguish $\tilde{s}$ from other sites in $C(\tilde{s})$.

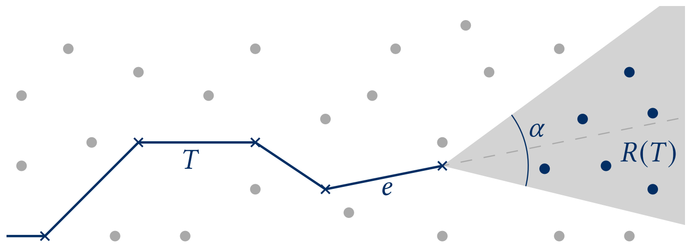


Figure 3.2: Schematic representation of a direction-based attack. The sites (round dots) inside $R(T)$ (colored black) are candidate sites this attack function returns. $R(T)$ is based on the terminating segment e of trajectory T.

Geometric S-TT. Specifically, the authors introduce a geometric S-TT algorithm by defining attack functions based on geometric characteristics of trajectories. Firstly, a proximity-based attack function f_p identifies the site closest to the trajectory's endpoint, building upon the intuition that trajectories often end near the user's destination:

$$f_\mathrm{p}(S,T) := \arg\min_{s \in S} \mathrm{dist}(\mathrm{end}(T), s), \tag{3.1}$$

where $\mathrm{dist}(\cdot,\cdot)$ denotes the Euclidean distance. This attack function can easily be extended to return the κ-nearest sites to $\mathrm{end}(T)$ to introduce some tolerance.

A second aspect that can be used to infer the destination is the direction in which the trajectory is headed. Specifically, the trajectory's last segment e is of interest. However, the segment e, most likely, does not point precisely towards $\tilde{s}$. Therefore, a V-shaped region $R(T)$ anchored in $\mathrm{end}(T)$ that is mirror-symmetric with respect to e and has an infinite radius is considered (Figure 3.2). The opening angle of the V-shape is fixed and denoted with α. The direction-based attack function f_d assumes that $\tilde{s}$ is located within the region $R(T)$:

$$f_\mathrm{d}(S,T) := S \cap R(T). \tag{3.2}$$

The attacks f_p and f_d are simple, yet common sense suggests that they may be reasonably successful. Figure 3.3 displays the geometric S-TT algorithm on an example trajectory. In the starting situation (Figure 3.3a), the closest site to $\mathrm{end}(T)$ is $\tilde{s}$. Thus, the algorithm suppresses the endpoint of T until the site closest to its endpoint is not part of the protection set $C(\tilde{s})$ of $\tilde{s}$ anymore (Figure 3.3b). At this point, some, but not all of the sites in $C(\tilde{s})$ are part of the candidate set for the direction-based attack (Figure 3.3b), violating k-site-unidentifiability for this attack function. Thus, truncation continues until either all the sites or none of the sites in $C(\tilde{s})$ are part of the candidate set of f_d anymore (Figure 3.3c). Note that, at this point, neither f_p nor f_d allow for a distinction between the sites in $C(\tilde{s})$. Thus, k-site-unidentifiability is preserved, and the algorithm is done.

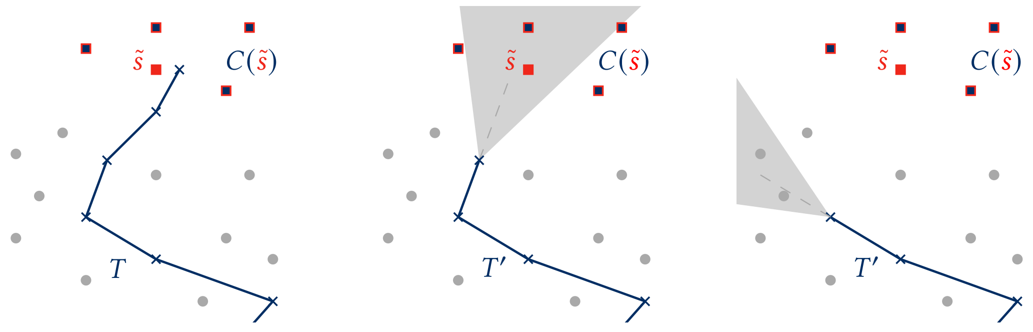


Figure 3.3: Schematic illustration of the geometric S-TT algorithm. (a) The original trajectory T leads to a destination site $\tilde{s}$, which has to be hidden among the sites in the protection set $C(\tilde{s})$, depicted as squares. The S-TT algorithm suppresses the trajectory's endpoint iteratively until (b) the site closest to the endpoint is not part of the protection set $C(\tilde{s})$ and (c) a V-shaped region that is aligned with the last segment of the truncated trajectory T' contains either all or, as in the depicted case, none of the sites in $C(\tilde{s})$.

Chapter 4

Map Matching for Semi-Restricted Trajectories

This chapter is mainly taken from collaborative work with Timon Behr, Thomas C. van Dijk, Jan-Henrik Haunert, and Sabine Storandt (Behr et al., 2021). While all co-authors contributed to various aspects of the article, their individual contributions are summarized as follows:

- Timon Behr conducted and documented the experimental evaluation of our approach (refer to Section 4.5).
- Thomas C. van Dijk focused on developing and implementing the extended network model (refer to Section 4.4.2, *Off-road candidates.*).
- I (Axel Forsch), with assistance from Jan-Henrik Haunert, developed the map-matching algorithm, including the energy model (refer to Section 4.4).
- Jan-Henrik Haunert and Sabine Storandt supervised the approach's development and the writing process.

The motivation for the work presented in this chapter stems from the field of trajectory analysis, which often requires associating information from the road network with the trajectory. In order to achieve this, map-matching algorithms are employed to find a corresponding path in the road network for each trajectory. This chapter covers the problem of matching trajectories to a road map, particularly considering trajectories that do not exclusively follow the underlying network. Such trajectories arise, for example, when a person walks through the inner part of a city, crossing market squares or parking lots. We call such trajectories semi-restricted. Sensible map matching of semi-restricted trajectories requires distinguishing between restricted and unrestricted movement. This chapter presents an approach that efficiently and reliably computes concise representations of such trajectories that maintain their semantic characteristics. The approach utilizes OpenStreetMap data to extract the network, areas allowing unrestricted movement (e.g., parks), and obstacles (e.g., buildings). This information is incorporated into the map-matching process. The method's applicability is demonstrated in an experimental evaluation of real-world pedestrian and bicycle trajectories.

4.1 Introduction

Map matching is the process of pinpointing a trajectory (given, e.g., as a sequence of GNSS measurements) to a path in an underlying network. The goal is to find the path that explains the observed measurements best. Map matching is often the first step of trajectory data processing as there are several benefits when dealing with paths in a known network instead of raw location measurement data:

- Both location measurements and geometric representations of roads are usually imprecise. Hence, constraining the trajectory to a path in a network is necessary to integrate the information given with a trajectory (e.g., recorded speed, vehicle type) with the information given with the road data (e.g., speed limit, road type). This is important to enable applications such as movement analysis or real-time navigation (Lou et al., 2009; Taguchi et al., 2018).
- Storing raw data is memory-intensive (especially with high sampling densities). Paths in a given network, on the other hand, can be stored very compactly, and many indexing methods have been developed that allow huge path sets to be queried efficiently (Funke et al., 2019; Krogh et al., 2016; Song et al., 2014a).
- Matching a trajectory to the road network enables data mining techniques that link attributes of the road network to attributes of the trajectories. This is used to deduce routing preferences from given trajectories (Sultan et al., 2017; Oehrlein et al., 2017; Funke et al., 2016; Brauer et al., 2021). In order to analyze and compare multiple trajectories, the trajectories need to be matched to a common network beforehand.

However, suppose the assumption that a given trajectory was derived from restricted movement in a particular network is incorrect. In that case, map matching might heavily distort the trajectory and possibly erase important semantic characteristics. For example, there could be two trajectories of pedestrians who met in the middle of a market square, arriving from different directions. After map matching, not only would the aspect that the two trajectories got very close at one point be lost, but the visit to the market square would be removed completely (if there are no paths across it in the given network). Not applying map matching at all might also result in misleading results, though, as the parts of the movement that actually happened in a restricted fashion might not be discovered and – as outlined above – having to store and query huge sets of raw trajectories is undesirable.

Hence, this chapter aims to design an approach that allows for sensible map matching of trajectories that possibly contain on- and off-road sections, which we call *semi-restricted trajectories*. The experimental evaluation shows that the approach computes paths that are significantly more compact than the raw trajectory data but, at the same time, still faithfully represent the original movement.

4.2 Related work

Due to its practical relevance, a vast body of work on map matching exists. Thus, this section focuses on related work most relevant to the envisioned application scenario.

Map matching for restricted trajectories. According to a recent survey on map matching (Chao et al., 2020), existing algorithms can be classified into four categories based on the underlying matching principle: similarity models, state-transition models, candidate-evolving models, and scoring models. Early map-matching approaches relied on geometric similarities between the road network and the recorded trajectories (White et al., 2000). One problem with these approaches is their sensitivity to measurement noise and low sampling rates. In order to overcome this problem, more sophisticated approaches try to model sensible transitions between consecutive states. In this context, e.g., Hidden Markov Models (HMM) (Haunert and Budig, 2012; Koller et al., 2015; Newson and Krumm, 2009) and reinforcement learning techniques (Osogami and Raymond, 2013; Zhang and Masoud, 2020) were applied successfully. Furthermore, incorporating guiding assumptions, such as that movements most likely follow shortest paths, can help compute more meaningful matches and decrease the running time (Eisner et al., 2011; Jagadeesh and Srikanthan, 2019).

Off-road and free space map matching. Map matching might also lead to artifacts in case the movement did indeed happen in an underlying network, but the network data is incomplete. Hence, several approaches that still produce high-quality matches in this scenario have been developed by allowing the path to contain off-road sections (Aggarwal et al., 2011; Haunert and Budig, 2012; Sasaki et al., 2019). However, these approaches rely on the assumption that the unrestricted movement sequences are short, which is not necessarily true when dealing with semi-restricted trajectories. Map matching for outdoor pedestrian trajectories with a high degree of freedom often requires additional data to yield good results, such as data derived from a smartphone accelerometer or compass (Ren and Karimi, 2012; Shin et al., 2010). In other lines of work, the goal is to model the possible movement network as precisely as possible (including, e.g., crosswalks) to be able to adapt conventional map matching techniques for restricted movement (Bang et al., 2016). Most map-matching approaches for pedestrians focus on indoor movement, though (Spassov et al., 2006; Wilk and Karciarz, 2014; Zhang et al., 2020). However, these approaches do not consider the possibility of switching between restricted and unrestricted movement and do not scale well enough to deal with large networks and rich map contexts.

Representation methods for unrestricted trajectories. Existing methods for storing and indexing raw trajectories are often based on a partitioning of the ambient space, e.g., into equally-sized grid cells or by constructing spatial data structures as CSE-Trees (Wang et al., 2008), ST-R-trees or TB-trees (Pfoser et al., 2000). However, these approaches do not feature any compression and are slow in case large trajectory sets are to be reported in a query. In contrast, map-matched trajectories can be managed much more efficiently in terms of memory and retrieval time using methods such as SPNET (Krogh et al., 2016), PRESS (Song et al., 2014a) or PATHFINDER (Funke et al., 2019). To gain these advantages when dealing with unrestricted trajectories, methods for computing an underlying graph for a given set of trajectories were investigated (Buchin et al., 2020). This task is closely related to map generation from trajectory sets (He et al., 2018; Li et al., 2019). However, these methods either discard outlier trajectories completely or produce rather large graphs, which is undesirable in our application. Further-

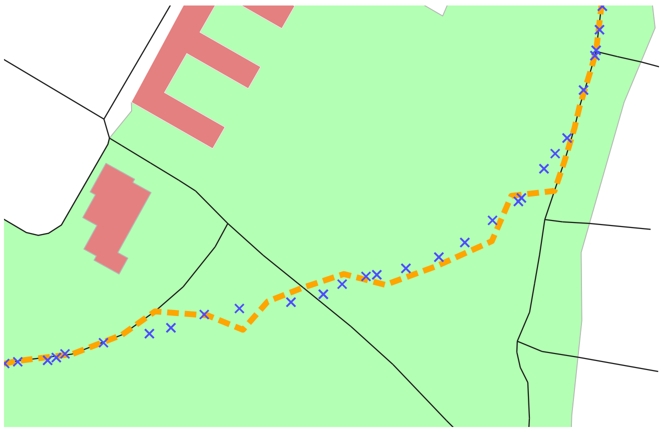


Figure 4.1: Example of a map with an embedded road network as well as free spaces and obstacles. The blue measurement-based trajectory is best explained by movement along the orange dashed path, which follows roads and free spaces where possible but contains no obstacle intersections.

more, the computed graphs have to be updated whenever new trajectories arrive. This chapter will discuss methods to produce graphs that cover free space areas so that all trajectories that may traverse said free space are represented sufficiently well. This problem has also been investigated in the context of indoor navigation. One approach is to represent the traversable space of floor plans by their skeleton, incorporating points of interest such as entrances and exits (Elias, 2007).

4.3 Contribution

This chapter presents a pipeline that can accurately match semi-restricted trajectories to a map, thereby overcoming several limitations of previous work. The following points are the main contributions:

- An extended network model capable of handling semi-restricted trajectories is introduced. In addition to the general graph representation of linear streets, tessellated free-space polygons that represent areas of unrestricted movement (such as marketplaces or parking lots) are added. An approach to obtain a sensible tessellation is discussed and an open-source implementation is provided at https://github.com/tcvdijk/tessa. *This contribution was mainly performed by Thomas C. van Dijk.*

- The map matching approach presented by Haunert and Budig (2012), originally developed to enable map matching on incomplete road network data, is extended significantly. First, the approach is made more efficient by incorporating ideas from Eisner et al. (2011). More importantly, the algorithm is modified to work on the extended network model. Among other things, this requires a carefully designed penalization scheme for off-road sections. An open-source implementation is provided at https://gitlab.igg.uni-bonn.de/graphlibrary. *This contribution was mainly performed by Axel Forsch.*

- Furthermore, obstacles are incorporated in the tessellation and the map-matching process. Obstacles are polygons the map-matched path cannot intersect (e.g., buildings for cycle trajectories). Taking obstacles into account requires additional processing steps but allows us to produce more meaningful results.

- The approach is evaluated in an experimental study on real-world trajectories. The evaluation investigates the accuracy of the computed paths, particularly concerning the recognition of unrestricted movement sections. Figure 4.1 shows an example outcome of the map-matching pipeline. *This contribution was mainly performed by Timon Behr.*

4.4 Methodology

In this section, the map-matching algorithm for *semi-restricted trajectories* is developed, starting with a baseline method for trajectories restricted to a network (Section 4.4.1) and extending it to deal with outliers, missing segments in the road data, and matches on designated free spaces (Section 4.4.2).

4.4.1 A baseline algorithm for trajectories restricted to a network

The algorithm is based on a state-transition model where each point of a given trajectory is represented by a set of matching candidates – the possible system states, i.e., positions of a moving subject. This is similar to HMM-based map-matching algorithms, which aim to find a sequence of positions maximizing a product of probabilities, each corresponding to a system state or a state transition. In contrast, the presented algorithm aims to minimize a sum of energy terms. By introducing weights for the energy terms that can be controlled with interactive sliders, the algorithm user has an intuitive tool for finding a good parameter setting.

Input requirements. The algorithm works on a trajectory $T = \langle p_1, \ldots, p_k \rangle$ of k points in $\mathbb{R}^2$, which we will call the trajectory points. Additionally, we have a directed, edge-weighted graph $G = (V, E)$ that models the transport network. Every directed edge $uv \in E$ corresponds to a directed straight-line segment representing a road segment with an allowed direction of travel. For a road segment that can be traversed in both directions, E contains the directed edges uv and vu. For an edge e, let $w(e)$ be its *weight*. In the basic setting, we will use the Euclidean length as the weight, but note that other values are possible.

System states. For every trajectory point p_i, we compute a set of matching *candidates* by considering a disk D_i of prescribed radius r around p_i: we select all road segments intersected by D_i and pick the point nearest to p_i on each. If such a point is not a vertex of G, we inject it as a new vertex v. (This means that we split each directed edge uw containing v into two directed edges uv and vw; we distribute the weight of uw to the edges uv and vw proportionally to their lengths.) Consequently, the set of matching candidates for p_i is a set of *vertices* $V_i \subseteq V$. If V_i is empty, we discard the trajectory point p_i and do not consider it for the matching.

State transitions. Possible transitions between candidates for two consecutive trajectory points are not modeled explicitly. Instead, they are implicitly modeled with the graph G by assuming that the transition between any two matching candidates $u \in V_i$ and $v \in V_{i+1}$ occurs via a minimum-weight u-v-path in G. Accordingly, the matched output path P is defined by selecting, for $i = 1, \dots, k$, one candidate v_i from the set V_i and connecting every pair of consecutive vertices in the sequence $\langle v_1, \dots, v_k \rangle$ via a minimum-weight path.

Energy model. To ensure that the output path P matches well to the trajectory, we set up and minimize an energy function that aggregates a state-based and transition-based energy:

- The *state-based energy* is $\sum_{i=1}^{k} \|p_i - v_i\|^2$, meaning that the energy increases quadratically with the Euclidean distance between a trajectory point p_i and the matching candidate v_i selected for it.
- The *transition-based energy* is $\sum_{i=1}^{k-1} w(P_{i,i+1})$, where $P_{a,b}$ is a minimum-weight v_a-v_b-path in G and $w(P)$ is the total weight of a path P (i.e., the sum of the weights of the edges of P). For now, we use the geometric length of an edge as its weight.

The two energies are aggregated using a weighted sum, parametrized with a parameter α_c. This yields the overall objective function quantifying the fit between a trajectory $\langle p_1, \dots, p_k \rangle$ and an output path defined with the selected sequence $\langle v_1, \dots, v_k \rangle$ of vertices:

$$\text{Minimize} \quad \mathcal{E}(\langle p_1, \dots, p_k \rangle, \langle v_1, \dots, v_k \rangle) = \alpha_c \cdot \sum_{i=1}^{k} \|p_i - v_i\|^2 + \sum_{i=1}^{k-1} w(P_{i,i+1}) \tag{4.1}$$

Algorithm. An optimal solution is computed using k runs of Dijkstra's algorithm on a graph that results from augmenting G with a few auxiliary vertices and edges. More precisely, we use an incremental algorithm that proceeds in k iterations. In the i-th iteration, it computes for the subtrajectory $\langle p_1, \dots, p_i \rangle$ of T and each matching candidate $v \in V_i$ the objective value $\mathcal{E}_i^v$ of a solution $\langle v_1, \dots, v_i \rangle$ that minimizes $\mathcal{E}(\langle p_1, \dots, p_i \rangle, \langle v_1, \dots, v_i \rangle)$ under the restriction that $v_i = v$. This computation is done as follows.

- For $i = 1$ and any vertex $v \in V_1$, $\mathcal{E}_1^v$ is simply the state-based energy for v, i.e., $\mathcal{E}_1^v = \alpha_c \cdot \|p_1 - v\|^2$.
- For $i > 1$, we introduce a dummy vertex s_i and a directed edge $s_i u$ for each $u \in V_{i-1}$ whose weight we set as $\mathcal{E}_{i-1}^u$; see Figure 4.2. With this, for any vertex $v \in V_i$, $\mathcal{E}_i^v$ corresponds to the weight of a minimum-weight s_i-v-path in the augmented graph, plus the state-based energy for v. Thus, we are interested in finding for s_i and every vertex in V_i a minimum-weight path. All these paths can be found with one single-source shortest path query with source s_i and, thus, with a single execution of Dijkstra's algorithm.

We observe that our objective function $\mathcal{E}(\langle p_1, \dots, p_i \rangle, \langle v_1, \dots, v_i \rangle)$ for the whole trajectory has the minimum value $\min_{v \in V_k} \{\mathcal{E}_k^v\}$ and is among the values we have computed. It is not difficult to maintain information during the run of the algorithm to enable the reconstruction of a solution attaining that value.

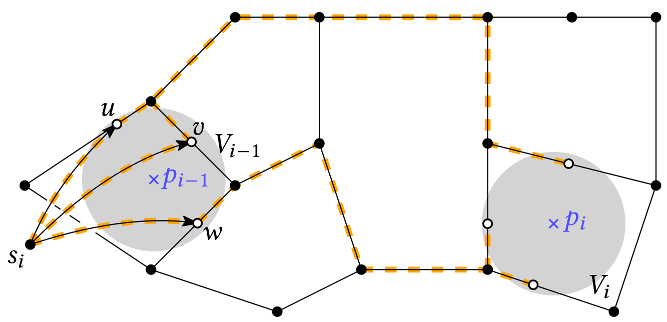


Figure 4.2: The incremental step of our algorithm. From the dummy vertex s_i, the shortest paths to all vertices in V_i are computed with one single-source shortest path query (see the dashed lines). The weight of each path plus the state-based energy of the corresponding terminal is used as the weight of a dummy edge in the next iteration.

4.4.2 Extensions for semi-restricted trajectories

The algorithm outlined in the previous section forces the output path to the road network, which can lead to unfavorable solutions. Consider the example in Figure 4.3, where road segments are missing in the road data. A long and unrealistic detour is generated by forcing the output path to the road network. More generally, we deal with the following issues:

- The trajectory contains outliers.
- The road data is incomplete, i.e., road segments are not modeled in the data.
- The trajectory traverses open spaces that are modeled as areas.

To deal with the first two issues, we allow trajectory points to be left unmatched and accordingly introduce *unmatched candidates*. The third issue is handled by augmenting the road network with additional edges, which we call *off-road segments*. Accordingly, a matching candidate on an off-road segment is called *off-road candidate*. These concepts are presented in detail below, followed by a description of the extension of the energy model.

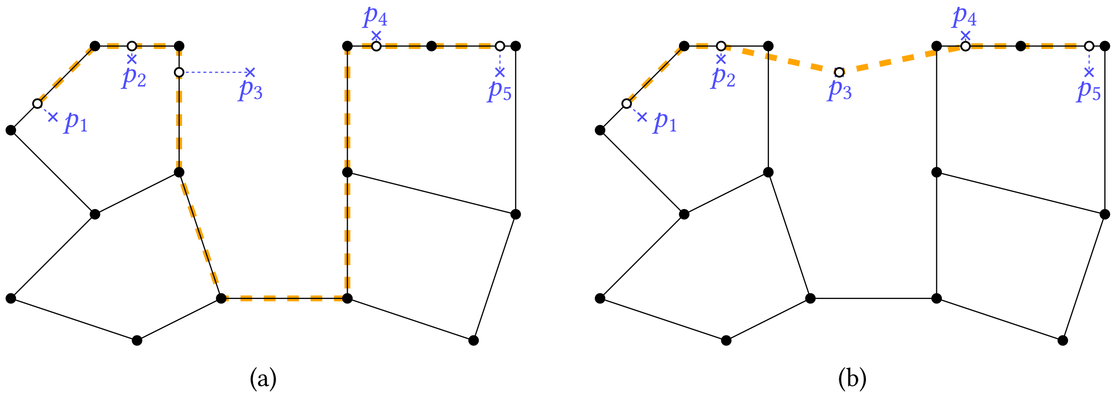


Figure 4.3: (a) Forcing the input trajectory (blue) to the network causes a detour in the output path (dashed). (b) Adding unmatched candidates and connecting segments prevents the detour.

Unmatched candidates. For each trajectory point p_i, we extend the set of candidates with a candidate at the observed location. Analogously to the baseline algorithm, we add this candidate into the graph by inserting it as a vertex u_i. For every vertex v_{i-1} in the candidate set of trajectory point p_{i-1} and every vertex v_{i+1} in the candidate set of trajectory point p_{i+1} we add directed edges $v_{i-1}u_i$ and u_iv_{i+1}, respectively. Figure 4.4 shows the extended graph for two consecutive GNSS points with the additional edges shown in green. This approach guarantees that there is always a fallback solution, which can be chosen if no path in the road network is similar to the trajectory. We will associate the additional edges with high energy (see below) to ensure they will be chosen only if necessary. In the following, we refer to the baseline method with the extension to unmatched candidates as Baseline+.

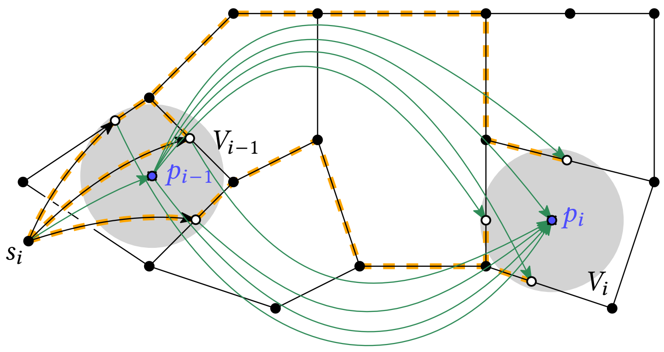


Figure 4.4: Extended graph to support unmatched segments. In addition to the candidates in the road network, every trajectory point is treated as a candidate itself. Additional edges (green) are introduced, connecting unmatched candidates (blue) to all candidates of the previous and next trajectory points.

Off-road candidates. Most data sources represent open spaces such as public squares, parks, or parking lots as polygonal areas. The algorithm described so far does not deal with this (see Figure 4.5a). We extend the road network by triangulating all open spaces and adding tessellation segments as edges into the graph (Figure 4.5b). In order to accurately represent polygons of varying degrees of detail and to provide an appropriate number of off-road candidates, CGAL's meshing algorithm (Boissonnat et al., 2000) is used with an upper bound on the length of the resulting edges and a lower bound on the angles inside triangles. The figure shows that this algorithm can introduce additional vertices to achieve these constraints. Note that original road segments can cross open spaces: this should be respected by the tessellation since on-road candidates will be preferred over off-road candidates.

Extensions of energy model. With the addition of the extensions to the baseline algorithm, we now have three different sets of edges in the graph: the original edges of the road network E_{r}, the edges incident to unmatched candidate vertices E_{u} and the off-road edges on open spaces E_{t}. We prefer on-road matches over off-road matches, while unmatched candidates are used as a fallback solution and thus should only be selected if no suitable path over on- and off-

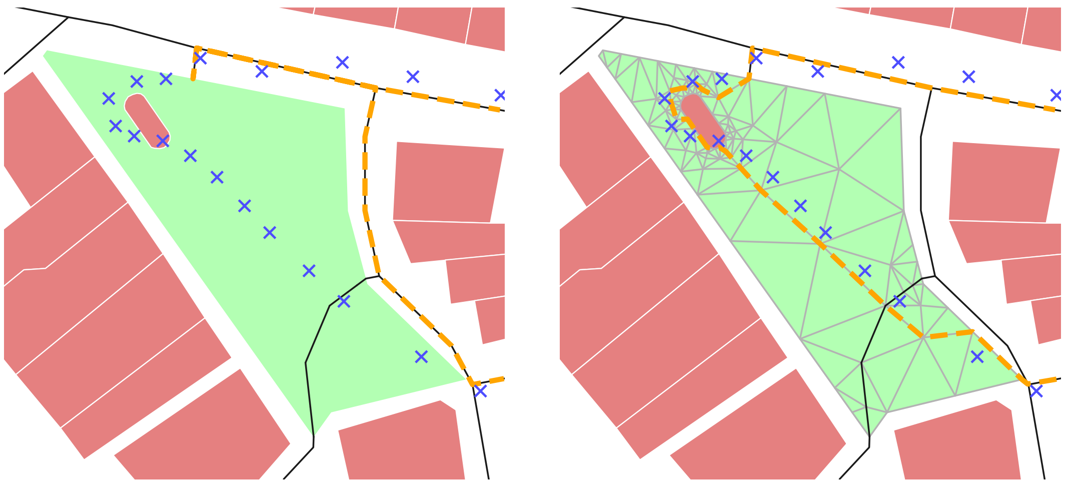


Figure 4.5: (a) Roads in the network data. Movement on the open space (green) cannot be matched appropriately (dashed). (b) Extended road network. Open spaces are tessellated in order to add appropriate off-road candidates. Note that the green polygon has a hole that remains untessellated since it is not open space. Also, note that additional vertices are added to prevent skinny triangles.

road edges can be found. To model this, we adapt the energy function by changing the edge weighting w of the graph. We introduce two weighting terms α_t and α_u that scale the weight $w(e)$ of each edge e in E_t or E_u, respectively. The edge weighting function thus is defined as:

$$w(e) = \begin{cases} \ell(e), & e \in E_\mathrm{r} \\ \alpha_\mathrm{u} \cdot \ell(e), & e \in E_\mathrm{u} \\ \alpha_\mathrm{t} \cdot \ell(e), & e \in E_\mathrm{t} \end{cases} \tag{4.2}$$

To favor matches in the original road network and keep unmatched candidates as a fallback solution, $1 < \alpha_\mathrm{t} < \alpha_\mathrm{u}$ should hold. Together with the weighting factor α_c for the state-based energy, our final energy function thus comprises three different weighting factors.

4.5 Evaluation

This section evaluates the presented algorithm on real-world data. For this, a user study was conducted around Lake Constance, Germany. The used data sources and the experimental setup are described in Section 4.5.1. In Section 4.5.2, a detailed analysis of the quality of the produced map-matching results is presented.

4.5.1 Experimental setup

In the experiments, data extracted from OpenStreetMap[1] (OSM) is used to model the road network. Trajectories of pedestrians and cyclists recorded within the scope of a user study are used as input trajectories. In the following, this setup is explained in detail.

Modelling the road network. The experimental region is composed of the area around Lake Constance, Germany. For this region, data from OSM is extracted to build the road network model used for matching. Elements tagged as highways are used to generate the road network as described in Section 4.4.1. To make the road network feasible for cyclists and pedestrians, all roads not traversable for these modes of transport are removed. Altogether, a road graph with 931,698 vertices and 2,013,590 directed edges was extracted.

The open spaces used for tessellation are identified by extracting polygons with special tags. A list of tags representing spaces with unrestricted movement and tags representing obstacles for movement was carefully selected. As the polygons extracted this way overlapped in some areas, the input was sanitized: if an obstacle overlapped an open space, the open space was cropped not to be covered by the obstacle. In the case that two open spaces overlapped, they were split such that no overlap exists in the final data. This way, 6,827 polygons representing open spaces were extracted.

The tessellation of the open spaces was performed with an upper bound of 25 meters for the length of the resulting edges. The lower bound for the inside angles of the triangles was kept at the CGAL default value of approximately 20.6°. These parameters were selected through visual inspection of tessellation results. Subsequently, employing these parameters resulted in a final graph that includes 1,148,213 vertices and 3,345,426 directed edges.

User study. A user study with five participants was conducted to evaluate the algorithm's performance. The participants were asked to record their cycling or walking trips with mobile phones or similar location-recording devices. After each trip, the participant annotated all sections of the trip where he or she left roads. This information serves as ground truth for the evaluation to determine whether these segments were identified correctly. In total, 58 trajectories were gathered during the study. The length of these trajectories varies from 300 meters to 41.3 kilometers, and they contain 66 annotated off-road segments.

4.5.2 Map matching results

The evaluation of the proposed map matching algorithm for semi-restricted trajectories is performed in two steps: In the first step, the sensitivity of the algorithm to parameter choices in the energy model discussed in Section 4.4.1 and Section 4.4.2 is investigated. Thereupon, a sensible parameter setting is deduced. In the second step, the matching results are analyzed more thoroughly by comparing them to the Baseline+ algorithm without tessellation.

[1] ©OpenStreetMap contributors

Parameter tuning. Three crucial parameters in the extended energy model govern the map-matching process. First, there are the edge cost coefficients α_t and α_u, which for values larger than one, penalize the usage of tessellation edges or edges incident to unmatched points, respectively, in comparison to the usage of edges in the road network. Furthermore, the parameter α_c allows us to fine-tune the impact of the Euclidean distance from a trajectory point to its matched point on the map in the objective function.

Including edges incident to unmatched points is considered a last resort. Thus, α_u is set to 10 in all experiments. This value is sufficiently large to avoid unmatched points whenever there are road network or tessellation edges in proximity. However, it is also small enough not to induce unreasonably long detours if the trajectory crosses an area with no roads and is also not a free space. For α_t and α_c, the best setting is less obvious, though. For too large α_t and too small α_c, the algorithm would match trajectories only to paths in the road network. Nevertheless, a too large value α_c would render the algorithm too inflexible to find reasonable paths, especially in the presence of outliers.

Hence, the following experiment is conducted to see how sensitive our algorithm is to the choice of α_t and α_c, and to figure out which configuration to use in subsequent studies: α_t is varied from 1.0 to 3.0 in increments of 0.1 and α_c is varied from 0.01 to 0.1 in increments of 0.01. Each of the resulting 310 combinations is assessed on how well the algorithm identifies restricted and unrestricted movement in our labeled trajectory benchmark set. More precisely, the true-positive rate of correctly recognized off-road segments (the higher, the better) and the false-positive rate of actual road parts treated as off-road segments (the lower, the better) are compared. The results are summarized on the left in Figure 4.6. The effect of certain parameter combinations for matching an off-road part is illustrated in Figure 4.6, right.

A significant range of value combinations is clearly dominated by others, in particular the ones with $\alpha_t > 2$ or $\alpha_c < 0.05$. For α_t close to 1.0, unsurprisingly, the highest true-positive rate is achieved; but also the highest false-positive rate for α_t in $[1.0, 2.0]$. The true-positive rate is relatively small for $\alpha_c < 0.5$. For α_c chosen larger than that, the precise choice is not that important as α_t then seems to have a more substantial influence on the achieved trade-offs. Furthermore, the precision and recall for the identification of on-road sections for all parameter combinations are computed with the results that precision and recall tend to be close to 1.0 for all tested α_t and α_c. Hence, the conclusion is that the algorithm is not overly sensitive to the precise parameter choice. For subsequent experiments we chose $\alpha_t = 1.1$ and $\alpha_c = 0.07$.

Quality analysis. In the parameter tuning experiment, the analysis was limited to identifying off- and on-road sections. The objective is to comprehensively analyze the overall matching quality and compare the extended approach for semi-restricted map matching to Baseline+, which does not use tessellated free spaces. This comparative evaluation will focus on two aspects: (i) shape preservation of the input trajectories and (ii) compression and indexing of trajectory sets.

The Fréchet distance and the Dynamic Time Warping (DTW) distance are computed between the input and the output curve to measure the preservation of the original trajectory's shape by the map-matching approach. These two distances are often applied to quantify shape preservation and are defined as follows.

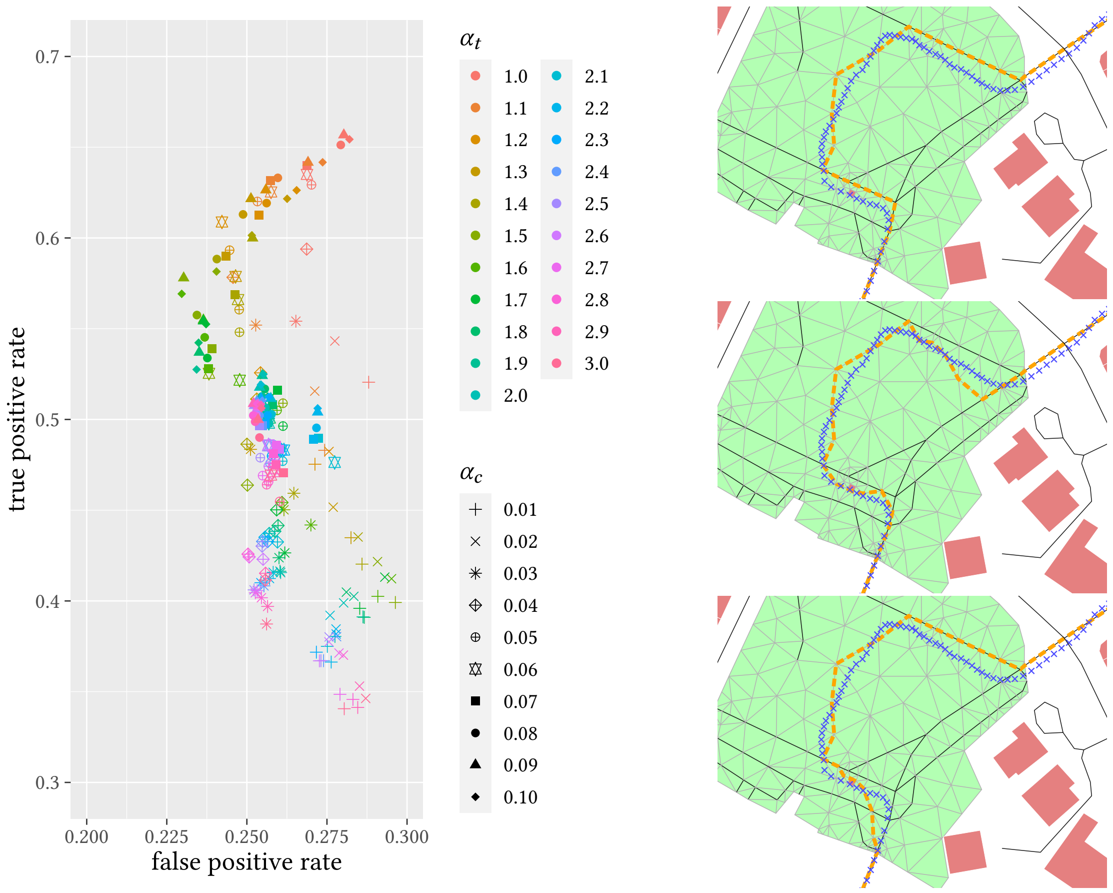
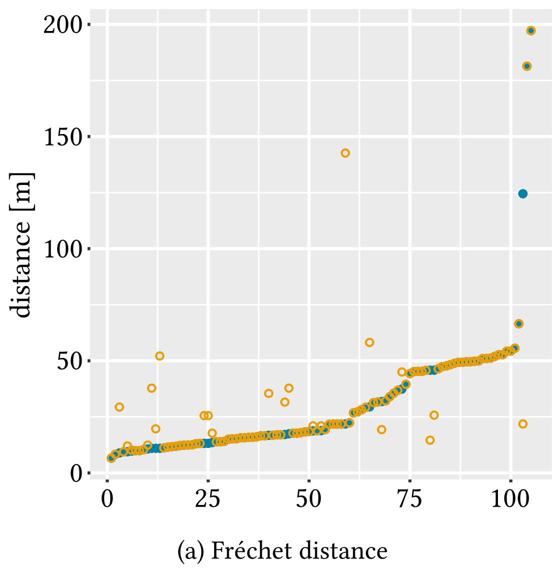


Figure 4.6: Left: Classification results for off-road section identification for different parameter configurations. Right: Effect of the parameter choice illustrated for an example instance. In the upper image ($\alpha_t = 1.5, \alpha_c = 0.01$), the trajectory points (blue crosses) are matched to the cheap road network edges whenever possible. In the middle image ($\alpha_t = 1.5, \alpha_c = 0.10$), the increased candidate cost leads to a matching which deviates from the measurements as little as possible. In the lower image ($\alpha_t = 1.1, \alpha_c = 0.01$), the decreased tessellation edge cost leads to more deviation from the road network as in the upper image. Despite those differences, the middle part is matched to the same edges for all three configurations.

Definition 3 (Fréchet Distance)**.** Given two polylines L, L' with length k and k', respectively, the Fréchet distance is defined as $d_F(L, L') := \inf_{\sigma,\theta} \max_{t\in[0,1]} \left\| L(\sigma(t)) - L'(\theta(t)) \right\|_2$ where $\sigma\colon [0,1] \to [1,k]$ and $\theta\colon [0,1] \to [1,k']$ are continuous and monotonic and $\sigma(0) = \theta(0) = 1, \sigma(1) = k$ and $\theta(1) = k'$.

Definition 4 (DTW Distance)**.** Given two polylines L, L' with length k and k', respectively, the DTW distance is the cost of the cheapest warping path between L and L'. A warping path is a sequence $p = (p_1, \dots, p_W)$ with $p_w = (l_w, l'_w) \in [1:k] \times [1:k']$ for $w \in [1:W]$ such that $p_1 = (1,1)$ and $p_W = (k, k')$, and $p_{w+1} - p_w \in \{(1,0),(0,1),(1,1)\}$ for $w \in [1 : W-1]$. Thereby, the cost of path p is defined as $\sum_{i=1}^{W} \left\| p_i \right\|_2$. The normed DTW distance is the path cost divided by $2 \cdot \max\{k, k'\}$.

The Fréchet distance is a bottleneck measure, sometimes even used as the main objective for map matching (Wei et al., 2013). DTW is frequently used in similarity analysis of time series and other sequenced data. It has the advantage that its value is determined by the whole shapes and not just by a local dissimilarity maximum. The normed DTW distance is used for the analysis, allowing for better comparability between input trajectories of different lengths. Figure 4.7 shows the Fréchet and DTW distances for all off-road sections identified in the user study; map matched by the tessellation-based algorithm and Baseline+. Although Baseline+ is



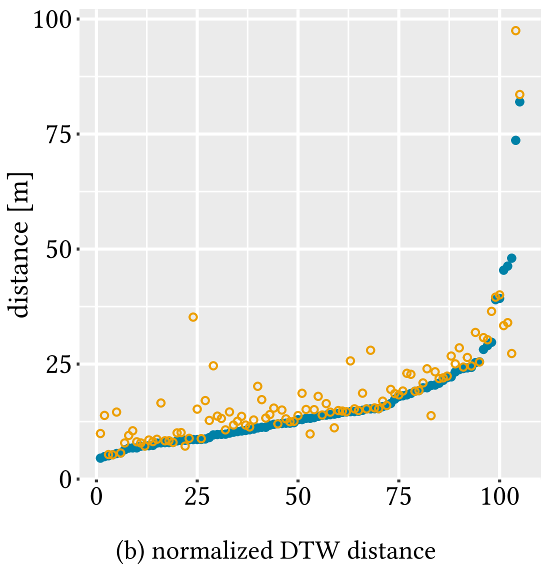


Figure 4.7: Fréchet distance (a) and normalized DTW distance (b) between input trajectories and the matched paths produced with the tessellation-based approach (blue) and Baseline+ (orange). In both cases, the trajectories (distributed over the x-axis) are sorted by the Fréchet distance (resp. normalized DTW distance) of the tessellation-based approach.

more likely to keep original trajectory points in the map-matched path whenever free spaces are traversed—and hence, the shape therein should be perfectly preserved—the tessellation-based approach achieves comparable or even better shape similarity on most inputs. On average, paths computed with the tessellation-based approach had a Fréchet/DTW distance to the original trajectory of 31.81/16.25, while for Baseline+ the respective values are 32.52/18.47. A possible reason for the better shape preservation with the tessellation-based approach is that transitions between restricted and unrestricted movement tend to be smoother; see Figure 4.8.

Next, further evaluation is performed to determine whether the tessellation-based addition of roughly 1.3 million edges to the road network proves beneficial not only for shape preservation and unrestricted movement identification but also for compression purposes. To this end, holes in the matched paths are counted. Here, a hole is a continuous subpath that is not represented by edges in the created graph but uses edges incident to unmatched points. The usage of Baseline+ led to 43% more holes than the tessellation-based approach. Furthermore, the holes resulting from Baseline+ are typically significantly longer. Holes require additional storage and indexing effort as the contained edges are unlikely to be used by any other trajectories. In contrast, tessellation edges may be traversed by many trajectory matches; see Figure 4.9 (left). This not only allows for better compression of trajectory sets but also helps to efficiently retrieve and analyze similar trajectories and extract common movement patterns within a trajectory set. Figure 4.9 further shows an example where Baseline+ computes a nonsensical match for a trajectory that follows the shoreline but is then forced to follow existing paths, but also an example where the tessellation-based approach fails due to a trajectory traversing an open space which was not chosen for tessellation based on the existing OSM tags. However, sensible matches using tessellation edges were identified for most of the tested instances.

4.6 Conclusion

This chapter introduced the notion of semi-restricted trajectories. It proposed a method for map matching such trajectories to a carefully constructed graph consisting of the road network and tessellated free spaces. It is shown that the resulting combined graph is only about 50% larger than the road network but enables a faithful representation of restricted and unrestricted movement at the same time. As evidenced in the experimental analysis, unrestricted movement is, in most cases, correctly identified as such by the presented map-matching approach. Nevertheless, many interesting directions for future work could improve or extend the approach. For example, different tessellation variants could be tested and compared. Currently, an upper bound for the edge length allowed in the free space triangulations and a lower bound for the angular resolution is used. The two bounds could be varied to achieve different precision and compression trade-offs for the matched trajectories. Other tessellation approaches, e.g., based on polygon skeletons, could further help model natural movement in free spaces. Furthermore, one could consider trajectories that enter and exit buildings similarly and try to reliably infer the respective transitions from outdoor to indoor or vice versa. At the moment, all buildings are treated as obstacles, but it is also possible to apply the model to the interior of buildings to allow for indoor matching. Finally, while the approach is sufficiently fast with query times below 0.5 seconds per trajectory, more efficient match computation might be possible by improved candidate vertex selection, especially within the tessellated free spaces.

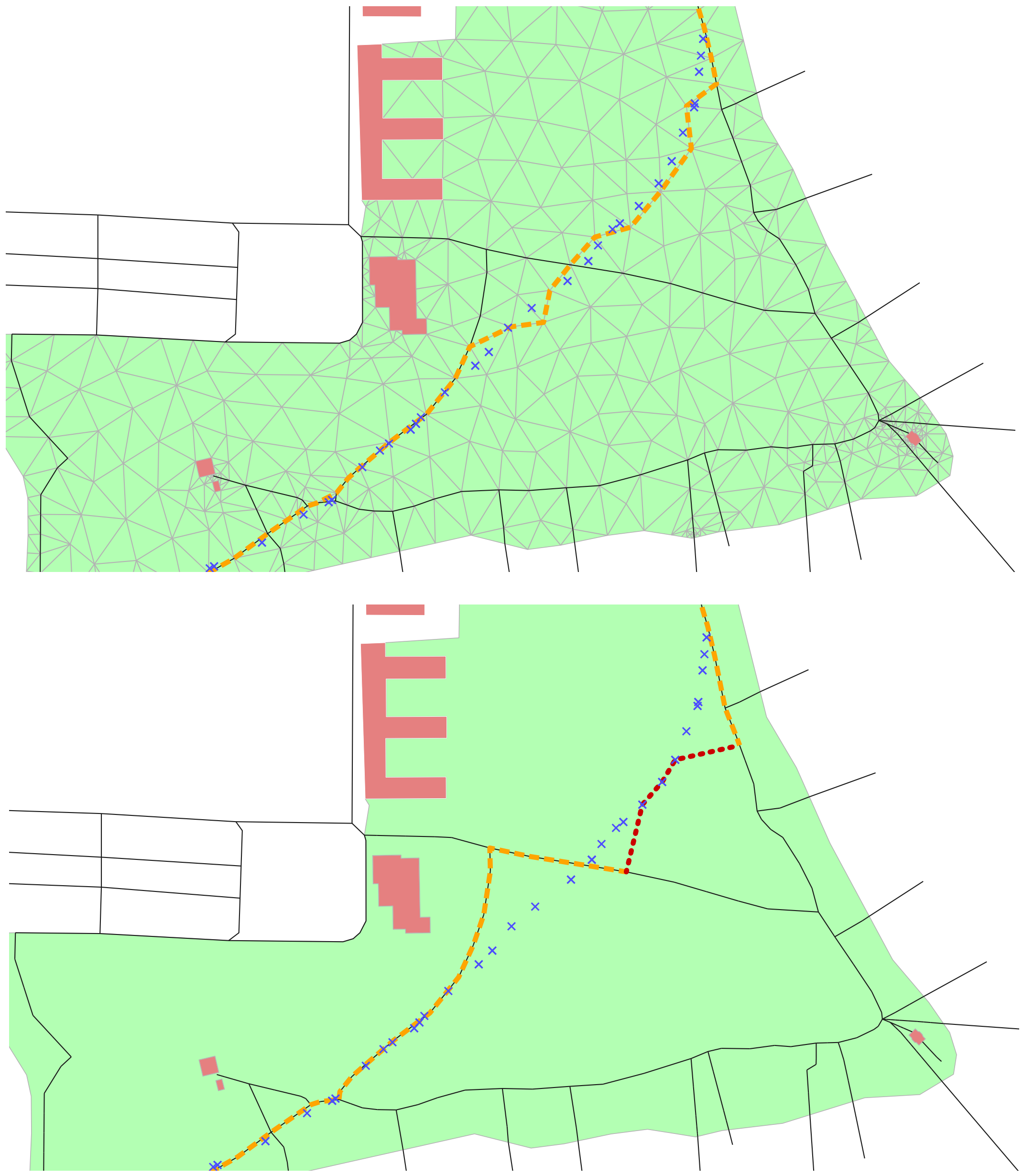


Figure 4.8: Original trajectory measurements (blue crosses) and matched paths produced by the tessellation-based approach (upper image) and Baseline+ (lower image). In tessellation-based matching, the shape of the original trajectory is faithfully preserved, and the unrestricted movement is correctly identified and matched to tessellation points and edges. Using Baseline+, the first part of the unrestricted movement is wrongfully matched to the path network, and subsequently, the shape is locally distorted when the switch from restricted to unrestricted movement occurs.

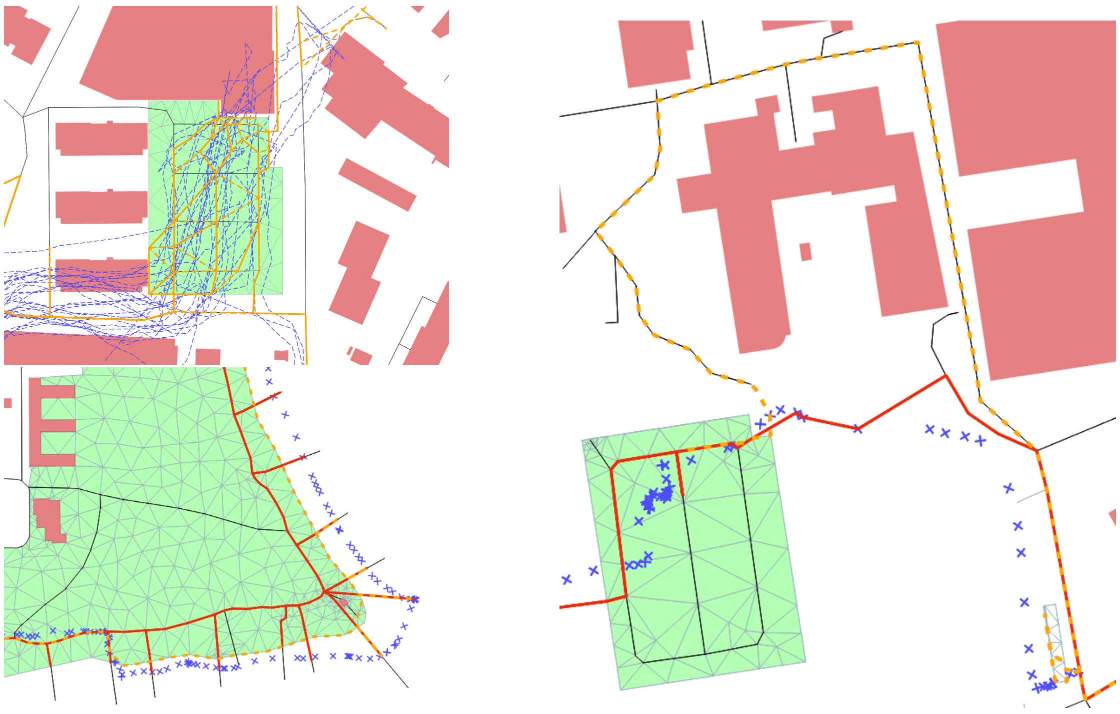


Figure 4.9: Upper left: Trajectories (dashed blue lines) traversing the same free space. The yellow edges indicate their matched paths. Lower left: A trajectory (blue crosses) following the shoreline of the lake. The Baseline+ approach matches this trajectory to far-away paths and several groins, while the tessellation-based approach produces a more sensible match. Right: Without sufficient tessellation edges, a trajectory traversing a large free space might be matched to a path with a rather large detour.

Part II

Inferring Routing Preferences from Trajectories

Chapter 5

Inferring Routing Preferences: Motivation & Research Context

The increasing urbanization and the accompanying increase in urban population strain the urban transportation system, leading to congestion, high levels of pollution, and noise emissions (Sumantran et al., 2017). At the same time, physically inactive lifestyles are a major public health challenge today. Active modes of transportation, such as walking or cycling, can positively affect both of these issues (Sallis et al., 2004). Therefore, urban transportation planning experiences a shift away from private motorized transport towards transportation modes such as walking, cycling, and public transport (Bertolini, 2020). The share of cycling as a mode of transportation differs regionally, and infrastructure that meets cyclists' requirements can encourage cycling (Dill and Carr, 2003; Hull and O'Holleran, 2014). Therefore, much research into bikeability as a measure to quantify the suitability of urban networks for cycling has been performed (Reggiani et al., 2022). Next to the general suitability for cycling, personal preferences and movement intent play a crucial role in cycling (Sultan et al., 2017; Oehrlein et al., 2018). A short introduction to bikeability is given in the following, followed by an overview of personal routing preference modeling.

5.1 Bikeability

In the context of transportation research, measures to assess the quality of the cycling infrastructure have been developed. One frequently cited term in this domain is *Bikeability*. Despite its widespread use, the concept of bikeability lacks a universally agreed-upon definition, as noted by Kellstedt et al. (2021). One commonly referenced definition is given by Lowry et al. (2012), who characterize bikeability as "an assessment of an entire bikeway network for perceived comfort and convenience and access to important destinations".

While definitions vary, Arellana et al. (2020) identify five factors commonly considered in quantifying bikeability: comfort, directness, coherence, attractiveness, and safety. Building upon these factors, different bikeability indices have been introduced, including the Bikability Index (BI, Lowry et al., 2012; Tayebeh Saghapour and Thompson, 2017), Bicycle-Level-of-Service (BLOS, Landis et al., 1997), Bicycle Level of Traffic Stress (BLTS, Chen et al., 2017a;

Kent and Karner, 2019), the Interaction Hazard Score (IHS, Landis, 1994), and the Bicycle Compatibility Index (BCI, Harkey et al., 1998). These indices have been employed to evaluate the Bikeability for numerous cities such as Graz (Krenn et al., 2015), Gothenburg (Manum et al., 2017), Basel (Grigore et al., 2019), Munich (Schmid-Querg et al., 2021), and Berlin (Hardinghaus et al., 2021), among others.

Crowd-sourced data holds significant relevance for quantifying factors influencing bikeability. For example, in the SimRa project (Karakaya et al., 2020) data on accidents, near-miss incidents, and bike routes are collected through crowd-sourcing to gain insights into cycling safety. Similarly, information on available bike parking is inferred from social media images (Knura et al., 2021; Bimbao et al., 2022).

5.2 Individual routing preferences

The preceding section introduced bikeability as a metric used to assess the suitability of urban infrastructure for cycling. However, while bikability aims to capture the cycling behavior of the entire population, cyclists have heterogeneous routing preferences. Consequently, a single routing model that suits all individuals typically does not exist (Reggiani et al., 2022). Moreover, even for a single individual, preferences may change depending on the purpose of the trip. For example, while commuting, one might prioritize the shortest route for efficiency, whereas, during recreation trips, scenic routes may be preferred (Krizek et al., 2007). Hence, much research has been performed to identify individual routing preferences.

Explicit vs implicit preferences. In psychology, two kinds of preference are distinguished: explicit and implicit preferences (Nosek, 2007; Wilson et al., 2000). Explicit preferences describe which preferences the user will state by themself when specifically asked for them. While the answer to this question gives valuable insight into the incentives and deterrents of the user, it is not clear whether the user knows their preference or whether the answer might be biased due to social pressure (Greenwald and Banaji, 1995). In contrast to that, implicit preferences correspond to subconscious behavior. Learning these preferences from questionnaires is impossible; they can only be derived from user actions (Friese et al., 2006). In some literature on routing preferences, explicit and implicit preferences are also referred to as *stated* and *revealed* preferences (Menghini et al., 2010; Rossetti et al., 2018; Hardinghaus and Weschke, 2022).

Discrete route choice modeling. Discrete route choice modeling is frequently applied in transportation science to model route choice behavior. These models have two components: generating a choice set comprising alternative routes between a given origin and destination and evaluating the selection among these alternatives (Ben-Akiva et al., 2004).

Several algorithms have been developed for the choice set generation. For instance, Bovy and Fiorenzo-Catalano (2007) propose a the *Doubly-Stochastic Generation Function* (DSGF) that creates alternative routes in a multi-criteria routing model by randomly assigning costs and cost coefficients to network links for each shortest-path query. In contrast, Hoogendoorn-Lanser et al. (2006) apply a rule-based approach for choice set generation, where logical constraints ensure route diversity. Rieser-Schüssler et al. (2013) introduce the *Breadth First Search* *on Link Elimination* (BFS-LE) algorithm. This algorithm iteratively determines the least-cost path between the origin and destination and subsequently introduces route diversity by systematically eliminating links within the network. Knapen et al. (2016) present a method to segment a path into a minimal number of concatenated least-cost paths. While they do not learn routing preferences directly, they highlight how to use this information in route choice problems. In specific, the distribution of the size of the minimum decompositions can be used for choice set generation. Our work is closely related to these authors' work, but instead of using geodesic shortest paths, we consider preference-weighted costs that are a linear combination of different basic costs.

Given the route choice set and the user's selected route (either through questionnaires or real-world observations in the form of GNSS trajectories), a routing preference is computed (Ben-Akiva et al., 2004). The Path Size Logit model, introduced by Ben-Akiva and Bierlaire (1999), is the most prevalent choice model in the routing context. It is a Multinomial Logit Model (Washington et al., 2009) with correction terms for overlapping paths. The quality of the analysis results is highly dependent on the choice set. Therefore, Halldórsdóttir et al. (2014) conducted an efficiency analysis of choice set generation methods by comparing them with real-world trajectories. Their findings suggest that both the BSF-LE and DSGF approaches generate realistic routes. However, the discrete nature of choice set generation poses a limitation, as it is impractical to enumerate all alternative routes.

Continuous preference mining. Continuous preference models have been applied in contrast to modeling routing behavior as a decision problem between discrete route choices. Much research has been done to predict user behavior by analyzing the historical trajectories of the users (Chen et al., 2011). These methods, however, strongly rely on frequently repeated routes, such as a user's daily trip to work. Hence, models that have been learned that way usually cannot be applied in areas where no trajectories have been recorded. Probably the first method that infers routing preferences from a sparse set of trajectories has been presented by Balteanu et al. (2013). Their algorithm can even be applied to single trajectories. However, their method has two drawbacks with respect to the practical relevance: Firstly, the algorithm yields reasonable results only if the trajectory is close to the *skyline*, i.e., the set of all Pareto-optimal paths, and thus cannot be applied to round trips. Secondly, the learned preference cannot be expressed as a single edge cost and, thus, it cannot be used to compute new routes with standard routing algorithms, such as the algorithm by Dijkstra (1959). Since then, further methods that apply to single trajectories have been developed. Funke et al. (2016) presented a method for inferring the routing preferences based on linear programming. Their algorithm requires, however, that a user's route is optimal for at least one linear combination of the edge costs. This is often not the case for longer routes. Therefore, several publications focus on combining trajectory segmentation and preference mining. Similarly, Barth et al. (2020) extended the approach of Funke et al. (2016) by segmenting a path into as few as possible subpaths, such that for each subpath, a linear combination of the edge costs can be found, for which it is optimal. However, as each subpath is explained with a different linear combination, it is not obvious how to calculate new routes from the results. In a later publication, Barth et al. (2021) cluster trajectories by searching a minimum set of routing preferences so that each trajectory is optimal for at least one of these preferences. Compared to the approach presented in Chapter 6, their approach can deal with an arbitrary dimension of the preference vector. However, the authors did not provide an upper bound on the running time of their algorithm. Additionally, the authors refer to trajectory segmentation approaches to guarantee that there is an optimal routing preference for each (sub-)trajectory. This segmentation is already included in the approach presented in Chapter 6. In recent years, deep learning approaches have been applied to preference mining. For instance, Yu et al. (2020) propose a Recurrent Graph Network model designed to deal with the sparsity and uneven distribution of trajectory data. Similarly, Marra and Corman (2021) use a convolutional neural network to model route choice in public transportation networks. While these deep learning approaches have the advantage that they can capture non-linear routing objectives, they typically require large amounts of training data, which is often not available.

Bicycle routing. Previous studies in the field of routing preferences examined the explicit preferences of cyclists. Sener et al. (2009) performed a survey on preferred route attributes and found that cyclists prefer routes with no on-road parking, low traffic volume, continuous bicycle facilities, and lower roadway speed limits. In a further analysis, they evaluated additional detours a cyclist is willing to take to stick to these preferences. Emond et al. (2009) and Majumdar et al. (2015) identified perceived safety as a major incentive for route choice. Based on this research, Rossetti et al. (2018) propose a choice model to aid cycling infrastructure planning. Similarly, Ehrgott et al. (2012) present a route choice model based on a bicriterial routing model. They analyze the trade-off between travel time and suitability for cycling that cyclists are willing to make. Hardinghaus and Nieland (2021) use a different approach for learning routing preferences. In their work, they analyzed the routing settings of users of a public bike routing engine. They conclude that minimizing the trip's length, using minor roads, and having a high-quality surface heavily outweigh the importance of dedicated cycling infrastructure. The diversity of the route preferences among cyclists is highlighted by Hardinghaus and Weschke (2022), who show that route preferences vary significantly between different demographic groups. According to the authors, especially for vulnerable road users, such as older people, dedicated bicycle infrastructure is important. The results of these studies allow for a pre-selection of routing criteria to be analyzed in the context of the implicit routing preferences of cyclists.

Chapter 6

An Algorithm Using a Compression Criterion

The following chapter is based on joint work with Johannes Oehrlein, Benjamin Niedermann, and Jan-Henrik Haunert (Forsch et al., 2023). An outline of a preliminary version of this algorithm has been published by Oehrlein et al. (2017). With support from Johannes Oehrlein, I (Axel Forsch) have improved the efficiency of the preliminary algorithm. Benjamin Niedermann and Jan-Henrik Hauert supervised me in developing and writing-up the improved algorithm, representing its first detailed publication.

The presented algorithm for inferring routing preferences is tailored for sparse sets of crowd-sourced trajectories. It addresses two main challenges often overlooked by existing methods: (a) the small number of trajectories provided per user and (b) the highly heterogeneous nature of the input, including round trips or trips with long detours that traditional preference models cannot explain. These challenges are tackled using a compression-based model predicated on the assumption that trajectories consist of optimal subpaths.

The optimal path between two vertices in a graph depends on the optimization objective, often defined as a weighted sum of multiple criteria. When integrating two criteria, their relative importance is expressed with a balance factor α. This chapter presents a new approach for inferring α from trajectories. The core of the approach is a compression algorithm that requires a graph G representing a transportation network, two edge costs modeling routing criteria, and a path P in G representing the trajectory. It yields a minimum subsequence S of the sequence of vertices of P and a balance factor α, such that the path P can be fully reconstructed from S, G, its edge costs, and α. By minimizing the size of S over α, we learn the balance factor that corresponds best to the user's routing preferences. In an evaluation with crowd-sourced cycling trajectories, the usage of official signposted cycle routes against other routes is weighted. More than 50% of the trajectories can be segmented into five optimal subpaths or less. Almost 40% of the trajectories indicate that the cyclist is willing to take a detour of 50% over the geometric shortest path to use an official cycle path.

6.1 Introduction

Route planning tools play an important role in modern mobility. They are used for commuting, i.e., to find the best path from one location to another, and for leisure activities, such as scenic cycling round trips that start and end at the same location. For both applications, it is crucial to choose the optimization objective, i.e., the definition of "best" such that it reflects a user's preferences. Classically, an optimal path from s to t is the geodesic shortest path in a road network, i.e., it is optimal with respect to the Euclidean length of the path's road links or the path with the shortest travel time. However, many more criteria (e.g., number of traffic lights, total ascent) can be applied. Hence, the definition of an optimal path varies widely. To provide users with good route planning, we must work with their personal preferences. Take the bicycle trajectories depicted in Figure 6.1 as an example. Between s and t, the red trajectory corresponds to the length-minimal path in the network. The question arises as to why the cyclist who recorded the yellow trajectory prefers a longer trip over the length-minimal path. In the concrete example, the cyclist used a path that, in contrast to the length-minimal path, consists of paths that are part of signposted cycle routes (green). Thus, a likely explanation is that the cyclist tried to use recommended paths, still trying to keep the length of the trip short.

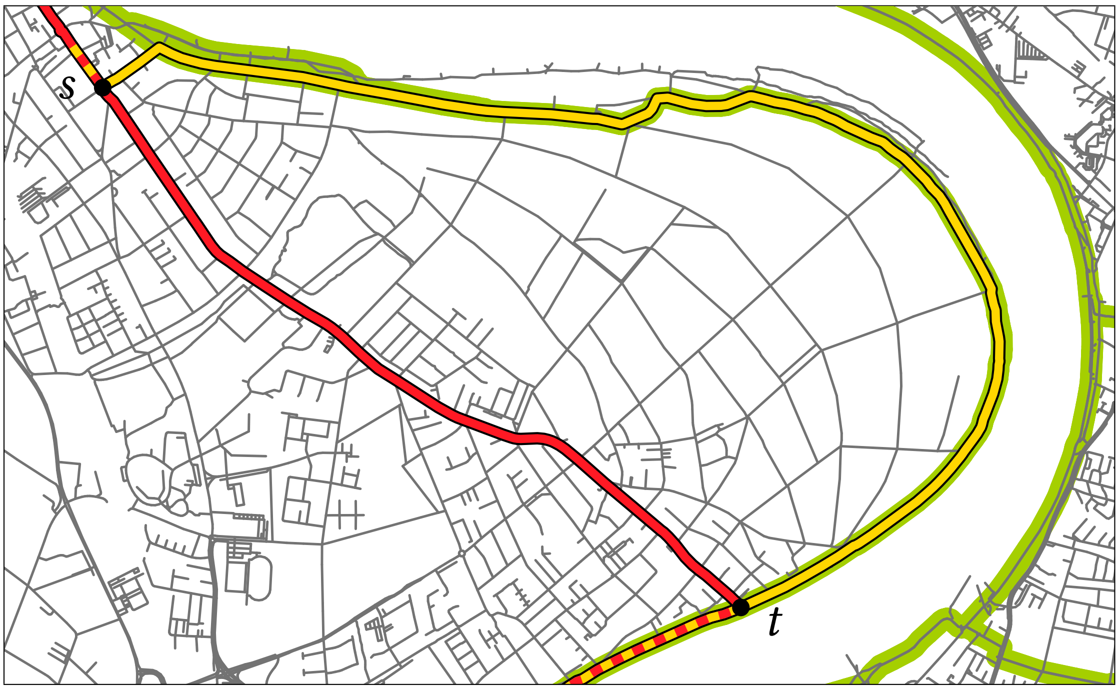


Figure 6.1: Extract of the road map of the German city of Cologne. The road network segments (gray) that are part of official cycle routes are presented with a green background. Furthermore, there are two bicycle trajectories with different s-t paths (red and yellow). The red trajectory is optimal with respect to distance and avoids official cycle routes completely. It has a length of 3.4 km. The cyclist of the yellow trajectory used a path with a length of 5.9 km. It consists mainly of official cycle routes, namely 95% of its length.

While this observation applies to that particular trajectory, a systematic analysis of hundreds of trajectories promises statistical evidence on the chosen optimization criteria and the extent to which the criteria have been taken into account. These insights into the users' preferences and priorities concerning the optimization criteria may help both individual route planning and infrastructure planning by appropriately balancing the applied optimization criteria. In the following, we distinguish between implicit and explicit routing preferences. In the example given in Figure 6.1, it may be valid to conclude that the cyclist who generated the yellow trajectory had an *implicit* preference for signposted cycle paths since the trajectory data suggests a strong bias towards such paths. However, the *explicit* preferences of the cyclist might have been to choose scenic routes or routes with low traffic volumes—conditions that are often found at signposted paths. This chapter focuses on inferring the user's implicit preferences based on their recorded trajectories.

Large sets of crowd-sourced trajectories are collected in the context of Volunteered Geographic Information (VGI, Goodchild, 2007). A key characteristic of these datasets is that they contain many trajectories of recreational activities. These recreational activities often do not aim to minimize a specific optimization criterion but to have a scenic route of a specific length. As such, they contain significant detours that often cannot be explained by a single (combined) optimization criterion. Thus, many established approaches for preference inference fail to analyze these trajectories. Take the example of a round trip: the corresponding trajectories start and end at the same location. Therefore, they are not optimal for any optimization criterion. Nonetheless, it seems natural that the user stuck to favorable roads as much as possible, resulting in multiple optimal subtrajectories. Appropriate tools are needed to analyze the rich information source provided by VGI sources concerning the underlying implicit optimization criteria. In this chapter, we suggest such a tool. Our approach is to determine the parameter of a routing model such that a given trajectory (or set of trajectories) can be partitioned into a minimum number of optimal paths with respect to that model. We base our approach on the findings of Gotsman and Kanza (2013) and Lerin et al. (2013), who have, independently of each other, shown how to compute a compact representation of a user's path by partitioning it into a minimum number of optimal paths. The authors have noted that the better the optimization criterion used by the optimal-path algorithm matches the user's routing preferences, the more compact representations are obtained. In this chapter, we reverse this argument by arguing that if a routing model induces a partition of a trajectory into a small number of optimal paths, then it reflects the user's preferences well. This adheres to the Minimum Description Length Principle, introduced by Grünwald et al. (2005), which states that the best explanation for a given data set is the one with the highest compression. The presented approach aims to infer routing preferences from a wide range of trajectories, e.g., trajectories of commuters and recreational cyclists. In particular, we neither require that the trajectories have been chosen to minimize a specific optimization criterion nor do we exclude such trajectories.

A preliminary version of the algorithm presented in this publication has already been used by Oehrlein et al. (2017, 2018) to analyze the influence of the slope on bicycle routing and to approve a classification of road types into favorable and unfavorable types. Both of these publications focus on the application of the algorithm without explaining its functionality in detail. This chapter gives a detailed presentation of the algorithm. Additionally, the computational complexity of the algorithm is improved. The algorithm's applicability is shown by its ability to analyze whether and how cyclists prefer signposted cycling routes over other routes. In the experiments, the algorithm classifies the trajectories in the data set into three classes, with the trajectories in the class representing cyclists willing to take detours to stick to signposted cycling routes accounting for 37% of all trajectories.

The chapter is structured as follows. An overview of related work is given in Section 6.2. In Section 6.3, the approach for finding the optimization criterion corresponding to the minimal segmentation of the input path is outlined. The results of the experimental evaluation of the approach are presented in Section 6.4. The chapter is concluded in Section 6.5. The implementation of the approach is available as open source at https://gitlab.vgiscience.de/forsch/routing-preferences.

6.2 Related work

The algorithm presented in this chapter is based on compressing the trajectories into as few as possible subtrajectories. In this section, a review of literature related to trajectory segmentation and trajectory compression is presented. A summary of related work in bicycle routing and routing preferences is given in Chapter 5.

Trajectory segmentation. In trajectory segmentation, a trajectory is split into subtrajectories by time interval, shape, or semantic meaning (Zheng, 2015). Existing approaches work point-based, e.g. by identifying stay points (Li et al., 2008) or by extracting interesting places (Feuerhake et al., 2011), or segment-based by requiring certain properties to be similar on a segment. We will focus on the latter, which is closely related to the approach presented in this chapter. Buchin et al. (2011) present a framework to segment a trajectory into a minimum number of segments such that each segment fulfills a spatio-temporal criterion. For *monotone* criteria (i.e., criteria that, if they hold for a certain segment, they also hold for every subsegment), the authors present an efficient algorithm to compute the segmentation. Aronov et al. (2016) show that this problem is, in general, **NP**-hard when considering non-monotone criteria. Nonetheless, they show that for some non-monotone criteria, polynomial-time algorithms exist. Alewijnse et al. (2014) introduce a broader class of criteria, namely *stable* criteria, whose validity does not often change along the trajectory. For such criteria, they present an efficient segmentation algorithm. Trajectory segmentation is often combined with a classification of the segments. In a first step, Zhang et al. (2011) segment trajectories based on changes in travel mode, and in a second step, they classify the segments according to the travel mode. Similarly, Alewijnse et al. (2018) partition trajectories into classifiable subtrajectories. For the classification, they restrict themselves to spatio-temporal information extracted from the trajectories, whereas we want to derive semantic information from an underlying graph. Depending on the application, this graph yields information about the road network or its surroundings. Segmentation has also been considered for trajectories matched to a network, which we call *paths*. Knapen et al. (2016) segment a path into a sequence of shortest subpaths. They present a method to enumerate all minimal decompositions of a trajectory, which are decompositions comprising the fewest subpaths possible. The authors emphasize the utility of their approach in travel simulation for generating route choice sets. Knapen et al. (2020) developed the concept of minimum path decompositions focusing on the context of modeling routing behavior. Here, it is utilized to detect locations in the network that serve as preferred intermediate destinations. However, the authors maintain that geometric shortest paths are used as the underlying principle for decomposition. The algorithm presented in this chapter relaxes this constraint by finding the minimal decompositions based on a routing preference model. This approach yields not only the minimal decomposition but also the corresponding preference value, thus giving direct insight into the routing behavior of the user.

Trajectory compression. Existing methods in trajectory compression can be classified into two groups: geometric and network-based approaches. Geometric approaches handle trajectory data in its raw form, represented as a sequence of spatio-temporal points. For instance, Wang et al. (2014) propose *SharkDB*, a framework for in-memory trajectory storage. Other algorithms in this context include variations of the R-tree spatial datastructure (Guttman, 1984), which are optimized for trajectory data (Pfoser et al., 2000; Šaltenis et al., 2000; Tao et al., 2003).

However, in the context of this chapter, network-based storage and indexing techniques are most relevant. In this variant, the trajectories are conceptualized as paths in an underlying network, obtained by applying map-matching algorithms to the raw trajectories (see Chapter 4). Song et al. (2014a) present the *PRESS* framework, which compresses the trajectories' temporal and spatial components individually. For compressing the spatial component, they replace sections of the paths that are geometric shortest paths between their first and last vertex with these vertices, a process known as *shortest-path encoding* (Krogh et al., 2016). Similarly, Kellaris et al. (2013) adopt a comparable approach but utilize shortest paths based on the number of vertices along the path to achieve higher compression ratios. Another notable framework, *PATHFINDER* by Funke et al. (2019), utilizes Contraction Hierarchies (Geisberger et al., 2012) to compress trajectories by representing them as shortcut paths in said hierarchy. Remarkably, this method is capable of handling continent-sized networks efficiently.

In conclusion, the decomposition of trajectories into optimal subpaths plays an essential role in trajectory compression. Thus, our approach to learning the balance factor that minimizes the number of subpaths could also prove beneficial in this domain.

6.3 Methodology

Section 6.3.1 introduces this chapter's notation and theoretical background. Subsequently, the approach for inferring routing preferences from sparse sets of trajectories is presented. The approach consists of two major components. At first, Section 6.3.2 presents an algorithm that, given a path, returns the set of balance factors for which this path is optimal. Section 6.3.3 then discusses how this algorithm can be used to calculate the balance factor for which the path segmentation into optimal subpaths is minimal.

6.3.1 Preliminaries

In this section, the graph-theoretical background of the approach is introduced and the terminology used throughout the chapter is defined. Let $G = (V, E)$ be a directed weighted graph with the vertex set V and the edge set E. G represents the road network, with edges being road links and vertices being road intersections or dead ends. Two non-negative edge costs $c_0(e)$ and $c_1(e)$ are assigned to each edge $e \in E$, representing the effort needed to traverse the edge. These costs model the routing criteria. For example, they quantify the distance along an edge, the time needed to traverse the edge or its total ascent. The algorithms require $c_0(e)$ and $c_1(e)$ to be integer numbers. However, this is not a severe limitation as we can choose a fine discretization of continuous edge attributes.

We consider a bicriteria routing model that describes the trade-off between the two criteria c_0 and c_1. We combine these costs linearly using a *balance factor* $\alpha \in [0, 1]$. We obtain a *personalized* edge cost

$$c_\alpha(e) = (1 - \alpha) \cdot c_0(e) + \alpha \cdot c_1(e). \tag{6.1}$$

The factor α expresses the preference towards the two initial edge costs: a value close to zero implies a preference towards c_1 while a value close to one implies a preference towards c_0. Given the balance factor of a user, an optimal route can be found by applying a shortest-path algorithm on G using their personalized edge cost.

A *trajectory* is a sequence of location measurements reflecting the movement of a person or an object. To combine the information of the road network and a trajectory, the trajectory is matched to the road network graph G (for reference, see Chao et al. (2020); Jiang et al. (2022)). The matched trajectory is a sequence $P = (e_1, \ldots, e_n)$ of n, not necessarily distinct edges in G, which we refer to as *path*. Equation 6.1 generalizes to the personalized cost of a path P by

$$\begin{aligned} c_\alpha(P) &= \sum_{e \in P} c_\alpha(e) \\ &= (1 - \alpha) \cdot \sum_{e \in P} c_0(e) + \alpha \cdot \sum_{e \in P} c_1(e) \\ &= (1 - \alpha) \cdot c_0(P) + \alpha \cdot c_1(P) \\ &= c_0(P) + \alpha \cdot \big(c_1(P) - c_0(P)\big). \end{aligned} \tag{6.2}$$

We say that P is *α-optimal* for a given value α if there is no other path Q that starts and ends at the same vertices as P and whose personalized cost $c_\alpha(Q)$ is less than $c_\alpha(P)$. Following from Equation 6.2, for a path P, the personalized cost c_α can be treated as a function over the balance factor α, resulting in a line with vertical intercept $c_0(P)$ and slope $c_1(P) - c_0(P)$. Consequently, we can consider the set of all s-t paths, that is, all paths sharing the same terminal vertices s and t, as a family of lines, see Figure 6.2. The following refers to a family of s-t-lines as *line arrangement*. The paths that are α-optimal for some $\alpha \in [0, 1]$ form the *lower envelope* $\mathcal{E}\colon [0, 1] \to \mathbb{R}_{\geq 0}$ of the corresponding line arrangement, see the orange highlighting in Figure 6.2b. A path (and thus a line) on the lower envelope for a given α can be found by performing a query for a path from s to t in G that is optimal with respect to c_α.

The section of a line on the lower envelope forms the *optimality range* of the corresponding path. For this α-interval the path is α-optimal. As a line is linear and defined over the whole

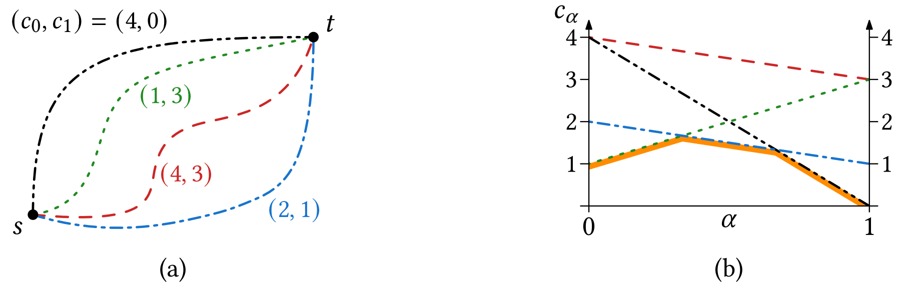


Figure 6.2: (a) A set of s-t paths with costs (c_0, c_1) and (b) the corresponding line arrangement; colors are chosen accordingly. In (b), the lower envelope $\mathcal{E}$ is highlighted in orange.

interval $[0, 1]$, every line has at most one section on the lower envelope. Thus, the optimality range is either empty, a single value, or an interval. In Figure 6.2, the optimality range of the red path is empty, and the optimality ranges of the green, blue, and black paths are $[0, 1/3]$, $[1/3, 2/3]$, and $[2/3, 1]$, respectively.

6.3.2 Calculating optimality ranges for paths

Given an s-t path P, we want to identify the α-interval for which P is optimal regarding the routing model introduced in Section 6.3.1. We call this problem OptimalityRange. More formally, we solve the following problem:

Problem 1 (OptimalityRange)**.** Given a graph G with edge cost functions c_0 and c_1, and a path P, find $\mathcal{I}_{\text{opt}} = \{\alpha \in [0, 1] \mid P \text{ is } \alpha\text{-optimal}\}$.

A known algorithm for finding elements on the lower envelope. Funke and Storandt (2013) have solved the problem of deciding whether a path P is part of the lower envelope. Their algorithm, called WitnessSearch, searches for a single α value for which P is optimal. If such a value can be found, P is part of the lower envelope; if no value can be found, P is not part of the lower envelope. We use WitnessSearch as the starting point for our algorithm. Due to the algorithm's importance for this chapter, we elaborate on their findings in more detail in the following. The algorithm of Funke and Storandt works iteratively. In every step, an interval $\mathcal{I}^*$ is considered that is guaranteed to contain the optimality range $\mathcal{I}_{\text{opt}}$ completely. From step to step, this interval $\mathcal{I}^*$ is reduced until a point α^* within the optimality range is found, or its size falls below the minimum size of an optimality range,[1] which implies that the optimality range is empty. If the optimality range is not empty, WitnessSearch returns a point α^* within the optimality range and the interval $\mathcal{I}^*$ as considered in the final iteration of the algorithm. In the other case, the path is not optimal for any α. Thus, the algorithm for finding the full optimality range can be omitted in this case. The running time of WitnessSearch is in $\mathcal{O}(\text{SPQ} \cdot \log(Mn))$, where SPQ is the running time of an optimal-path query in G, n is the number of vertices in G, and M is the maximum edge cost among all the edge costs c_0 and c_1.

[1] Mehlhorn and Ziegelmann (2000) have shown that the minimal slope difference between two lines is bound by $\mathcal{O}(n^{-2}M^{-2})$, where M is the maximum edge cost, resulting in a minimal bound on the size of an optimality range.

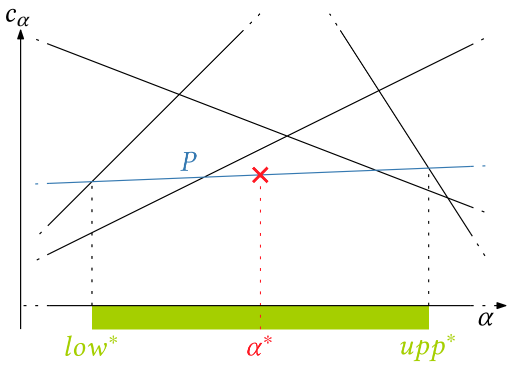


Figure 6.3: The output of WitnessSearch that we use as input for our algorithm: a balance factor α^* for which path P (blue) is α-optimal and the interval $\mathcal{I} = [low^*, upp^*]$ (green) that is guaranteed to contain the optimality range $\mathcal{I}_{\text{opt}}$ of P completely. In specific, low^* and upp^* are values where the line for P crosses another line of the line arrangement and $\alpha^* = \frac{low^*+upp^*}{2}$.

An extension yielding the optimality range. So far, only one element α^* within the optimality range $\mathcal{I}_{\text{opt}} = [\alpha', \alpha'']$ of P is known. We continue by searching for the boundaries of $\mathcal{I}_{\text{opt}}$. To that end, we use the results of WitnessSearch that are displayed in Figure 6.3. In particular, these are a balance factor α^* for which P is optimal with respect to c_{α^*} and the interval $\mathcal{I}^* = [low^*, upp^*]$ that is guaranteed to contain the optimality range $\mathcal{I}_{\text{opt}}$ of P completely. More specific, low^* and upp^* are values where the line for P crosses another line of the line arrangement and $\alpha^* = \frac{low^*+upp^*}{2}$.

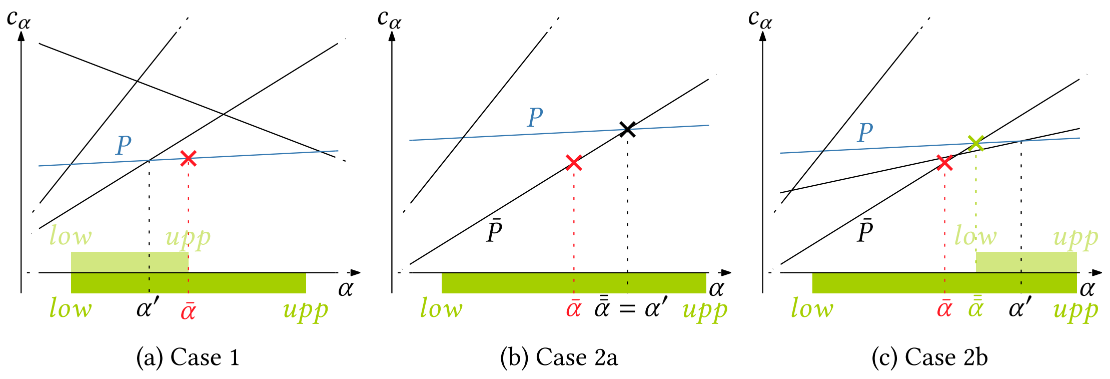


Figure 6.4: Search for the lower bound of the optimality range. The search interval is marked green (its current state is darker than the next). (a) If P is optimal with respect to $c_{\bar{\alpha}}$, the search for the lower bound continues in $[low, \bar{\alpha}]$. (b) If P is optimal with respect to $c_{\bar{\bar{\alpha}}}$, the lower bound of the optimality range is found, and the algorithm terminates. (c) At $\bar{\alpha}$, the path P is dominated by $\bar{P}$. The search continues in $[\bar{\bar{\alpha}}, upp]$ where $\bar{\bar{\alpha}}$ marks the intersection of P and $\bar{P}$.

We know that $\alpha', \alpha'' \in \mathcal{I}^* = [low^*, upp^*]$. In particular, $\alpha' \in [low^*, \alpha^*]$ and $\alpha'' \in [\alpha^*, upp^*]$ hold. In the following, we describe how to determine α' with a binary search that

Algorithm 2: FindOptimalRangeLowerBound

```
Data: path $P$, $\alpha^*$, $low^*$ as given by WitnessSearch (Funke and Storandt, 2013)
Result: lower bound $\alpha'$
upp ← $\alpha^*$;
while true do
    $\bar{\alpha} \leftarrow (low + upp)/2$ ;
    $\bar{P} \leftarrow \mathrm{argmin}_{P'}\, c_{\bar{\alpha}}(P')$ ; // optimal path wrt. $c_{\bar{\alpha}}$
    if $c_{\bar{\alpha}}(P) = c_{\bar{\alpha}}(\bar{P})$ then          // Case 1: P is optimal wrt. $c_{\bar{\alpha}}$ (Figure 6.4a)
        $upp \leftarrow \bar{\alpha}$;
    else                                                  // Case 2: P is not optimal wrt. $c_{\bar{\alpha}}$
        $\bar{\bar{\alpha}} \leftarrow \frac{c_0(\bar{P}) - c_0(P)}{c_1(P) - c_0(P) - c_1(\bar{P}) + c_0(\bar{P})}$ ; // crossing of $\bar{P}$ and $P$
        $\bar{\bar{P}} \leftarrow \mathrm{argmin}_{P'}\, c_{\bar{\bar{\alpha}}}(P')$ ; // optimal path wrt. $c_{\bar{\bar{\alpha}}}$
        if $c_{\bar{\bar{\alpha}}}(P) = c_{\bar{\bar{\alpha}}}(\bar{\bar{P}})$ then      // Case 2a: P is optimal wrt. $c_{\bar{\bar{\alpha}}}$ (Figure 6.4b)
            return $\bar{\bar{\alpha}}$;
        else                                              // Case 2b (Figure 6.4c)
            $low \leftarrow \bar{\bar{\alpha}}$;
```

uses the structure of the line arrangement. The search for α'' is organized symmetrically. This procedure is summarized in Algorithm 2 and visualized in Figure 6.4. In every iteration, a search interval $\mathcal{I} = [low, upp]$ is given and the following invariants hold:

(1) the lower bound α' is contained in $\mathcal{I}$,

(2) at value low, the line for P crosses another line of the line arrangement,

(3) P is optimal with respect to c_{upp}.

The interval $\mathcal{I}_1 = [low^*, \alpha^*]$ as given by WitnessSearch fulfills these properties and is used as input for the first iteration. We consider the central value $\bar{\alpha} = \frac{low+upp}{2}$ of $\mathcal{I}$. In each iteration, one of the following cases can occur; see Figure 6.4. Figure 6.4a displays Case 1. In this case, P is optimal with respect to $c_{\bar{\alpha}}$, and the next iteration continues with a search in $[low, \bar{\alpha}]$. Otherwise, in Case 2, there exists another path $\bar{P}$ that is optimal with respect to $c_{\bar{\alpha}}$. This path $\bar{P}$ has the same cost as P for some value $\bar{\bar{\alpha}}$ and dominates P in $[0, \bar{\bar{\alpha}}]$. Case 2 has two subcases depending on the α-optimality of P with respect to the crossing point $\bar{\bar{\alpha}}$. Figure 6.4b shows Case 2a, where P is an optimal path with respect to $c_{\bar{\bar{\alpha}}}$ and thus the lower bound of the optimality range is found, and the algorithm terminates. Otherwise, in Case 2b highlighted in Figure 6.4c, the search continues in $[\bar{\bar{\alpha}}, upp]$. In all cases, the invariants hold, and either the solution is found or the search interval $\mathcal{I}$ is reduced by at least half.

The lower bound, α', is found at the latest when the size of the search interval $\mathcal{I}$ falls below the minimum size of the optimality range. The search for α'' is done symmetrically. Hence, in the worst case, the search interval needs to be reduced from $[0, 1]$ to below the minimal size twice: once for the lower bound α', once for the upper bound α''. Consequently, the optimality range is found in $\mathcal{O}\big(\text{SPQ} \cdot \log(Mn)\big)$ time. Thus, the computational complexity of the presented algorithm to solve OptimalityRange is lower by a factor of $\mathcal{O}\big(\log(Mn)\big)$ compared to the version presented by Oehrlein et al. (2017).

6.3.3 Calculating minimal milestone segmentations

Given a path P and a balance factor α a *milestone segmentation* is a decomposition of P into the minimum number of subpaths $\{P_1, \dots, P_k\}$, such that every subpath P_i with $i \in \{1, \dots, k\}$ is α-optimal. We call the split vertices between the subpaths *milestones*. To infer the routing behavior of the user, we search for the balance factor $\alpha \in [0, 1]$ whose milestone segmentation has the minimum number of subpaths over all milestone segmentations of P.

Problem 2 (MilestoneSegmentation)**.** Given a graph G with edge cost functions c_0 and c_1, and a path P, find $\alpha \in [0, 1]$ and a milestone segmentation of P with respect to α that is as small as possible. That is, no $\tilde{\alpha}$ yields a milestone segmentation of P with a smaller number of subpaths than α.

To solve MilestoneSegmentation, we compute a set that contains a milestone segmentation of P for every α, then we select an optimal milestone segmentation. Enumerating the set of milestone segmentations is done by adopting a concept known as *start-stop matrix* (Alewijnse et al., 2014; Aronov et al., 2016). A start-stop matrix is a matrix $\mathcal{M}$, where entry $\mathcal{M}[i, j]$ is a Boolean value indicating whether the subpath starting at the i-th vertex of the path and ending at the j-th vertex of the path fulfills the segmentation criterion. In our case, α-optimality is the segmentation criterion. In MilestoneSegmentation, we do not solve the segmentation for a single α, but we search for the α value minimizing the number of subpaths over all $\alpha \in [0, 1]$. Therefore, instead of storing Boolean values in the start-stop matrix, we store the optimality interval for the corresponding subpath. This allows us to use the same matrix for all α values. Figure 6.5 shows the (Boolean) start-stop matrix for a single α, retrieved by applying the expression $\alpha \in \mathcal{M}[i, j]$ to all elements of the matrix. For a path P consisting of k vertices, we consider a $(k \times k)$-matrix $\mathcal{M}$ of subintervals of $[0, 1]$. The entry $\mathcal{M}[i, j]$ in row i and column j corresponds to the optimality range of the subpath of P starting at its i-th vertex and ending at its j-th vertex. Since we are interested only in subpaths with the same direction as P, we focus on the upper triangle matrix with $i \leq j$. Solving MilestoneSegmentation with a start-stop matrix splits the problem into filling the matrix and solving for an optimal segmentation.

Computing the start-stop matrix. Due to the optimal substructure of shortest paths (Cormen et al., 2009), for $i < k < j$, both $\mathcal{M}[i, j] \subseteq \mathcal{M}[i, k]$ and $\mathcal{M}[i, j] \subseteq \mathcal{M}[k, j]$ hold. This results in the structure visible in Figure 6.5, where no red cell is below or left of a green cell. Therefore, when computing the start-stop matrix, we start at the lower left corner and fill each row starting from its main diagonal. That way, we can limit the search space for $\mathcal{M}[i, j]$ to the intersection of the already computed values of $\mathcal{M}[i + 1, j]$ and $\mathcal{M}[i, j - 1]$. This is especially useful if one of these intervals is empty, meaning the current interval is also empty, and computation in this row can be stopped.

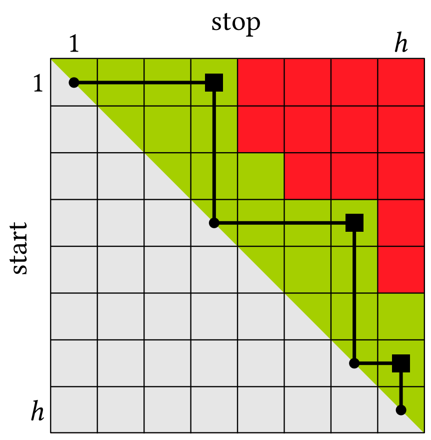


Figure 6.5: Depiction of a start-stop matrix of a path consisting of $h = 8$ vertices for a fixed $\alpha \in [0,1]$. Green elements indicate that $\alpha \notin \mathcal{M}[i,j]$ whereas red elements indicate the opposite. The black line represents a minimum path segmentation into three subpaths (v_1, v_4, v_7, v_8) resulting from a forward search.

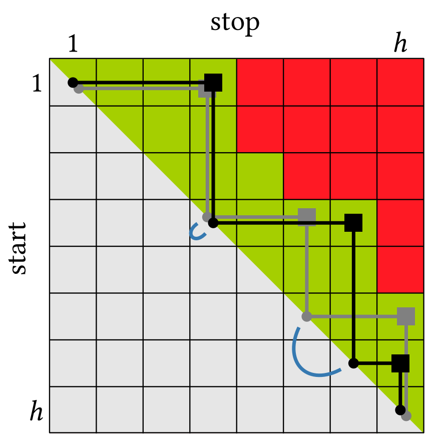


Figure 6.6: In addition to the segmentation of the forward search (black, see also Figure 6.5), the segmentation when searching backward (gray) is displayed. The blue lines indicate the milestone suites. Note that the first milestone suite degenerated into a single vertex.

Solving for an optimal segmentation. Once the start-stop matrix is set up, the abovementioned substructure optimality makes it easy to find a solution to the segmentation problem for a fixed α. According to Buchin et al. (2011), for example, an exact solution to this problem can be found with a greedy approach in $O(h)$ time; see Figure 6.5. Knapen et al. (2016) have shown that applying the greedy approach twice, once starting from the beginning vertex and once from the end vertex of P, results in pairs of milestones that form the terminal vertices of a *milestone suite*. The milestone can be chosen arbitrarily inside a milestone suite without violating the segmentation into optimal subpaths. Note, however, that choosing a milestone in a specific milestone suite might fix the milestone location in other milestone suites. The size of a milestone suite is the distance along the path between its terminal vertices. In Figure 6.6, the forward and backward searches are indicated by a black and gray line, respectively, resulting in one milestone suite of size zero and one milestone suite of size one (blue). If multiple segmentations have the same number of subpaths, we use their summed milestone suite sizes as a tiebreaker. To have the highest adaptability of the learned balance factor, we consider larger milestone suite sizes better than smaller ones.

Since we consider a finite set of intervals, we know that if a minimal solution exists for an $\alpha \in [0,1]$, it also exists for one of the values bounding the intervals in $\mathcal{M}$. Consequently, we can discretize our search space without loss of exactness, and each of the $O(h^2)$ optimality ranges yields at most two candidates for the solution. For each candidate, a minimum segmentation is computed in $O(h)$ time. Thus, we end up with a total running time of $O\big(h^2 \cdot (h + \mathrm{SPQ}\log(Mn)\big)$ where n denotes the number of vertices in the graph and h denotes the number of vertices in the considered path. Thus, the algorithm is efficient and yields an exact solution to MilestoneSegmentation. The solution consists of the intervals of the balance factor producing the best personalized costs with respect to the input criteria.

Guaranteeing the existence of a milestone segmentation. Finally, there may be no milestone segmentation for a general graph $G = (V, E)$ and a path P in G. Consider, for example, the case in Figure 6.7a, in which P consists of a single edge $e = \{s, t\}$ whose costs $c_0(e)$ and $c_1(e)$ are larger than the costs $c_0(Q)$ and $c_1(Q)$, respectively. Since P is dominated by Q for any $\alpha \in [0, 1]$ and since we cannot subdivide P into shorter paths in G, the path P does not admit a partition into α-optimal subpaths.

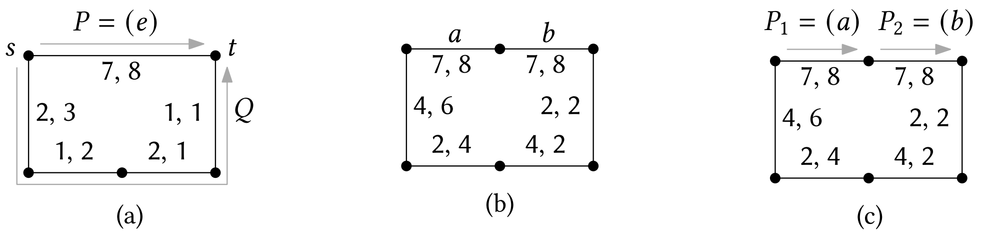


Figure 6.7: (a) A graph with a path $P = (e)$ that does not admit a milestone segmentation and (b) the result of our method that ensures the feasibility of the problem. Comma-separated numbers represent c_0 and c_1. (c) Milestone decomposition after applying our method. Path P is split into two subpaths, with the newly introduced vertex being the milestone.

However, we can ensure a solution exists with a simple pre-processing step. For this, we introduce a new vertex w for each edge $\{u, v\} \in P$ and replace that edge with two edges $a = \{u, w\}$ and $b = \{w, v\}$, both in G and P. We define $c_0(a) = c_0(b) = c_0(e)$ as well as $c_1(a) = c_1(b) = c_1(e)$ and double the costs of each unchanged edge in G; see Figure 6.7b. Thereby, we ensure that the costs c_0 and c_1 are still integer. The costs of paths in the original graph are exactly doubled. Thus, if they were α-optimal in the original graph, they remain α-optimal after the modification. After the pre-processing step, however, every edge $\{u, v\}$ of P is α-optimal for any $\alpha \in [0, 1]$. Therefore, a milestone segmentation exists, and an optimal milestone segmentation of P contains at most $|P|$ paths — or $2|P|$ paths if we consider the size $|P|$ of P in the original graph. Note that P in its entirety is still not α-optimal for any $\alpha \in [0, 1]$. We deliberately do not want to change the optimality of paths in the original graph. Instead, P is now guaranteed to have a milestone segmentation using the newly introduced vertices as milestones; see Figure 6.7c. The asymptotic running time of our algorithms is not affected by our method to guarantee the existence of a milestone segmentation.

6.4 Evaluation

This section presents the experimental evaluation of the algorithm. The experiments are set up to examine how much cyclists stick to official bicycle routes. For this, the following edge costs are defined:

$$
\begin{aligned}
c_0(e) &= \begin{cases} 0, & \text{if } e \text{ is part of an official bicycle route} \\ d(e), & \text{else.} \end{cases} \\
c_1(e) &= \begin{cases} d(e), & \text{if } e \text{ is part of an official bicycle route} \\ 0, & \text{else.} \end{cases}
\end{aligned} \tag{6.3}
$$

Hence, in our cost function, we use edge costs c_0 and c_1 that, apart from an edge's geodesic length d, depend only on whether the edge in question is part of an official bicycle route. Since c_0 and c_1 are discrete cost functions, the costs of each edge are rounded to the accuracy of one decimeter. With respect to the personalized cost $c_\alpha = (1-\alpha) \cdot c_0 + \alpha \cdot c_1$, the possible outcome can be interpreted as follows. Suppose there is a path P_1 entirely using official bicycle routes. Its cost is: $c_\alpha(P_1) = \alpha \cdot c_1(P_1) = \alpha \cdot d(P_1)$. Further, let P_2 be a path entirely using routes *not* part of official bicycle routes. Its cost is $c_\alpha(P_2) = (1-\alpha) \cdot c_0(P_2) = (1-\alpha) \cdot d(P_2)$. If both paths have the same cost given a certain value for α, the following equation holds:

$$\alpha \cdot d(P_1) = (1-\alpha) \cdot d(P_2) \tag{6.4}$$

A minimal milestone segmentation for $\alpha = 0.5$ suggests that minimizing geodesic length outweighs the interest in official cycle paths. A value $\alpha < 0.5$ indicates avoidance of official cycle paths, whereas $\alpha > 0.5$ indicates the opposite, i.e., preference. Specifically, if we are interested in the relative detour δ_{PRO} a cyclist is willing to take in order to stick to official bicycle routes, we can transform Equation 6.4:

$$\delta_{\text{PRO}} = \frac{d(P_2)}{d(P_1)} = \frac{\alpha}{(1-\alpha)} \tag{6.5}$$

In contrast, the detour δ_{CON} a cyclist is willing to take in order to avoid official bicycle routes can be computed using the inverse of Equation 6.5. For our analysis, we will use the following combined detour function, which Table 6.1 provides some exemplary values for:

$$\delta(\alpha) = \begin{cases} \delta_{\text{CON}} - 1, & \text{if } \alpha < 0.5 \\ 0, & \text{if } \alpha = 0.5 \\ \delta_{\text{PRO}} - 1, & \text{if } \alpha > 0.5 \end{cases} \tag{6.6}$$

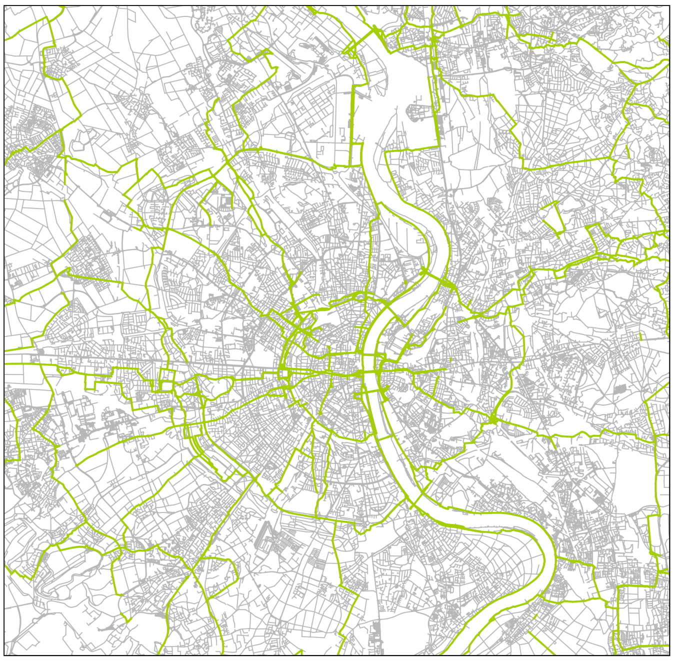
Table 6.1: A selection of α values and their meaning. The column $\alpha = 0.5 \pm \mathbf{0.1}$ indicates that with $\alpha = 0.4$ (or 0.6), the user is willing to cover 50% extra distance to use preferred roads.

$\alpha = 0.5 \pm x$	0	0.0025	0.01	0.1	1/6	0.25	1/3	0.4	0.45
detour δ	0	1%	4%	50%	100%	200%	400%	800%	1800%

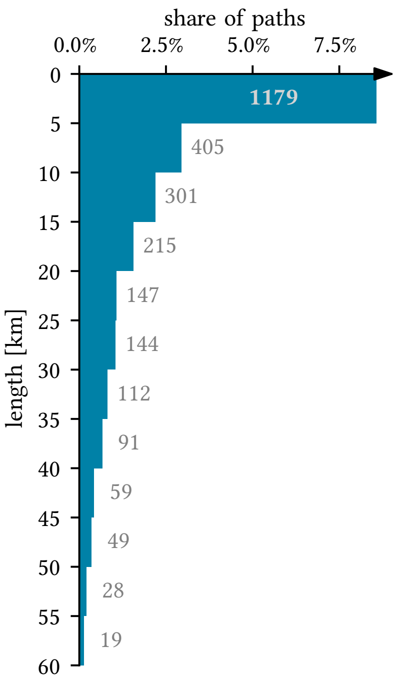
In the experiments, we limit ourselves to $\alpha \in [0.05, 0.95]$ to avoid $\alpha \in \{0, 1\}$. In these two cases, for every road segment e, either $c_0(e) = 0$ or $c_1(e) = 0$ holds. In such a graph, for any s-t path, there are numerous alternatives of the same cost, which only differ in additional segments of cost zero. Computing shortest paths is time-consuming in this graph since many vertices must be explored. Besides, considering a detour of up to 1800% seems sufficient.

6.4.1 Data and experimental setup

This section describes the setup in more detail. The road network, including information about bicycle routes, is obtained from OpenStreetMap[2] (OSM) and comprises the area $5.99° - 7.67°$ E and $50.47° - 51.47°$ N. It contains only roads that are, according to OSM tags, open to cyclists.



(a) road network



(b) path lengths

Figure 6.8: (a) The road network of the city of Cologne, the center of the area containing the examined trajectories. Only roads and paths that are accessible for cyclists are on display. Roads and paths that are part of an official signposted cycle route are drawn as green lines. (b) Geometric length of the matched paths.

Figure 6.8a shows an extract of this road network data. We transform the road network into a directed graph $G = (V, E)$, such that for each edge $(u, v) \in E$ also the reverse edge (v, u) is contained in E. This means that we do not consider one-way restrictions in the experiments, though generally, the algorithm can handle them. The two cost functions $c_0, c_1 \colon E \longrightarrow \mathbb{N}$ are defined as in Equation 6.3.

[2] www.openstreetmap.org

In order to reduce the complexity of the graph, we remove vertices with degree two from G. More precisely, when removing a vertex u with two outgoing edges (u,v) and (u,w) as well as the corresponding incoming edges (v,u) and (w,u), we replace the four edges with (v,w) and (w,v). The costs of (v,w) are the summed costs of (v,u) and (u,w); the costs of (w,v) are the summed costs of (w,u) and (u,v). We denote the resulting graph by H. That way, the graph could be reduced from 3.5 million vertices and 7.8 million edges to 0.9 million vertices and 2.5 million edges.

The trajectory data used in the experiments stems from the user-driven platform GPSies[3]. Different groups of cyclists use this platform, which features a wide range of cycling activities, from scenic bike touring to mountain biking to road biking. Thus, the analysis is expected to find different routing preferences. In total, 1016 trajectories are obtained by querying trajectories recorded by cyclists around the German city of Cologne in a search radius of 25 km. The trajectories have a length up to 60 km.

To analyze the trajectories with our algorithm, they must be converted to paths in G. We are working with data from VGI sources; therefore, we need a map matching that is robust to possible data inconsistencies. To that end, the map-matching algorithm Baseline+ as presented in Chapter 4 is used. This algorithm is designed explicitly for incomplete road networks or trajectories leaving the road network. If such an inconsistency is detected, the corresponding trajectory is split into multiple parts, and each part is matched to an individual path. The map-matching algorithm has two parameters: the candidate and unmatched costs. We set the parameter candidate cost to 0.02 and the parameter unmatched cost to 3.0. This parameter choice is recommended by the authors and performed well on our input data, as validated by a visual examination of the results. We filtered the resulting paths with a minimum length of 100 meters. We end up with a total number of 2750 paths with a mean distance of 12.3 kilometers (see Figure 6.8b).

In case a start or end point of a path P was mapped to a point p in the interior of an edge e of G, we temporarily injected p as a new vertex of G by splitting e into two edges. We obtain P as a sequence of edges of G. We linearly distributed the costs of the edge e over the two new edges. In the reduced graph H, we temporarily insert the sequence of degree-two vertices of P containing additional vertices (like p) as well as the connecting edges, which allows us to consider P as a sequence of edges in H. We further apply the pre-processing step described in Section 6.3.3 to ensure that the path has a milestone segmentation. All changes on H are undone before the next path is considered.

The experiments were performed on an AMD Ryzen Threadripper 3970X 32-core processor. The machine is clocked at 3.7 GHz and provides our program with 13 GB RAM. The implementation is written as a single-threaded Java application, and the experiments were executed in a Java 11 Runtime Environment. The implementation uses two open-source libraries: GeoTools (Turton, 2008) for handling geographic data and JGraphT (Michail et al., 2020) for the graph structures and algorithms.

[3] www.gpsies.com, [no longer availiable], downloaded: May 2018

6.4.2 Case study on a single trajectory

Before analyzing the examined 2750 paths, the results for a single path of 10.0 km length are scrutinized. Figure 6.9 shows the trajectory as well as an optimal solution of MilestoneSegmentation.

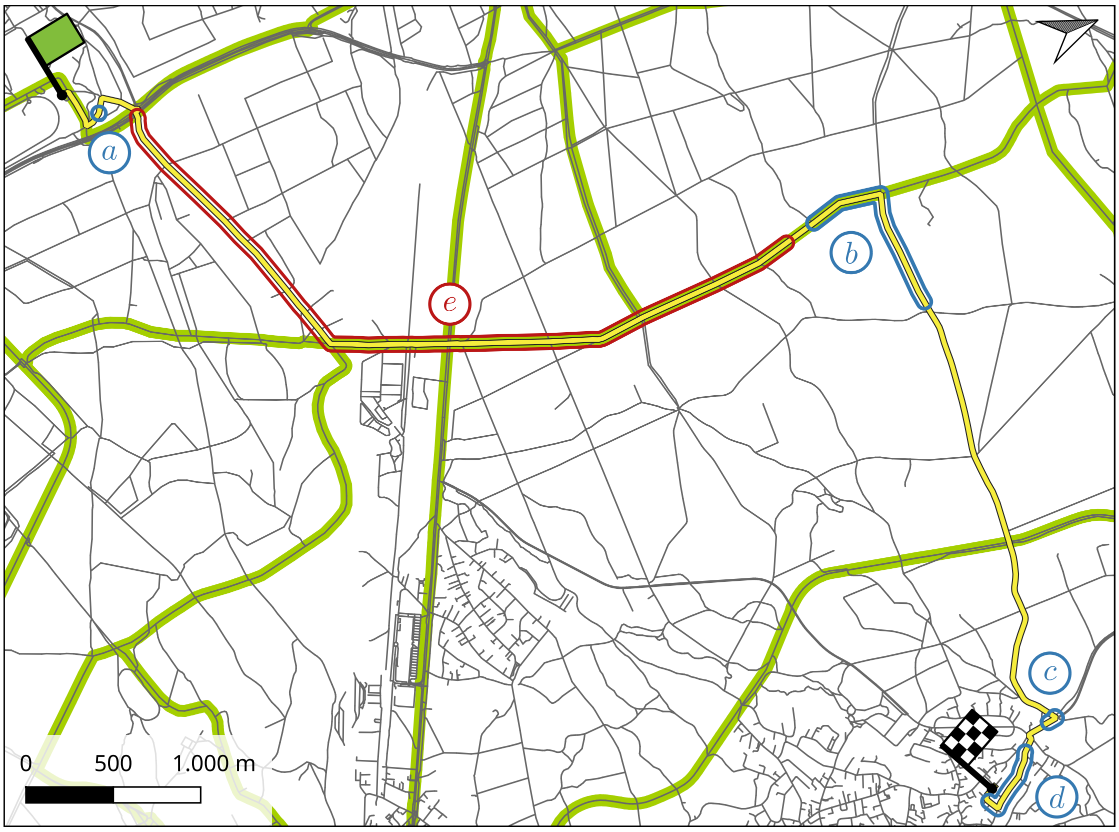


Figure 6.9: The user's path (yellow) has an optimal milestone segmentation with four milestones: a, b, c, and d (blue). Disregarding minor shiftings of a and/or b, this is a valid optimal segmentation for each $\alpha \in [0.511, 0.607]$ (compare with Figure 6.10). The subpath (a, b) is α-optimal for $0.511 \leq \alpha \leq 0.607$ only. Therefore, for $\alpha = 0.5$, i.e., the geodesic shortest path, an additional milestone e (red) is needed.

In this example, we found an optimal segmentation by assuming that cyclists are willing to take a detour between $\delta = 4\%$ and $\delta = 54\%$ over the length of the shortest path, corresponding to $\alpha \in [0.511, 0.607]$. The milestone suites for this interval are shown in blue. When assuming that cyclists are sticking to the geodesic shortest path, an additional milestone must be placed inside milestone suite e depicted in red. Figure 6.10 (right) depicts every segmentation our algorithm finds for this trajectory. The two segmentations shown in Figure 6.9 are highlighted by a blue and red line, respectively. In dark gray, the milestones found by the forward search are shown with the corresponding milestone suite depicted in light gray. We observe that the

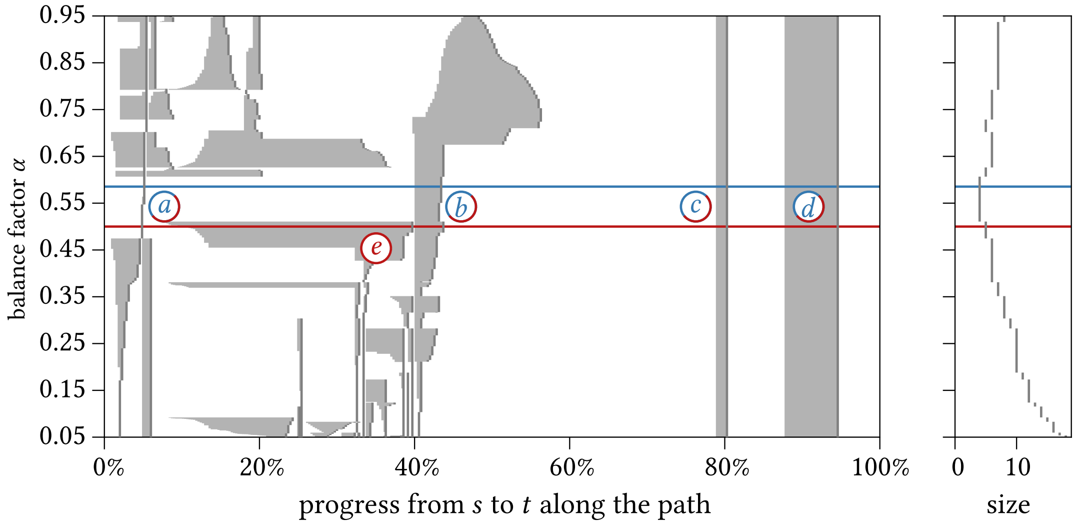


Figure 6.10: (left) Milestone plot for the path presented in Figure 6.9. The milestone segmentations over all α values are shown. The gray marks indicate the milestone positions as found by a forward search. The light gray areas denote the milestone suites for the corresponding milestones. The blue line represents an optimal milestone segmentation at $\alpha = 0.58$ with milestones in a, b, c, and d. For $\alpha = 0.5$ (red), an additional milestone in e is needed. (right) The number of milestones needed per α.

sizes of the individual milestone suites grow towards the optimal interval $\alpha \in [0.511, 0.607]$. Specifically, look at milestone suite e: where it first appears at $\alpha = 0.511$, it spans a large portion of the path, while it quickly shrinks until it is replaced by two separate milestones at $\alpha \approx 0.36$. This observation supports our initial assumption that larger milestone suites are more adaptive and, thus, less prone to overfitting. The milestone plot shown in Figure 6.10 is suitable for further analysis of the trajectory and its surrounding road network as it shows for every α value where milestones are necessary. Since this chapter focuses on analyzing user preferences, research on this matter is postponed. Figure 6.10 (left) summarizes the milestone plot by showing how the size of the shortest milestone segmentation depends on α.

6.4.3 Analyzing all trajectories

First, we use the algorithm to divide the set of trajectories into three groups: Pro, Con, and Indiff. The group Pro comprises trajectories for which a milestone segmentation of minimal size can only be found for $\alpha > \frac{1}{2}$. We interpret such a result that way that the trajectory was planned in favor of official cycle routes. Conversely, we consider it as avoidance of official cycle routes if milestone segmentations of minimal size exist only for $\alpha < \frac{1}{2}$. The group Con represents these trajectories. Considering trajectories lacking this clarity, we refrain from a strict categorization into one of the two groups above. These trajectories form a third group, Indiff. The results of this classification are displayed in Figure 6.11. The group Pro being over four times larger than the group Con is a first indicator that cyclists prefer official cycle routes over other roads and paths. The group Con is the smallest group with a share of 8% of all paths. We assume this group mainly consists of road cyclists who prefer using roads over cycleways. This could be verified in future research by analyzing a data set of annotated trajectories.

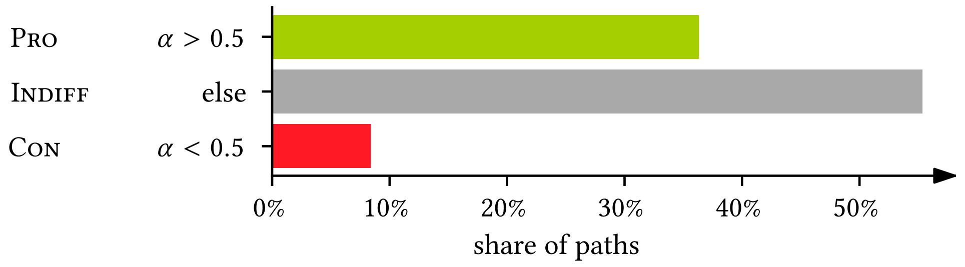


Figure 6.11: Size of the categories Pro (green, 998 paths), Con (red, 230 paths), and Indiff (gray, 1522 paths)

For a more detailed evaluation of the results, we introduce the *deficiency* Δ of a segmentation $\mathcal{S}$ to compare segmentations of different paths to each other. For a path P, the deficiency $\Delta(\alpha)$ that compares the number of subpaths in $\mathcal{S}(\alpha)$ to the number of subpaths in an optimal segmentation of P, $|\mathcal{S}_{\text{opt}}|$:

$$\Delta(\alpha) = \frac{|\mathcal{S}(\alpha)|}{|\mathcal{S}_{\text{opt}}|} - 1. \tag{6.7}$$

Phrased differently, the deficiency describes the share of additional subpaths needed for a given α value. Thus, it can be considered a percentage. Following this definition, the deficiency of an optimal segmentation of P is 0%, and the worse an α value fits the user's preference, the more the deficiency increases.

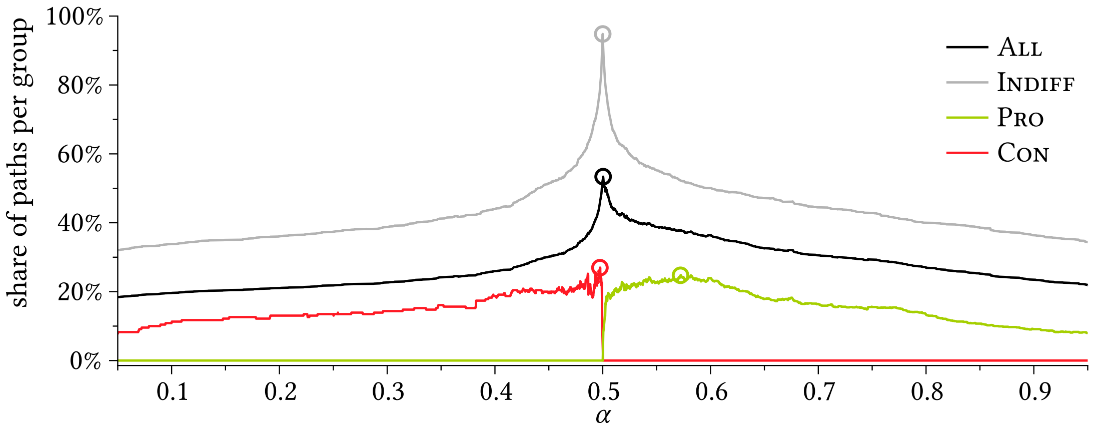


Figure 6.12: Share of paths per group that are optimal for the given α. The maxima are marked with a circle and are achieved for $\alpha \approx 0.4974$ (Con, red), $\alpha \approx 0.5721$ (Pro, green), $\alpha = \frac{1}{2}$ (Indiff, gray), and $\alpha = 0.5004$ (all trajectories, black).

Figure 6.12 displays for every α the share of the evaluated paths that are optimal with respect to the corresponding combined cost function c_α. Put differently, the figure shows the share of paths with zero deficiency for the given α. Considering the group Pro (green), the maximum share is reached with $\alpha \approx 0.5721$ as the balance factor. This corresponds to detours of more than 30% and emphasizes the preference for official cycle routes. The second group, Indiff (gray), reaches its optimum for $\alpha = 0.5$, i.e., for minimizing route length. Indeed, more than 94% of this group's paths have an optimum milestone segmentation for this α value. This causes a significant shift in the overall results (black). The balance factor that has, over all groups, the highest share of optimal paths is $\alpha \approx 0.5004$, corresponding to detours of about 0.2%. Although this value is relatively small, the relevance of official cycle paths is undeniable. Figure 6.13 supports this claim. It displays for every α the average deficiency of the evaluated paths. A segmentation for $\alpha \approx 0.5004$ takes, on average, only about 14% more milestones than the respective minimum segmentation. Also, on average, less than 50% milestones extra compared to the minimum segmentation is achieved for $\alpha \in [0.4824, 0.5980]$ corresponding to detours of less than 8% (avoiding cycle routes) and 50% (favoring cycle routes), respectively. This shows how much better the trajectories can be described under the assumption that official cycle routes are preferred. Considering an accepted detour of 50% likewise to avoid official cycle routes (i.e., $\alpha = 0.4$), one needs, on average, more than 2.5 times the number of milestones compared to the minimum segmentation, see the red line in Figure 6.13.

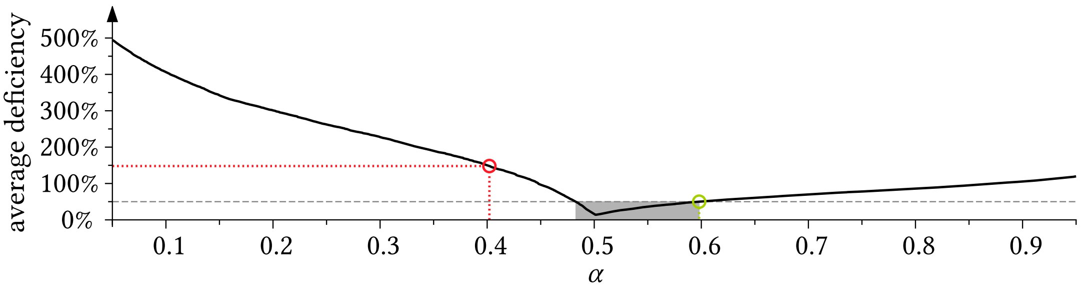
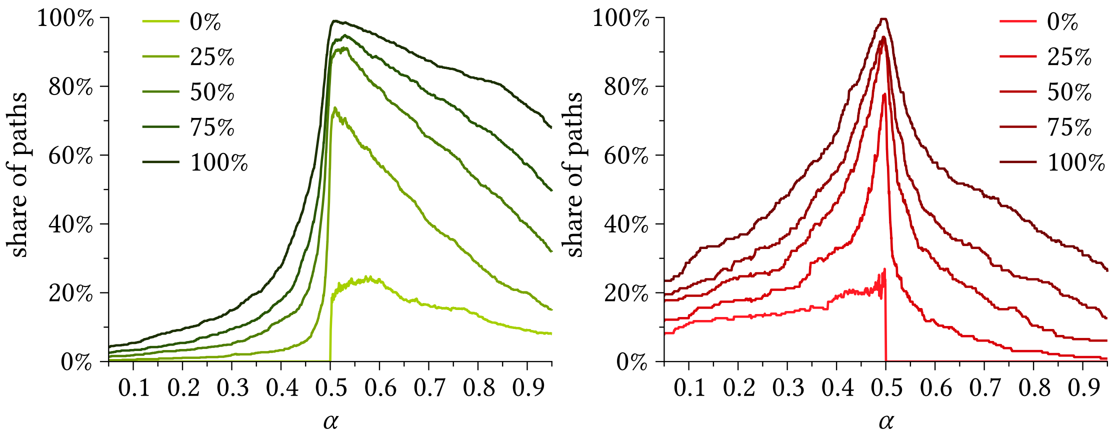


Figure 6.13: For every α value, the average deficiency over all evaluated paths is displayed. For the interval $\alpha \in [0.4824, 0.5980]$ (gray), the average deficiency is lower than 50% (indicated by the dashed line). For $\alpha = 0.4120$ (red), the symmetric value to $\alpha = 0.5980$ (green), assuming an avoidance of official cycle routes, the deficiency is three times as high.

Even if cyclists want to stick to official cycle routes, they might not take the optimal path because the difference between the optimal path and the path taken is too small to be noticeable. Still, in our mathematical model, a milestone needs to be inserted. Therefore, instead of only looking at an optimal segmentations, we can evaluate for which α values a path can be segmented with a given maximum deficiency. Not requiring zero deficiency allows for introducing some inaccuracies into the model. Specifically, for a path P, we call a balance factor α *acceptable* if $\Delta(\alpha) \leq \Delta^*$ for a given maximum deficiency Δ^*. Figure 6.14 shows the share of trajectories in each group acceptable for maximum deficiencies of $\{0\%, 25\%, 50\%, 75\%, 100\%\}$. Following the definition of the groups, for $\Delta^* = 0\%$, group Pro contains no acceptable paths for $\alpha < 0.5$, and group Con has no acceptable paths for $\alpha > 0.5$. When the maximum allowed deficiency increases, the share of acceptable paths naturally increases for a given α. Note, however, how the shapes of the plotted lines change: while the lines for group Pro are very asymmetric with a steep increase at around $\alpha = 0.5$, the lines for group Con get more or less symmetric around $\alpha = 0.5$. In fact, for $\Delta^* = 100\%$, the share of acceptable paths for $\alpha > 0.5$ is larger than the share of acceptable paths for $\alpha < 0.5$. While cyclists of this group seem to avoid official cycling routes, the earlier-mentioned observations reveal that this is only a minor repulsion.



(a) group Pro (b) group Con

Figure 6.14: Share of trajectories per group acceptable for a given maximum deficiency Δ^*. Each line per graph shows one maximum deficiency value between 0% and 100%. Note that the lines for 0% are already displayed in Figure 6.12.

Next to the inference of routing preferences, our approach can be used to compress trajectory data. Assuming no equal-cost s-t-paths are in the graph G, a path P in G can be fully reconstructed given s, t, α, and the corresponding milestones. The paths in our data set consist of 172614 vertices in G. Using our approach, this data can be expressed by 2357 vertices, resulting in a compression factor of 98.63%.

Finally, we take a look at the running times. For each of the 2750 paths, we measured the running time of our algorithm. The median running time of our algorithm per path was 1.40 minutes, and the maximum running time was 4165 minutes. Of the 2750 runs, 984 runs needed more than five minutes, while 1307 required less than one minute. We further observe that the shortest path queries dominate the running time. More precisely, the shortest path queries consume 60% of the running time on average. Hence, the shortest path queries offer considerable potential for further improvement in the running times. Until now, we are using a bidirectional variant of Dijkstra's algorithm (Dijkstra, 1959) as provided by JGraphT. In previous years, speed-up techniques such as contraction hierarchies have been optimized for multi-dimensional routing criteria. Using the algorithm presented by Funke et al. (2017) would result in considerable speed-ups for our approach.

6.5 Conclusion

This chapter presented and elaborated an idea for analyzing user preferences in bicriteria routing models. To this end, the problems OptimalityRange and MilestoneSegmentation were formalized. Assuming integral edge costs in the road graph, algorithms that yield exact solutions for both problems were developed and presented. Both algorithms are exact and efficient with an overall running time in $O\big(h^2 \cdot (h + \mathrm{SPQ} \cdot \log n)\big)$ where n denotes the number of vertices in the graph, h denotes the number of vertices in the considered path and SPQ denotes the time needed for one optimal-path query. The benefits of our approach are demonstrated by the experimental results obtained from a prototypical implementation. A major difference of the presented approach compared to the work by Barth et al. (2020) is that the presented approach segments a user's path into optimal subpaths of the same underlying cost function. On the one hand, this improves the approach's utility by providing an explicit balance factor for future routing. On the other hand, this also reduces the risk of overfitting the model.

In the experiments, the extent to which cyclists prefer official cycle routes over the shortest paths was analyzed. To that end, the balance factors that yield minimal milestone segmentations were determined. Out of the 2750 processed trajectories, only 483 were α-optimal paths in the road network. On the one hand, this leads to the conclusion that the criteria considered —geodesic length and usage of official cycle routes— do not suffice to fully explain how cyclists choose their routes. On the other hand, we could segment each of the given trajectories into a small number of α-optimal paths. Thus, we can conclude that the bicriterial model yields reasonable results, especially if we take its simplicity into account. The high number of trajectories having an optimal segmentation for α values close to 0.5 affirms the importance of minimizing the geodesic distance when planning a route. However, it is only a little more than 35% of all trajectories with a minimal segmentation for exactly 0.5. In particular, considering all trajectories, the least number of necessary milestones is achieved for $\alpha \approx 0.5013$ (see Figure 6.13), which indicates that the average cyclist in our experiment prefers official cycle paths. Furthermore, more than 36% of all trajectories have minimal milestone segmentations only if we assume that official cycle routes are preferred, i.e., if $\alpha > 0.5$. That means a large group of cyclists accept actual detours to use cycle routes. Here, the highest number of minimum milestone segmentations can be achieved assuming that cyclists are willing to cover 50% distance extra, i.e., $\alpha \approx 0.5721$; see Figure 6.14.

Although the developed algorithm is efficient, the observed running times are too long to be suitable for large-scale applications. On the one hand, running times can probably be substantially reduced by using specialized shortest-path algorithms, such as those presented by Funke and Storandt (2013) and Funke et al. (2017). On the other hand, future work could relax the exactness of our algorithm to speed up the computation. There are several ways one could achieve this. Firstly, a deficiency could already be introduced in the computation of the optimality ranges. Instead of strictly requiring a subpath to be optimal, small losses in the cost function could be accepted. Secondly, instead of computing the milestone segmentation continuously for $\alpha \in [0, 1]$, the interval could be sampled. Last but not least, the method for guaranteeing the existence of a milestone segmentation increases the number of vertices per path by a factor of two and, thus, the size of the interval matrix by a factor of four. The definition of a milestone segmentation could be relaxed to a segmentation of *basic path components* (i.e., optimal subpaths or single, non-optimal edges) (Knapen et al., 2016). While this might lead to additional milestones compared to our current approach, introducing the additional vertices is unnecessary. This has the added benefit that the graph remains unchanged for different paths, benefiting the compression of the data.

The algorithm's results yield much more information than this chapter has used and analyzed. Besides testing sets of trajectories on the relevance of specific criteria, we think of a different application with a more critical role for individual trajectories. Figure 6.10 shows that the milestones are not distributed equally along the path. In particular, there are accumulations of milestones for various α values. This data promises information about the surrounding road network, which may be interesting for traffic planners as it indicates where improvements of the considered network are reasonable or necessary.

An interesting open question is whether the discussed problems provide a basis for efficient algorithms if more than two criteria are considered. Using our algorithm, one approach for this multi-criteria problem is first to merge two criteria into a combined criterion. Then, this combined criterion can be merged with an additional criterion. This process can be iterated for an arbitrary amount of criteria. Initial tests of this method using six different criteria show promising results. Nonetheless, there are open questions regarding this approach. For example, the influence of the merging order on the final result needs to be analyzed. This analysis will be part of future research.

Finally, we note that users of online route planners such as Google Maps[4] frequently plan their routes by (first) computing a route between a start and an end vertex and (second) adding intermediate vertices to tailor the route according to their preferences. We think our algorithm is beneficial in this route-planning scenario since by using the α-value that yielded the best milestone segmentation for existing trajectories; the user will probably be able to plan new routes with few intermediate vertices. To verify this assumption, however, a further evaluation of our method is necessary, particularly in interactive route planning.

[4] www.google.com/maps

Part III

Geovisualization

Chapter 7

Geovisualization: Motivation & Research Context

Geovisualization provides methods for visually exploring, analyzing, synthesizing, and presenting geographic data (MacEachren and Kraak, 2001). The research discipline emerged in the 1990s when the development of digital displays allowed for a shift from static maps to dynamic visualizations (Çöltekin et al., 2018). Geovisualization draws upon approaches from cartography, scientific visualization, information visualization, and exploratory data analysis (Dykes et al., 2005). The boundaries between the research branches are fuzzy. Nonetheless, the concept of geovisualization often involves map interaction (Roth, 2013). An important early framework for geovisualization is "*Cartography*^{3}" by MacEachren (1994). In this framework, map use is defined in a three-dimensional cube with the following dimensions, see Figure 7.1:

1. **user**: is the map generated and used by individuals (private) or made for public use?
2. **task**: is the visualization's goal to explore unknowns or to communicate existing knowledge?
3. **interaction**: can the user interact with the map or the displayed data, e.g., by switching layers, or is the visualization static?

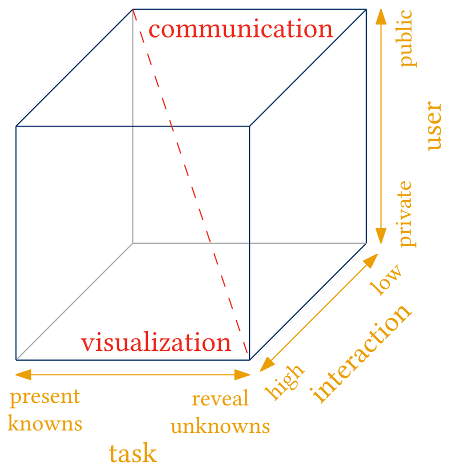


Figure 7.1: The Cartography^3 cube as introduced by MacEachren (1994). According to the model, (geo)visualizations are characterized by extensive human-map interaction, primarily intended for individual use. The objective of such visualizations is to unveil hidden patterns within the data. As such, they are contrasting cartographic communication, primarily aimed at conveying information to a broader audience.

In this cube, MacEachren locates geovisualization in the corner, representing exploratory, private map use with high interaction, and contrasts it to cartographic communication, which is generally geared to communicate information to the public.

In the context of this thesis, not only the explorative nature of geovisualization but also the communication aspect is of high relevance. In particular, in our use case, where we learn complex user preferences based on an abstract mathematical model, geovisualization allows the users to understand the provided information easily. Especially in the context of VGI, this can support user engagement: If a user sees how their provided data is used, it can motivate them to contribute even more data (Composto et al., 2016; Gómez-Barrón et al., 2016; Gómez-Barrón et al., 2019). The following takes a closer look at related work in two geovisualization fields closely related to this thesis's context: visualizing movement data and visualizing routing choices.

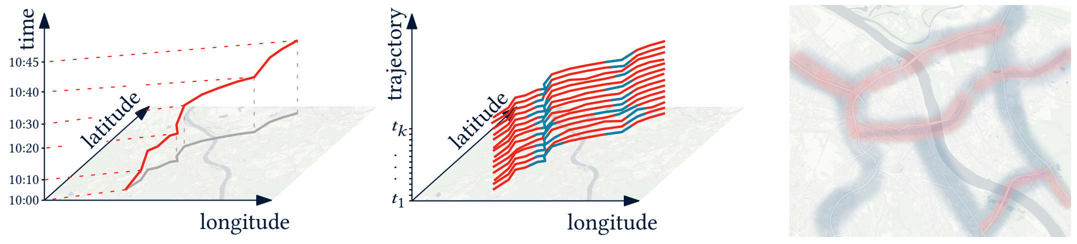
7.1 Visual exploration of movement data

In the context of geospatial data, visual exploration plays a crucial role, particularly in analyzing movement data. In the research field of statistics, Tukey (1977) framed the term *exploratory data analysis* to describe methods aimed at visually analyzing data. These methods employ visualizations to uncover patterns within complex, multidimensional datasets, enabling the viewer to easily extract insights from the data. The subsequent section provides an overview of methods utilized to analyze movement data, followed by a more comprehensive examination of approaches focusing on the visualization of trajectory data.

Movement data analysis. In movement-data analysis, datasets of trajectories, paths, or flows of objects or phenomena are examined over space and time (Demšar et al., 2021). Geovisualization techniques offer powerful tools for exploring and understanding such data. By employing interactive maps, time-series visualizations, and animation, analysts can gain valuable insights into movement patterns, trends, and anomalies (Demšar et al., 2015; Dodge and Noi, 2021). For instance, in transportation planning, geovisualization supports understanding traffic flows, identifying congestion hotspots, and optimizing route planning (Bachechi et al., 2022; Scheepens et al., 2016). In wildlife ecology, it facilitates tracking animal movements, studying migration patterns, and assessing habitat suitability (Demšar et al., 2015; Zhang, 2019; Sun et al., 2021; de Rivera et al., 2022). Moreover, in epidemiology, geovisualization assists in tracking disease spread and planning disease prevention and control (Robinson, 2007; Wei et al., 2020; Meddah, 2022).

Visualizing trajectories. A central part of the visual analysis of movement is the visualization of trajectories. In a survey, Andrienko and Andrienko (2013) categorize existing methods into four groups: (1) visualizing the spatial and temporal component of individual trajectories, (2) visualizing the movement characteristics and attributes, (3) using generalized visualizations for multiple trajectories, and (4) investigating movement in relation to other moving objects.

One commonly used trajectory visualization technique is the *space-time cube* (Kraak, 2003). This method presents trajectories in a three-dimensional space, where the horizontal plane



(a) space-time cube (b) trajectory wall (c) heat map

Figure 7.2: Sketches for different visualization approaches for trajectories. (a) In the space-time cube, the vertical dimension indicates time. (b) In the trajectory wall, the vertical dimension displays multiple trajectories. Additional properties like transportation mode (e.g., blue for walking and red for running) can be color-coded to improve exploration. (c) In heat maps, the point density is displayed, making identifying areas with high activity easy. Background map: ©OpenStreetMap contributors ©CARTO

represents spatial information while the height dimension encodes the time, see Figure 7.2a. Tominski et al. (2012) extend this approach of using 3D visualizations with the concept of a *trajectory wall*, by visualizing multiple trajectories at varying heights, see Figure 7.2b. They utilize color-coding to represent attribute data along the trajectories. However, these three-dimensional approaches are limited by potential obstructions within the scene. Li et al. (2017) address this issue by optimizing the viewpoint to minimize these obstructions and thus maximizing the exploration effectiveness. Nonetheless, these methods are only effective for visualizing a small number of trajectories.

Heat maps are another prevalent visualization technique, representing spatial values by using color gradients, see Figure 7.2c. Density-based heat maps visualize the kernel density (Silverman, 2018) of the points and have frequently been applied to visualize large trajectory datasets. Scheepens et al. (2011) developed an interactive, heat-map-based visualization for multivariate trajectories by overlaying density fields for different attributes. Similarly, Oksanen et al. (2015) and Sainio et al. (2015) focus on generating heat maps for sports tracking data, considering participation bias in crowd-sourced data (Antoniou and Skopeliti, 2015) to safeguard user privacy. Density-based visualizations allow the summarization of large sets of trajectories and facilitate the extraction of spatial patterns from the data. However, this generalization process results in loss of information, such as the temporal aspect of the trajectories. To address this limitation, Bonerath et al. (2020) introduce a data structure to dynamically generate density maps for given time windows.

Baur et al. (2022) present "*PATHFINDER^{VIS}*", a framework to efficiently filter and visualize large sets of trajectories. This tool offers multiple approaches for visualizing data, including heat map visualizations, clustering common trajectory segments, and continuously refining results to reduce processing time.

Due to their elongated nature in the direction of travel and minimal extent perpendicular to it, trajectories pose challenges for display on regular screen sizes. Thus, the entire trajectory can usually only be visualized on small map scales. However, when zooming in to examine details, the overall context of the trajectory gets lost. In Chapter 8, an approach to visualize the off-screen evolution of trajectories is presented, allowing the user to keep track of the global context even in larger map scales. Related work addressing the mapping of off-screen objects is explored in the corresponding chapter.

7.2 Visualizing route choices

In Part II of this thesis, mathematical approaches for learning routing preferences from trajectory data have been discussed. The general field of route choice and travel behavior has also been studied in the context of geovisualization. Research in this area can be broadly classified into two groups: communicating visualizations that provide route recommendations and exploratory visualizations that support extracting patterns from existing routes.

Route recommendations. In addition to providing personalized route optimization for individual users, route recommendations can also play a crucial role on a policy level in managing traffic flow (Ringhand and Vollrath, 2018; McCall et al., 2015). Geovisualization tools can support route recommendation systems by enhancing users' decision-making processes. Chen et al. (2017b) present, *PathRec* an interactive recommenation tool for touristic routes in cities. Based on crowd-sourced image data, their system extracts features for points of interest (POIs) and creates route recommendations while visualizing the diversity of POIs along different paths. Fuest and Sester (2019) propose a framework for visualizing different routes between two locations in a road network. They propose two approaches to recommend the (temporarily) optimal route to the user: one method distorts the length of non-optimal routes, making them appear longer than they actually are, while the other distorts the smoothness of non-optimal paths. The primary objective of these approaches is not to dictate a specific path to the user but to assist them in making informed route choices. In a subsequent study, Fuest et al. (2021) evaluate six different design variants for route choice recommendation through a user study. Their findings suggest that route visualizations that create the impression of more convenient travel, such as those employing length or smoothness distortion, support users' decision-making process. Rauscher et al. (2024) introduce *SkiVis*, a route recommendation system for skiing. SkiVis enables users to interactively receive route recommendations based on their preferences. Their recommendation system uses crowd-sourced VGI trajectories from OSM.

Exploring routing patterns. Several visual analytics systems have been developed to support route choice modeling (Lu et al., 2017; Shin et al., 2023). These systems commonly support the analyst in three stages. The trajectory dataset is reduced to specific origin-destination pairs in the filtering stage. In the second stage, common routes within the filtered trajectories are visualized to allow for exploration. In the last stage, analysis and hypothesis construction is supported. Liu et al. (2011) concentrate on extracting and visualizing areas characterized by high route diversity, wherein numerous trips with the same origin-destination pair exhibit varying routes. Although the systems mentioned above provide basic modeling of the route choices, their primary emphasis lies in providing tools and visualizations for interactive visual analysis.

In contrast to exploring route choice based on existing trajectories, there has been limited research on visualizing known routing preferences. In Chapter 9, an approach to generate schematic isochrones, i.e., lines of equal travel time, in public transportation networks is presented. This approach can be utilized for routing profiles, enabling the visualization of reachable areas based on specific routing preferences. These visualizations can support analysts by helping them identify road network deficits.

Chapter 8

Visualizing the Off-Screen Evolution of Trajectories

The following chapter is mainly taken from a joint work with Friederike Amann and Jan-Henrik Haunert (Forsch et al., 2022a). This chapter introduces an approach for visualizing the off-screen continuation of trajectories. The method resolves around glyphs: for each trajectory leaving the displayed area, a glyph is generated at the screen's margin that visually encodes how the trajectory evolves beyond this margin. Three different glyph designs are introduced, and their usability is evaluated in an online user study. I (Axel Forsch) developed the glyph design and algorithm for computing them. With Friederike Amann's assistance during a student job, we implemented the algorithm and conducted the user study. Jan-Henrik Haunert provided overarching support and supervision throughout the process, from conceptualization to final publication.

In the context of Volunteered Geographic Information, large sets of trajectories of humans and animals are collected. Visually analyzing these trajectories is often complicated due to limited display sizes. For instance, when a user chooses a large map scale to inspect the details of a trajectory, only a small part of the trajectory is visible on the map. Therefore, this chapter presents an approach for visualizing the off-screen evolution of trajectories, i.e., their continuation outside of the displayed map. The propsed method involves visual cues in the form of glyphs displayed at the map's boundary and consisting of one or multiple disk sectors of varying size and opening angle. These glyphs indicate the direction and variability of a trajectory's continuation outside the map frame. An algorithm for computing the glyphs efficiently is presented and the glyphs are evaluated in a user study. The results show that the glyphs are intuitive to understand even without explanation. Finally, the chapter concludes by suggesting enhancements to the glyph design based on the study outcomes.

8.1 Introduction

Volunteered geographic information (VGI) provides a rich source for spatial analysis (Goodchild, 2007). In movement pattern analysis, for example, researchers profit from large sets of crowd-sourced animal and human trajectories (Huang et al., 2013). For instance, we imagine a city planner working with a map that displays a set of trajectories of cyclists as well as information on the geographical context. A challenge arises if the city planner zooms in to a large map scale, e.g., to inspect a local area for planning reasons. Only small parts of the trajectories can be shown at this large map scale, and the overall context gets lost. Therefore, we aim to augment the map with visual cues about how the trajectories evolve outside the map frame. With this, we intend to provide the city planner with wider-area information about the movement of the cyclists.

Apart from movement pattern analysis, the visualization of trajectories also matters in the context of navigation systems and location-based services (Narzt et al., 2004). For example, a hiker may wish to follow a route computed by an algorithm or a GNSS track that has been recorded and voluntarily shared by a different user. In this application, the trajectory must be visualized on a mobile device such as a smartphone or a hand-held navigation system. The mobile device shows a map providing a local view of the track to allow the hiker to decide where to turn. Additionally, the hiker is interested in knowing how the trajectory will evolve in the next few minutes. However, due to the limited size of the device, this part of the trajectory cannot be displayed on the currently visible part of the map. Thus, the hiker faces a similar problem as the city planner. Using visual cues to indicate the off-screen evolution of trajectories could ease this problem.

In this thesis, we follow this idea by using glyphs indicating the off-screen evolution of trajectories (i.e., their continuation outside the displayed map). In particular, we aim to visualize an initial section of the trajectories' off-screen evolution to provide a more global comprehension of the trajectories' direction in addition to the mere local view. Figure 8.1 shows an example with three trajectories (red). The glyphs (blue) are computed for each trajectory leaving the map frame and are displayed as a map overlay within a margin along the currently displayed map extent (gray). The glyphs are varied in length, opening angle, and orientation to distinguish different directions and shapes of the off-screen parts of the trajectories.

In detail, each glyph is computed by searching the *disk sector* with the smallest perimeter covering the trajectory's off-screen part. For most applications, not the evolution of the trajectory as a whole is of interest, but only of the neighboring off-screen section. The dimension of this considered section, the *look-ahead value*, is part of the parametrization of the glyphs. The disk sector must only cover a certain share of the off-screen section to account for possible outliers and filter out the trajectory recordings' local protrusions. In order to fit onto the overlay area, the initially calculated disk sector is scaled down with a constant factor for all glyphs. Next to this basic glyph, we developed two additional glyphs that use multiple disk sectors. These advanced glyph versions allow for a more detailed trajectory preview at the cost of higher complexity. A user study is conducted to analyze whether this increase in complexity is warranted.

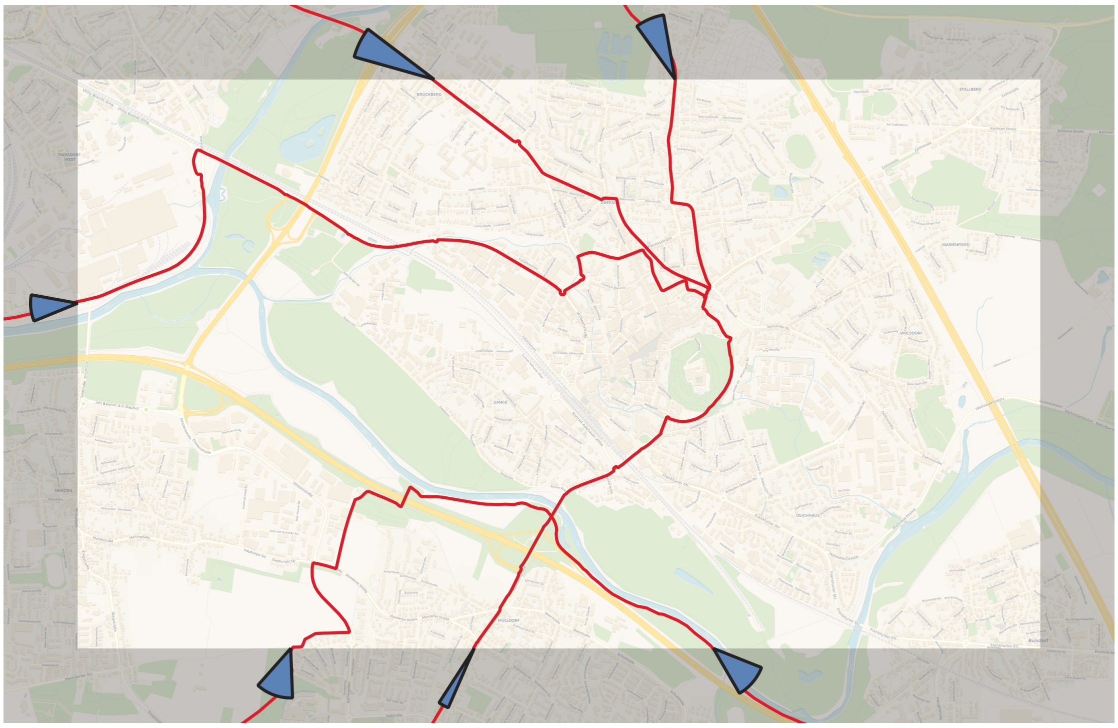


Figure 8.1: The glyphs (blue) indicate the direction in which the displayed trajectories (red) will evolve outside of the map frame. They are displayed within an overlay area along the map extent (gray).

The chapter is structured as follows. In Section 8.2, related work is introduced. Section 8.3 presents the methodology for generating the glyphs. Section 8.4 deals with the user study. The chapter is concluded in Section 8.5. The source code for generating the glyphs can be accessed at https://github.com/GeoinfoBonn/offscreen-evolution.

8.2 Related work

The visualization of trajectory data has two main applications: user navigation and the visual analysis of large trajectory data sets in expert user systems. In the context of user navigation, visualization on small-screen devices has received considerable attention during the last decades. Much research has been done on visualizing point features outside of the screen, while visualizing off-screen line features has not been considered yet. At the same time, visual exploration techniques for trajectory data have been developed. The following reviews previous work on these topics as the presented approach aims to support both tasks.

Visualization on small-screen devices. With the advent of small mobile devices, new challenges in cartography have emerged. To deal with a limited storage capability and a low resolution, Sester and Brenner (2005) presented a progressive generalization approach to transmit

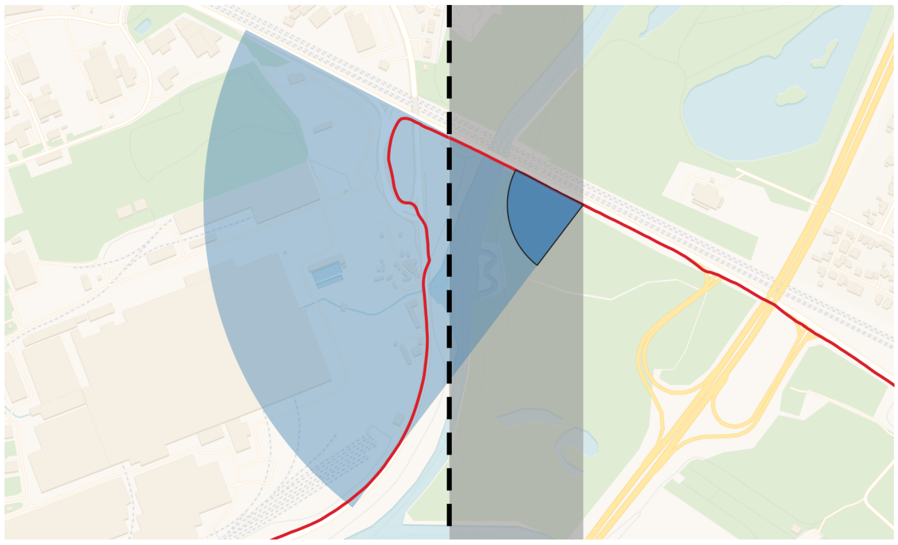


Figure 8.2: The basic glyph with displayed map extent (right of dashed line) and off-screen part of the map (left of dashed line). The glyph (solid blue) is a scaled-down version of the minimal disk sector covering a certain share (here: 95%) of the neighboring off-screen trajectory points (light blue) such that it fits on the overlay area (gray).

detail and overview information. Yoo and Cheon (2006) introduced the fisheye view as a visualization method that effectively lays out information on the screen. Carmo et al. (2007) integrate filtering mechanisms based on semantic criteria, object relevance, and number of displayed objects to visualize icons superimposed on a map. Gedicke et al. (2021) worked on label placement on small screens. Their method is designed to reduce the necessity to zoom the map while exploring the data to ensure a high level of detail while keeping the global context.

Visual exploration of trajectory data. In the context of large sets of trajectories Andrienko and Andrienko (2010) propose aggregation and summarization methods. Given the increasing amount of available urban data Ferreira et al. (2013) target the visual exploration of the data. Their model lets users query taxi trips visually and supports origin-destination and spatio-temporal queries. More recently, Custers et al. (2021) presented a different approach for identifying mobility patterns based on simultaneous aggregation and simplification of both the trajectories and the underlying network. Visual exploration has also been studied in the context of large groups of moving entities. Scheepens et al. (2014) developed non-overlapping glyphs representing aggregated groups of moving objects. Cakmak et al. (2020) present *MotionGlyphs* as a visualization technique for dense spatio-temporal networks. They use these glyphs to analyze animal behavior. Similarly, Demšar et al. (2015) perform movement analysis based on large animal movement data. They use a trajectory's segmentation into pieces that fulfill geometric criteria and identify a representative path for a certain set of tracks. Moreover, the authors apply several multi-dimensional visualization methods, such as the space-time cube. The space-time cube, initially introduced by Hägerstrand (1970), is one of the most frequently used visualizations for spatio-temporal data. The data is visualized in 3-D using two dimensions for the spatial information while the last dimension represents time (Kraak, 2003; Filho et al., 2020; Ssin et al., 2019). Visual exploration of trajectories in the context of transportation is investigated by Markovic et al. (2020). The authors create heat maps of important waypoints and published several interactive online animations.

Visualizing off-screen landmarks. Several studies focus on visualizing off-screen objects by glyphs displayed at the boundary of the currently displayed map extent. Baudisch and Rosenholtz (2003) present *Halo*, an approach that uses an arc to let the user infer the direction and distance to the off-screen object: A circle surrounds the object and reaches into the border region on the displayed map. The user can understand the distance information without needing a legend based on *amodal perception*, which allows the user to automatically complete shapes that are only partially visible. At the same time, Zellweger et al. (2003) presented *City Lights*, which is a fisheye technique where size and distance information are conveyed using different colors.

Gustafson et al. (2008) concentrated on a *Wedge* approach using acute isosceles triangles to achieve a clutter-free visualization. The tip of the Wedge coincides with the off-screen object while the base and the legs are displayed on screen. Like the Halo approach, the user mentally triangulates the tip beyond the screen. Overlap and clutter between different Wedges are avoided by changing rotation or intrusion on the display window. An evaluation showed that users were significantly more accurate when using Wedges than Halos.

Li and Zhao (2017) measured the distance perception on resized icons representing off-screen landmarks. The further away a landmark is, the smaller the corresponding icon is displayed. The authors conducted a study that revealed possible misinterpretations of the off-screen icon's location and its size which is meant to represent the actual distance.

Burigat et al. (2006) performed a comparative evaluation of the *Halo* and *City Lights* approaches and approaches using arrows. The arrow approaches use small arrow symbols indicating the direction and distance to a certain object using varying arrow orientation and size. The larger the arrow, the closer the off-screen object is. One central finding is that arrow approaches are more effective than the Halo approach when the object configuration would cause cluttering on the small screen.

A further study of Burigat and Chittaro (2011) compared the aforementioned approaches to *Overview+Detail*. The latter is not based on contextual cues but shows the map with an additional detailed view of an area of interest. This approach has limitations, especially on small mobile devices, due to readability reasons.

Information visualization. The visual exploration of data also received attention in the context of statistics. Tukey (1977) introduced the term exploratory data analysis for techniques to discover patterns in data sets. Most related to our work are box-and-whisker plots, which graphically indicate the locality and spread of the data points by showing a box containing all points between a lower and an upper percentile. Analogously, we use contextual cues shaped as disk sectors, which contain a given percentile of the off-screen trajectory points.

The studies discussed above present methods to handle huge sets of trajectories and to display information on small-screen devices. They also introduce cues to convey information on off-screen landmarks. Nonetheless, visualizing off-screen trajectories has not been considered in previous work. This chapter fills this gap by combining these approaches.

8.3 Methodology

In this section, we present the design of the basic glyph and outline our algorithm to compute it. We then extend this definition to generate more advanced versions of the glyph.

We start by introducing the terminology and formalizing the problem. We are given a map frame of a fixed extent and one or multiple trajectories passing through it. Following previous work on off-screen visualization, we define an *overlay area* of fixed width b along the map display's boundary. This area is used to display the glyphs. The rest of the map frame is the *main area*. Everything outside of the main area we call *off-screen* (even though the overlay area is still technically on the current map extent). If a trajectory enters the map frame more than once, we consider each on-screen part separately. Each trajectory that passes the main area has zero, one, or two off-screen parts, as both ends can be off-screen. Each off-screen part is of arbitrary length, and not all of it might be of interest to the application. Therefore, only an initial section of each off-screen part of a given length defined by a *look-ahead value* ℓ is considered, the *off-screen section*. Each off-screen section's start point is the trajectory's crossing point with the overlay area's inner boundary.

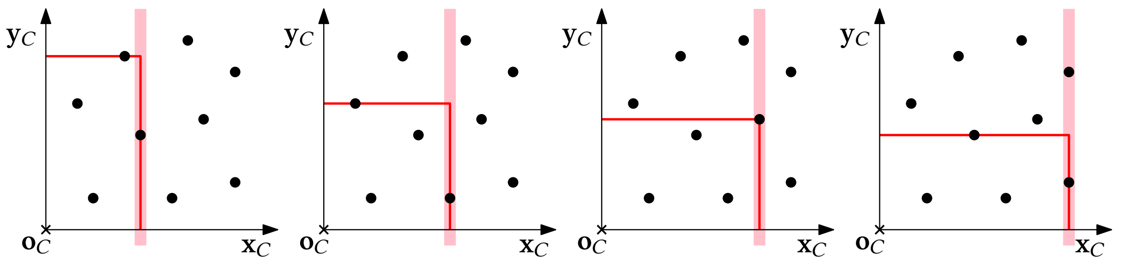
The problem is finding a suitable visualization for the off-screen section. The key properties of the off-screen section we want to communicate are its direction and direction variability. Additionally, we try to keep the visualization as simple as possible. We solve this task using glyphs in the form of disk sectors that we display on the overlay area of the map. Disk sectors are suitable for this task as their orientation encodes the direction, the opening angle encodes the direction variability, and the radius encodes the velocity. To formalize this, the idea is to find a disk sector attached to the off-screen section's start point that covers the points of the off-screen section and that is minimal with respect to a prescribed objective function f. We soften this definition to account for possible outliers and filter out local protrusions in the trajectory so that only a share κ of the points must be covered. In order to fit on the overlay area, the glyphs are scaled down with a constant factor.

The approach for computing a glyph can be broken down into two parts. The first part is finding the minimal disk sector given the orientation of its right boundary (i.e., the line segment bounding the disk sector to the right). We developed an algorithm that computes a minimal enclosing rectangle; by transforming the input points to polar coordinates, we can apply this algorithm to find the minimal disk sector. The second part is finding the orientation of the right edge that minimizes the disk sector.

Finding a minimal k-rectangle. Given a coordinate frame C together with a point set $P = \{p_1, p_2, \dots, p_n\}$ of n input points with $x_i \geq 0$ and $y_i \geq 0$ for $i = 1 \dots n$, we search for a rectangle with one corner at the point of origin $\mathbf{o}_C$ of C and edges parallel to the coordinate axes of C, such that it covers at least $k = \lceil \kappa \cdot n \rceil$ of the input points. We call such a rectangle a *k-rectangle*.

With MinKRectangle we refer to the problem that asks for a k-rectangle minimizing f. Our assumptions with respect to f are that it is monotonically increasing in both the width and the height of a k-rectangle and that it can be evaluated in constant time for a given k-rectangle. This assumption holds, for example, if we define f as the perimeter or the area of a k-rectangle.

A k-rectangle is *non-dominated* if no other k-rectangle is smaller in one of the two dimensions, width or height, and at most as large in the other dimension. We solve MinKRectangle by enumerating the set of all non-dominated k-rectangles and taking one optimal with respect to f. Using a sweep-line approach, we enumerate the non-dominated k-rectangles in increasing order of their widths. This yields an algorithm with running time in $O(n \log n)$. Due to the monotonicity of f, any optimal solution has to be non-dominated. Hence, the algorithm is correct. This approach is visualized in Figure 8.3. Pseudo-code for the algorithm is given in Algorithm 3.



(a) $f = 66$ (b) $f = 64$ (c) $f = 70$ (d) $f = 72$

Figure 8.3: Visualization of the algorithm for $k = 4$. All four non-dominated k-rectangles are displayed in (a)-(d) with the sweep line shown in light red; f corresponds to the area. The optimum with respect to f is found in step (b).

Computing minimal disk sectors. The algorithm for solving MinKRectangle is based on the fact that the parameters of a rectangle, its width and height, are equivalent to the x- and y-values of its spanning points. In the case of disk sectors, the defining parameters are the opening angle θ and the radius r. Therefore, we can transform the points of the off-screen section from cartesian coordinates (x, y) to polar coordinates (φ, ρ) and use these as input for the algorithm. The range ρ is equivalent to the radius r, and the direction φ is equivalent to the disk sector's opening angle θ. The radius is always positive, and the direction can be normalized to the interval $[0, 2\pi)$ so that the requirements on the input points for MinKRectangle are fulfilled. The objective functions for minimizing area or perimeter are the following:

$$f_{\text{area}} = \frac{1}{2}\theta r^2 \tag{8.1}$$

$$f_{\text{perimeter}} = 2r + \theta r \tag{8.2}$$

For computing the direction, a reference direction ϕ_0 is needed. This reference defines the right boundary of the resulting disk sector. When searching for the smallest disk sector enclosing a given set of points, one of the points must be located on the right boundary of the disk sector.

Algorithm 3: MinKRectangle

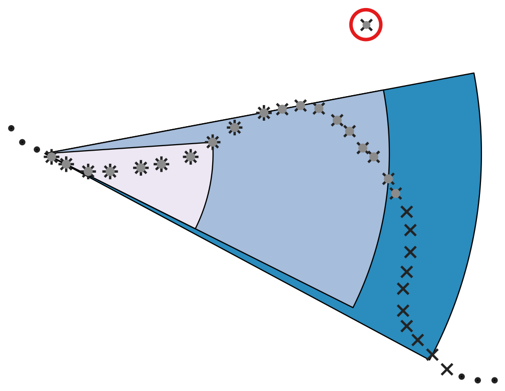
```
Data: point set $P$, number of points $k$, objective function $f$
Result: width and height of a $k$-rectangle minimizing $f(width, height)$
$X[1 \dots n], Y[1 \dots n] = P$ sorted by $x$-/$y$-coordinate respectively ;
$i_X = k$ ; // index in $X$
$i_Y$ = index of point with maximum $y$ of first $k$ points in $X$ ; // index in $Y$
$\bar{x} = X[i_X].x$, $\bar{y} = Y[i_Y].y$ ;
$f_{\text{best}} = f(\bar{x}, \bar{y})$ ; // initial solution defined by the $k$ left-most points
for increment $i_X$ until $\|P\|$ do
    if $X[i_X].y \geq \bar{y}$ then
        continue ; // skip point as its y-value is higher than current y
    $\bar{x} = X[i_X].x$ ;
    for decrement $i_Y$ until $k$ do
        if $Y[i_Y].x > \bar{x}$ then
            continue ; // skip point as its x-value is higher than current x
        $\bar{y} = Y[i_Y].y$ ;
        $f_{\text{curr}} = f(\bar{x}, \bar{y})$ ;
        if $f_{\text{curr}} < f_{\text{best}}$ then
            update $f_{\text{best}}$ and remember solution as $(\hat{x}, \hat{y})$ ;
        break;
return solution $(\hat{x}, \hat{y})$ corresponding to $f_{\text{best}}$ ;
```

Otherwise, we can shrink the disk sector on this side to obtain a smaller one containing the same number of points. Therefore, we apply the polar version of MinKRectangle once for each input point p_i, using the direction to point p_i as ϕ_0. Finally, we select the minimal disk sector as the solution. Applying the algorithm for MinKRectangle n times results in an overall running time of $O(n^2 \log n)$. Sorting the input points only once for each of the two dimensions reduces the running time to $O(n^2)$.

Different glyph designs. Three different glyphs of increasing complexity are developed based on the described algorithm.

The *basic glyph* comprises a single disk sector. The look-ahead value ℓ and the inlier ratio κ parameterize it. This glyph is kept simple to make it easy to read and understand. As a drawback, the glyph's information is limited as the considered off-screen section is summarized into a single disk sector.

To tackle this issue, the *staggered glyph* comprises multiple disk sectors, each having different look-ahead values. The number of disc sections n is an additional parameter for this type. The individual disk sectors are layered in order of the look-ahead value, with the largest value in the back. Figure 8.4a shows the staggered design. The largest disk sector in this glyph is identical to the basic glyph. The staggered glyph allows for finer differentiation of the tra-


(a) staggered glyph

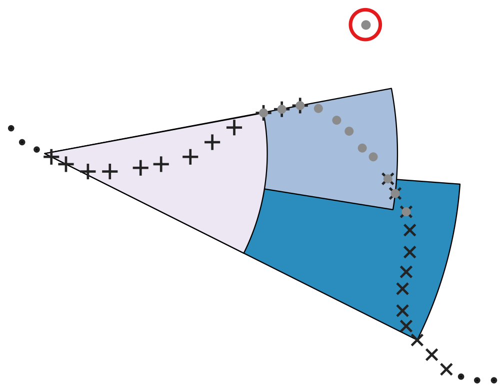

(b) consecutive glyph

Figure 8.4: Considered vertices of the first disk sector (black plus), the second disk sector (gray dots), and the third disk sector (black cross) for the advanced glyphs. Not all vertices need to be covered by the disk sectors, so the outlier (marked red) is not included in the glyphs.

jectory's off-screen section's shape. For instance, turns at more distant look-ahead values are only visible in the larger disk sectors.

In the staggered variant, the neighboring off-screen section is part of all disk sectors. To counteract this, the *consecutive glyph* decouples the disk sectors by calculating them for consecutive subsections of the off-screen section. Each disk sector only considers points between two given look-ahead values $\ell_{\min}$ and $\ell_{\max}$. Figure 8.4b shows the consecutive glyph. Due to the inlier ratio, not all points considered need to be covered by the individual disk sector. Therefore, the disk sectors can be disjoint. To ensure that a glyph is a contiguous region and thus to improve readability, the look-ahead ranges of two consecutive disk sectors may overlap ($\ell_{\max,i} > \ell_{\min,(i+1)}$). This is comparable to a sliding average filter.

Glyph scaling. The glyphs need to be scaled down to fit on the overlay area. By construction, the glyphs cover the off-screen section of the trajectory, which has a length ℓ. Considering the extreme case that the off-screen section is a straight line, the maximum possible radius $r_{\max}$ for a glyph is ℓ. This glyph needs to fit on an overlay area of width b. We scale each disk sector's radius with a scale factor $s = \frac{b}{\ell}$ to ensure that all glyphs fit on the overlay area.

8.4 Evaluation

In this section, we evaluate the glyphs. Specifically, we want to analyze whether the glyphs are intuitive to understand and convey a correct impression of the off-screen evolution of the trajectory. Particular emphasis lies on the comparison of the glyph types.

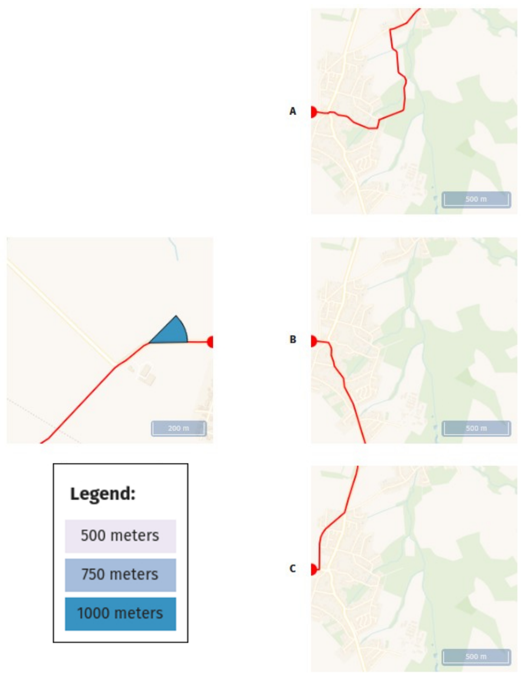
For the evaluation, we performed an online user study. The details of the study design are given in Section 8.4.1. In Section 8.4.2, we present a detailed discussion of our results.

8.4.1 Experimental setup

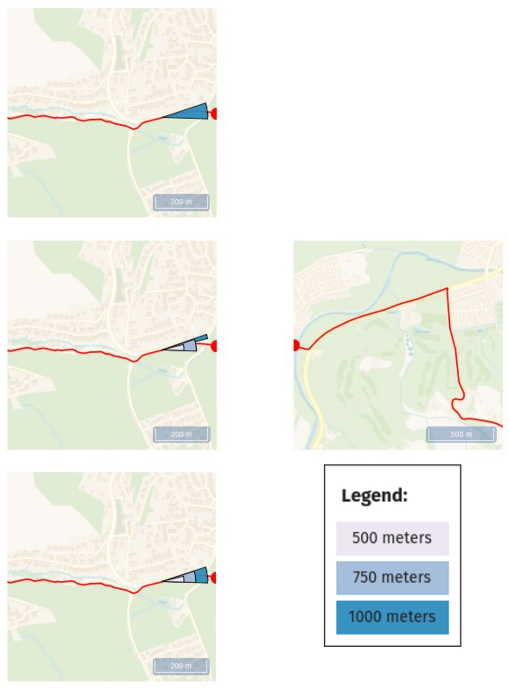
In the user study, participants were asked questions on the off-screen evolution of trajectories based on the glyphs. The study is separated into two different tasks:

1. *Interpretation* Given a real-world trajectory and the corresponding glyph, the participant has to select the correct off-screen evolution out of three options.
2. *Preference* The three glyph types and the corresponding on- and off-screen trajectories are shown. The participant is asked to pick the glyph they prefer for the given situation.

Figure 8.5 shows two examples for these task types. Task 1 (Figure 8.5a) features multiple-choice questions where exactly one option is correct. The answers to this task give information on the interpretability of the glyphs. We recorded the given answer and the time each participant needed to answer. All options in Task 2 (Figure 8.5b) are correct; they only differ in the type of the displayed glyph. This task aims to get a subjective opinion of the participant on which glyph matches the situation best. A fourth option is provided if none of the glyphs pleases the participant. In the study, participants receive six questions for Task 1 and three for Task 2. The complete study can be examined at https://www.geoinfo.uni-bonn.de/offscreen-evolution-study/.



(a) Task 1: Interpretation



(b) Task 2: Preference

Figure 8.5: Illustration of the tasks. Task 1 tests if the glyph comprehensively indicates the further evolution of the track. The correct trajectory continuation has to be selected. Task 2 allows the participant to choose the most appropriate glyph. All three variants fit the trajectory on the right side, but the participant can select his preferred glyph type.

Question layout. To avoid a learning effect in the study, we separated the participants for Task 1 into three groups: A, B, and C, and generated six question instances. An instance corresponds to a map frame with a single real-world trajectory and three possible off-screen evolutions, of which only one is the original, correct one. All three different glyph types are shown to each group so that answers for each glyph type are present for each instance. The instances are sorted by glyph type to each group, starting with the basic glyph, continuing with the staggered one, and ending with the consecutive one. We choose this order to present the glyphs in order of rising complexity. An overview of the study layout for Task 1 is given in Table 8.1.

The study starts with a short tutorial introducing the two tasks. For each task, one introductory instance is presented. The tutorial guarantees that the timing of the answers is not skewed at the beginning. We deliberately refrain from explaining the glyphs to evaluate their intuitiveness.

Task 1	Glyph Type		
	Basic	Staggered	Consecutive
Instance 1	A_1	B_3	C_5
Instance 2	C_1	A_3	B_5
Instance 3	B_1	C_3	A_5
Instance 4	A_2	B_4	C_6
Instance 5	B_2	A_4	B_6
Instance 6	C_2	C_4	A_6

Table 8.1: Task 1: each group (A, B and C) receives six questions: two per glyph type. The subscript indicates the question order.

Instance generation. To simulate a real-world example as closely as possible, we use real-world trajectories recorded from cyclists in the metropolitan areas of Cologne and Bonn to generate the instances. The map frames are selected randomly along the trajectories. The map frames all have the same extent and are rotated such that the trajectory leaves the map at the right-hand side boundary to ensure equal preconditions. The wrong options for the off-screen evolution used for Task 1 are generated by selecting an alternative destination outside the displayed map and using an online bicycle routing service[1] to compute an optimal cycling path to this destination.

The parameters for the glyphs are identical across all glyph types and instances. We selected the parameters according to the setting of bicycle navigation. Specifically, we limited the maximum look-ahead value $\ell_{\max}$ to 1000 meters as this relates to 3 to 4 minutes of cycling, which seems sufficient for general usage. Further, we selected the inlier ratio $\kappa = 0.90$ and the width of the overlay area $b = 200$m. For the staggered and consecutive glyph, additional look-ahead

[1] http://brouter.de/brouter-web/

values for a second and third disk sector of $\ell_0 = 500\text{m}$ and $\ell_1 = 750\text{m}$ and an overlap of 50% of the look-ahead ranges have been chosen, resulting in the ranges $\mathcal{I}_1 = [0, 500], \mathcal{I}_2 = [250, 750]$ and $\mathcal{I}_3 = [500, 1000]$ (all in meters). All disk sectors are minimized regarding their perimeter using $f_{\text{perimeter}}$ as the objective function. We refrain from using the f_{area} as initial testing showed that this favors narrow disk sectors, which are hard to recognize on a map. Using the perimeter instead creates compact disk sectors. Using our implementation, the computation of individual glyphs took less than 0.1 seconds. Further testing showed that glyphs for large instances with $n = 2000$ points and an inlier ratio of 50% could be computed in under 0.3 seconds.

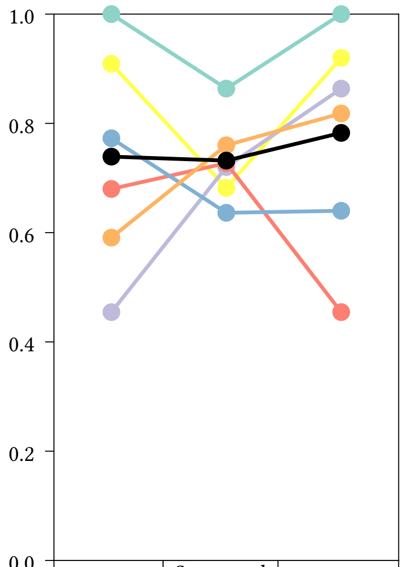
8.4.2 Results and discussion

The study was conducted online and distributed via social media, the student association, and word-of-mouth recommendations. In total, 72 participants volunteered for the study and randomly got assigned to one of the groups, resulting in the following group sizes: $||A|| = 26$, $||B|| = 22$, $||C|| = 24$. The participants were asked for basic demographic and social statistics. Most participants were in the age range of 16 to 35 years ($\approx$ 74%). Approximately 54% of the participants stated male as their gender, while 36% stated female.

Quantitative analysis. Task 1 allows us to analyze the interpretability of the glyphs. Participants scored one point for each question if they selected the correct answer and zero otherwise. Divided by the number of questions, the score gives information about the share of correct answers. Figure 8.6a displays the share of correct answers, differentiated by instance and glyph type. Overall, 75.2% of the answers were correct. However, there are substantial differences when comparing individual instances to each other. For example, Instance 1 has a correctness of 94.4% while Instance 4 has only been answered correctly 61.8% of the cases. While all three glyph types have a similar correctness between 73.2% and 78.3%, the results for a single instance differ substantially for different types. For example, the consecutive glyph has nearly twice as much correctness as the basic one in Instance 3. These observations suggest that the interpretability of the glyph highly depends on the individual instance. The length of the look-ahead ranges defines the resolution of the glyph. Thus, glyphs for very curvy trajectories, where the lengths of the curves are shorter than the considered look-ahead ranges, are hard to interpret because this resolution is too low.

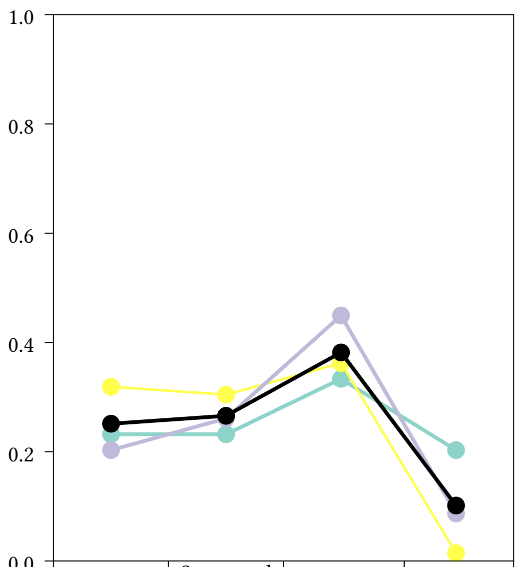
In addition to the score, we recorded the answering times for each question to conclude how intuitive the glyphs are, assuming the faster a participant could answer, the more intuitive the glyph is. The average answering times for each glyph type were $t_{\text{basic}} = 29.6\text{s}$, $t_{\text{staggered}} = 22.5\text{s}$, and $t_{\text{consecutive}} = 17.9\text{s}$. While this implies that the more complex glyphs are more intuitive, we avoid emphasizing these results, as the types occurred in order basic $\longrightarrow$ staggered $\longrightarrow$ consecutive during the study. Thus, the speed-up might be related to learning effects or subsiding diligence towards the end of the study. Future study designs should account for such effects.

Figure 8.6b shows the glyph preference as stated by the participants in Task 2. The consecutive glyph is preferred by more users than the other glyph types for all instances. Nonetheless, the preferred types are pretty balanced, with the lowest preference still above 20%. Noticeable is the large share (22.2%) of participants that found neither of the shown glyphs fitting for In-



	Basic	Staggered	Consecutive
Instance 1	100.0%	86.4%	100.0%
Instance 2	90.9%	68.2%	92.0%
Instance 3	45.5%	72.0%	86.4%
Instance 4	68.0%	72.7%	45.5%
Instance 5	77.3%	63.6%	64.0%
Instance 6	59.1%	76.0%	81.8%
Total	**73.9%**	**73.2%**	**78.3%**

(a) correctness



	Basic	Staggered	Consecutive	None
Instance 7	23.2%	23.2%	33.3%	20.3%
Instance 8	31.9%	30.4%	36.2%	1.4%
Instance 9	20.3%	26.1%	44.9%	8.7%
Total	**25.1%**	**26.6%**	**38.2%**	**10.1%**

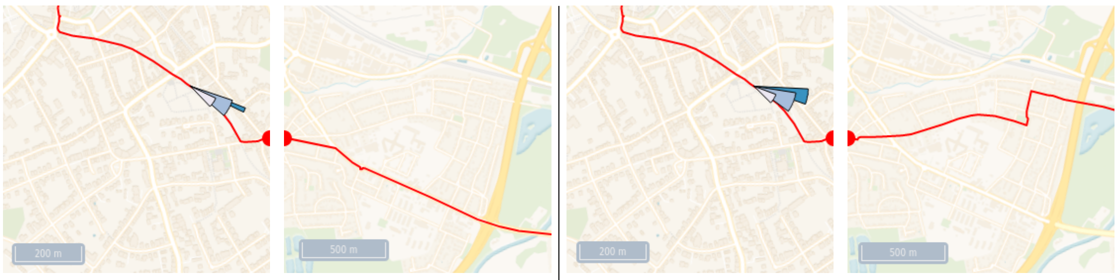
(b) preference

Figure 8.6: (a) Share of correct answers per instance and glyph type; (b) stated preference by instance and glyph type.

stance 7. This coincides with the previous observation that the instance significantly impacts glyph quality.

Qualitative analysis. The quantitative analysis of the result revealed that a large factor determining the readability of the glyphs is the individual configuration of an instance. Therefore, we take a closer look at instances that resulted in ambiguous results in our study.

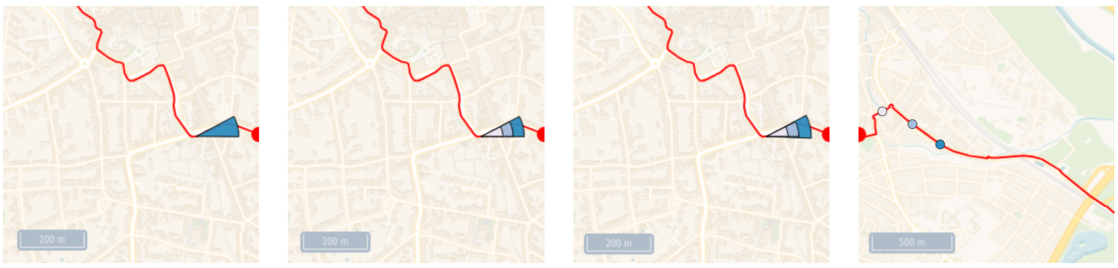
Even though the consecutive glyph achieved the overall best results, its accuracy in Instance 4 is only 45.5% (Figure 8.6a). Of the 22 recorded answers, 12 participants chose the wrong option *A* instead of the correct option *B*. Option *C* was not selected at all. The glyph thus seems to be ambiguous for the first two options. Figure 8.7 shows Instance 4, with the given glyph shown in Figure 8.7a. The narrow disk sector for the largest look-ahead value might convey the wrong perception of evolution more to the top of the map as seen in option *A* (Figure 8.7d) compared to option *B* (Figure 8.7b). For comparison, the correct glyph for the off-screen trajectory in option *A* is shown in Figure 8.7c. Comparing the two glyphs, the disk sectors for the first two look-ahead values are nearly identical, but the last changes significantly. A reason for the high share of wrong answers might be the arrangement of the options in the study (see Figure 8.5a): only option *B* is placed right next to the visible map frame, for the other options the perspective might be skewed and the directions might not be easy to identify.



(a) given glyph (b) Option *B* (correct) (c) glyph for Option *A* (d) Option *A* (wrong)

Figure 8.7: (a) Instance 4, consecutive glyph: (d) many participants selected answer *A*, (b) but *B* is the correct one. (c) Glyph if option *A* was correct.

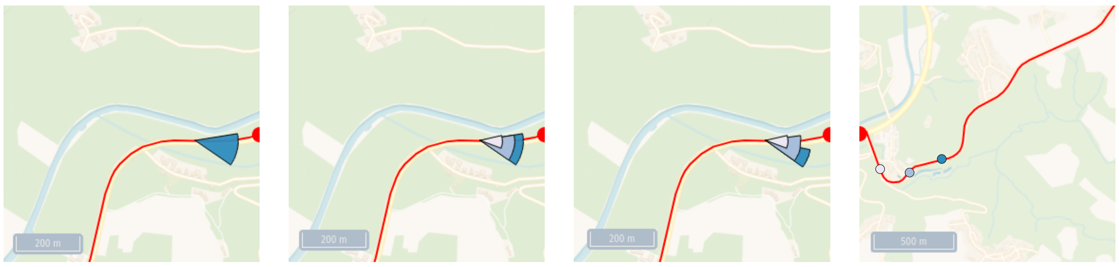
In Instance 5, all glyph types score a similar correctness of 64.0%-77.3%. Figure 8.8 shows the glyphs with the off-screen evolution. All three variants look pretty similar. This is because of the curvy off-screen trajectory section near the cut-off point and the large overlap in the look-ahead intervals. Figure 8.8d shows the look-ahead values 500m, 750m and 1000m. As these positions are derived following the trajectory and due to the curvy nature of the off-screen section, these positions are all very close to the map boundary. The visualization can be improved by selecting the look-ahead position based on a *radius* instead of along the trajectory. Another improvement can be achieved by reducing the overlap parameter. Both of these measures decrease the shared information between the disk sectors and thus increase the information conveyed by the glyph.



(a) basic (b) staggered (c) consecutive (d) evolution

Figure 8.8: Instance 5: all three variants (a) basic, (b) staggered and (c) consecutive look very similar due to the curvy evolution of the trajectory in close vicinity to the cut-off point. (d) The off-screen evolution with loook-ahead values indicated by blue markers.

Many participants could not decide on a preferred glyph in Task 2, Instance 7 (Figure 8.9). Similar to the previous example, the near-off-screen evolution is quite curvy; thus, the maximum look-ahead value is close to the map boundary. Again, we suggest using a fixed radius as a look-ahead value to improve the visualization.



(a) basic (b) staggered (c) consecutive (d) evolution

Figure 8.9: Instance 7: participants cannot decide on a preferred glyph for the given situation. (d) The look-ahead distances are not evident due to the curves in the off-screen part.

8.5 Conclusion

In this chapter, we have proposed using glyphs in the form of disk sectors to indicate how trajectories evolve outside of the visible map frame. We have outlined an algorithm that computes the glyphs in $O(n^2)$ time under relatively mild assumptions regarding the optimization objective. Future research may find an algorithm of sub-quadratic time for specific objective functions, which has been achieved for related algorithmic problems (Eppstein and Erickson, 1994).

We have opted for the perimeter of a glyph as the minimization objective, as minimizing the area of a glyph resulted in very narrow glyphs. We evaluated the approach in an online user study. The results show that the glyphs are simple and informative. Nonetheless, the trajectories' geometry greatly impacts the glyphs' readability, and, in some cases, the tested glyphs do not seem to convey the intended information. In our evaluation, we identified the parameter settings of the glyphs used in our study as one reason for this. For future applications, we therefore recommend reducing the overlap of the look-ahead intervals for glyphs. Additionally, we recommend using a fixed radius to determine the off-screen sections instead of a distance along the trajectory.

In our evaluation, we focused on the efficacy of single glyphs. For the visual exploration of large sets of trajectories, many glyphs must be displayed on the same map (see Figure 8.1). Therefore, challenges such as the overlap between multiple glyphs remain to be addressed in future research.

Chapter 9

Multimodal Travel-Time Maps with Schematic Isochrones

The following chapter is mainly taken from joint work with Youness Dehbi, Benjamin Niedermann, Johannes Oehrlein, Peter Rottmann, and Jan-Henrik Haunert (Forsch et al., 2021). The project originated from a student project attended by Peter Rottmann and myself, Axel Forsch, under the supervision of Johannes Oehrlein and Benjamin Niedermann. The project led to the development of preliminary algorithms designed for routing in multi-modal transportation networks and for visualizing travel times. Building upon the foundational work established during the course, I extended the results by integrating the algorithms and developing a schematic visualization. Jan-Henrik Haunert supervised and supported the development of the algorithms and the write-up.

The automatic generation of travel-time maps is a prerequisite for many fields of application, such as tourist assistance and spatial decision support systems, e.g., to analyze the accessibility of health facilities. The task is to determine outlines of zones that are reachable from a user's location in a given amount of time. The focus of this chapter is on travel-time maps with a formally guaranteed separation property in the sense that a zone exactly contains the part of the road network that is reachable within a pre-defined time from a given starting point and start time. The approach revolves around generating schematized travel-time maps aimed at reducing visual complexity. Each zone is represented by an octilinear polygon, where the edges of the polygons adhere to one of eight pre-defined orientations. Further, map legibility is supported by aiming for polygons with minimal bends. The reachable parts of the road network are determined by integrating timetable information for different modes of public transportation, e.g., buses or trains and pedestrian walkways, based on a multi-modal time-expanded network. Moreover, the generated travel-time maps visualize multiple travel times using a map overlay and incorporate natural barriers like rivers. In experiments, comparisons are made between the schematic visualizations and travel-time maps created using other visualization techniques. Simple but robust quality measures, such as the number of bends and the perimeter of the zones, are used to evaluate the effectiveness of the different approaches.

9.1 Introduction

Travel-time maps provide the user with an easily comprehensible visualization of areas that are reachable from a selected location within a prescribed amount of time. The boundaries of such areas are commonly referred to as *isochrones*. We present a new method for generating *schematic* travel-time maps automatically – a preview of some results generated with our method is provided in Figure 9.1. The motivation for using a schematic visualization is to avoid the high graphical complexity that can occur with existing solutions, which we review in detail in Section 9.2. However, we not only aim for a low graphical complexity but also for a visualization inducing a formally correct classification of the transport network into reachable and unreachable parts.

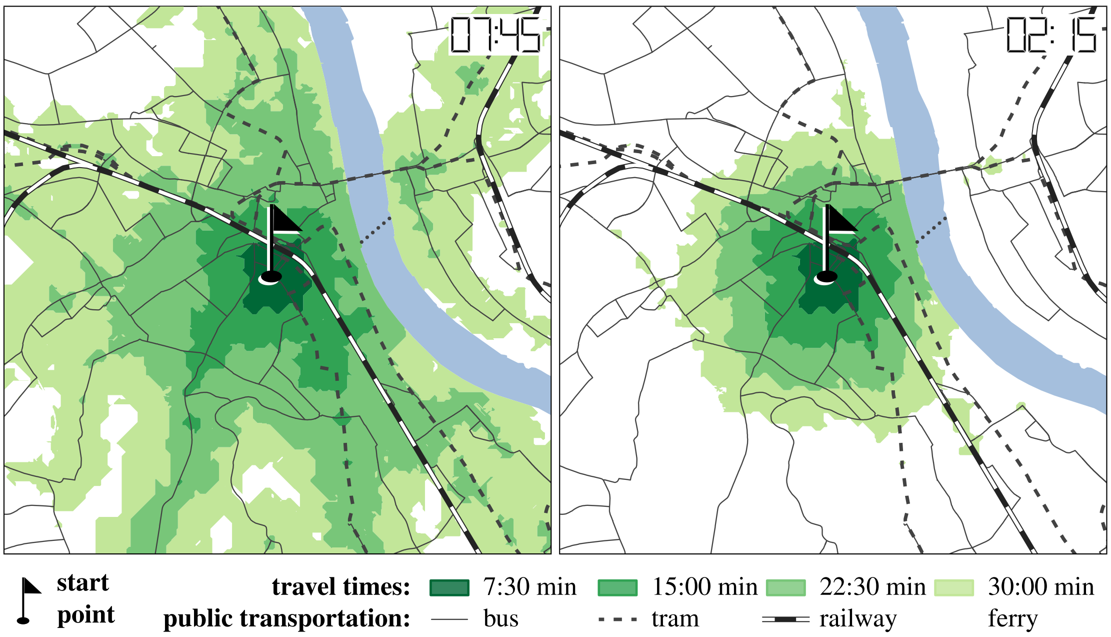


Figure 9.1: Travel-time maps generated for the city of Bonn, Germany. The maps show the accessibility of the road network starting from the same location for different start times in the morning (left) and during the night (right) of a work day.

Before formally defining the property ensured with our method, we discuss a classical approach (referred to as *time buffering* in the following) and its possible flaws. An early map of Melbourne generated with time buffering is shown in Figure 9.2. The approach consists of two steps:

Step 1: Travel times are estimated from a selected point to all stations in the transport network (in the map of Melbourne, the tram and railway network).

Step 2: To visualize the area reachable within some amount of time τ, a disk is drawn around each reachable station s such that the disk's radius equals the available amount of time $\tau - \tau_s$ (where τ_s is the travel time to s) multiplied with a constant (which may correspond to the assumed speed of walking outside of the transport network).

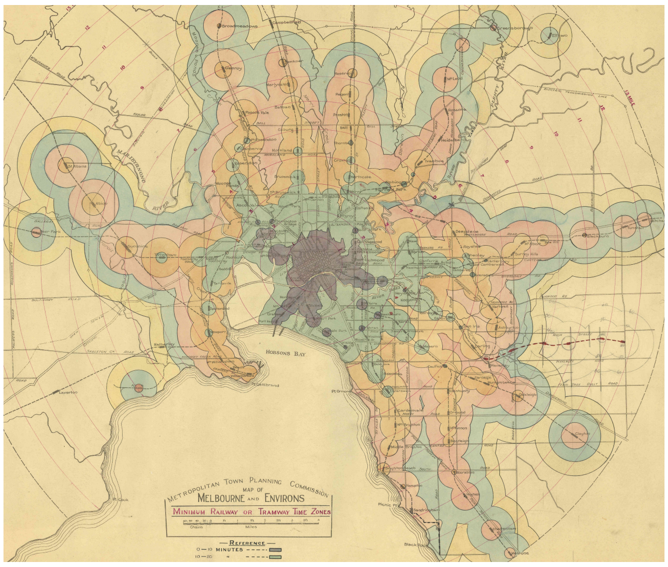

Metropolitan Town Planning Commission
Map of
Melbourne and Environs
Minimum Railway or Tramway Time Zones
Reference
Hobsons Bay

Figure 9.2: An early travel-time map of Melbourne rail transport travel times, 1910-1922. The public transportation is considered: for each reachable station, the remaining travel time is visualized by concentric circles. *Source: Melbourne and Metropolitan Tramways Board – State Library of Victoria.*

Usually, as in the example of Figure 9.2, this approach is repeated with multiple different values for τ, resulting in multiple areas (in the following referred to as *time zones*). Furthermore, the time buffering approach can be easily generalized to situations where, in Step 1, accurate travel times are estimated not only for a discrete set of stations but also for the continuous set of all points in a geometric graph – an example we generated ourselves using public transportation timetable data and road data is shown in Figure 9.3a. We observe that many points in the network that in Step 1 were classified as unreachable (dashed lines) fall into the area highlighted as reachable (red region). We consider this inappropriate since no evidence exists that the space between the roads is traversable. More precisely, the assumed constant walking speed for areas outside the network lacks a reasonable justification. We do not make this assumption and keep the classification of points in the network as it results from Step 1. Still, we would like to use areas to visualize the reachable part of the network. We argue that schematic polygons are particularly well-suited because they lower the graphical complexity. The travel-time map created with our approach is shown in Figure 9.3b. Aside from the low visual complexity, another advantage of schematized time zones is that schematization is widely applied for visualizing transport networks, such as metro maps. Since schematic metro maps are so commonly applied in practice, it can be assumed that most people understand that the geometric information provided with schematic polygons needs to be taken with a grain of salt (i.e., not all points of the map plane contained in a time zone we display are actually reachable). Nevertheless, qualitative information displayed for the transport network, such as the classification into reachable and unreachable parts, is correct.

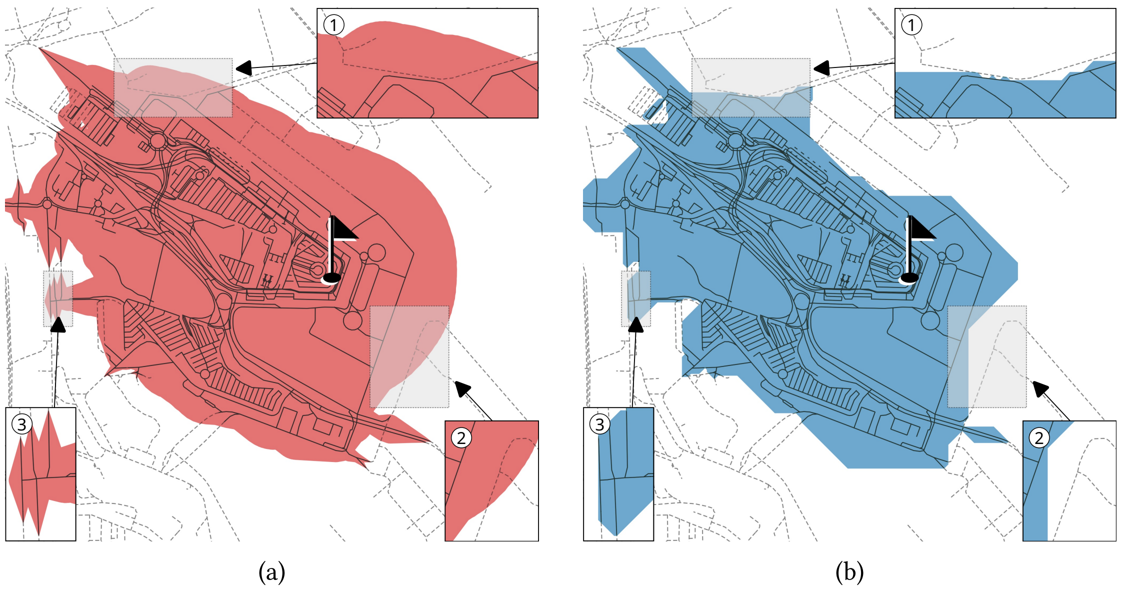


Figure 9.3: A single time-zone of 15 minutes starting at the flag. Unreachable roads are dashed. (a) Time-zone created by buffering the remaining travel time for each point of the road network. ①–②: The time zone covers unreachable roads. ③: Time zone has ramified outgrowths. (b) Time-zone created by our approach. ①–②: Time-zone only covers reachable roads. ③: Schematization simplifies the visualization.

The correct classification of the transport network into reachable and unreachable parts is formalized as follows. A location t in a road network $\mathcal{R}$ is *reachable* from the user's location s in time τ if there is a route r from s to t using footpaths in $\mathcal{R}$ and connections in the public transportation network such that it takes at maximum time τ to travel along r; all other locations of $\mathcal{R}$ are *unreachable*. Due to the possibly disconnected structure of the reachable subnetwork, we describe its outline, the time zone of temporal extent τ, not by a single polygon but rather by a set of polygons whose boundaries do not intersect; we explicitly allow holes in the polygons modeling unreachable parts contained in reachable parts of the network. We consider a time zone *formally correct* if it satisfies the following separation property:

Definition 5 (Separation Property)**.** The time zone Z satisfies the *separation property* if each reachable location of $\mathcal{R}$ is contained in one of the polygons of Z and each unreachable location of $\mathcal{R}$ is not.

In contrast to previous work (Gamper et al., 2011; Krismer et al., 2017; Baum et al., 2018), we follow the idea of *schematic mapping* to only use octilinear orientations for the edges of the time zone, i.e., they are either horizontal, vertical or diagonal. To that end, we discretize the solution space by an octilinear grid. In the exceptional case that our algorithm does not find a strictly octilinear polygon that separates the reachable and unreachable vertices, we allow non-octilinear lines. To keep the visual complexity low, we heuristically minimize the number of bends of the polygons.

Altogether, the core contribution of this chapter is an algorithm for creating a high-quality, schematized travel-time zone for a starting location given a start time and maximum travel times. It fulfills the following requirements:

(1) MultiModalNetworks: The travel-time map is based on a multi-modal transportation network, considering public transport and pedestrian movement.
(2) LowVisualComplexity: The travel-time map has low visual complexity without ramified structures.
(3) SeparationProperty: The travel-time map is formally correct and accurately represents reachable and unreachable locations in the road network.
(4) Schematization: The travel-time map is schematized such that the shape of each time zone is restricted to a fixed number of edge directions.

We have developed our method, particularly for users roaming through a city, e.g., tourists exploring different sightseeing spots distributed all over the city. In that scenario, typically, the user gets around on foot and uses public transportation to bridge larger distances. Hence, to determine the reachable part of the road network in Step 1, we consider both public transportation and the parts of the road network that are accessible by foot. Taking public transportation into account poses additional challenges when creating travel-time maps. Firstly, when determining the reachable part of the road network in Step 1, the underlying routing component must support integrated path-finding in the road and public transportation network. To that end, we utilize the time-expanded model introduced by Pyrga et al. (2008) for routing in public transportation networks with timetables and connect it with the road network. This enables us to compute realistic routes whose travel times depend on the current timetable. Secondly, a time zone is not necessarily a single component but may consist of multiple components, each possibly containing holes representing unreachable parts. This needs to be taken into account in Step 2.

Additionally, we show how to embed this algorithm in a holistic framework for creating travel-time maps with multiple, correctly nested travel-time zones. To that end, we introduce the following extensions.

1. *Multiple travel times.* Time zones of different travel times are overlaid. The algorithm is adapted such that the time zones are correctly nested to avoid cluttered maps.
2. *Clipping.* A post-processing step incorporates natural barriers, such as rivers, that cannot be overcome by foot. Overlaying them with time zones may lead to the wrong impression that a region is easily accessible by foot; see Figure 9.4a. We, therefore, clip time zones to natural barriers; see Figure 9.4b.
3. *Polishing.* To enhance the visualization, a *morphological closing operation* borrowed from image processing is adapted on graph structures and applied to resolve small artifacts.
4. *Multiple zoom levels.* In a generalization step, the time zone is adapted to multiple zoom levels, showing different levels of detail.

The chapter is organized as follows. We discuss related work in Section 9.2. In Section 9.3, we give a high-level overview of the components of our approach, which we explain in greater detail in Section 9.3.1–9.3.3. We describe possible extensions in Section 9.3.4. In Section 9.4, we present our experiments on real-world data and their evaluation. An implementation of the approach is available as open source at https://github.com/GeoinfoBonn/travel-time-maps.

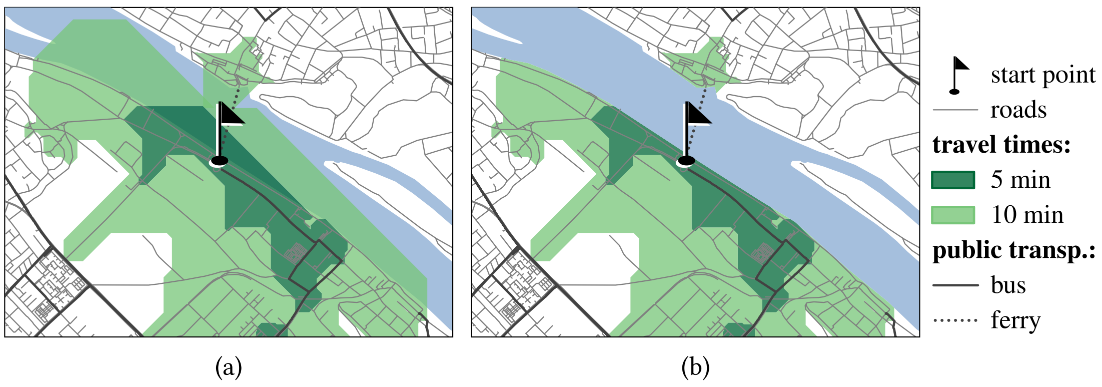


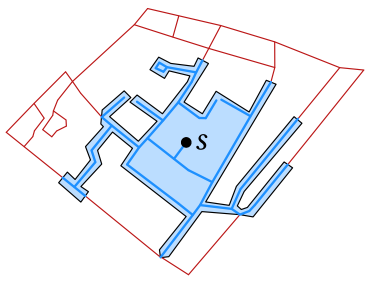
Figure 9.4: Incorporating natural barriers. (a) Time zones without cutting natural barriers. (b) Time zones after cutting natural barrier.

9.2 Related work

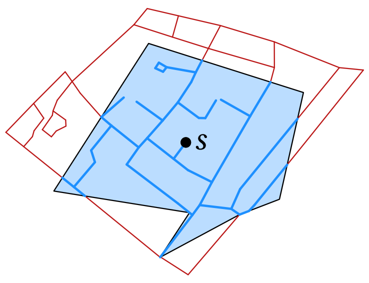
Two basic algorithmic problems lie at the core of creating travel-time maps. The first is about determining the reachable part of the transportation network. The result is a possibly disconnected set of components of the network that describe the reachable subnetwork for a given travel time. For our scenario, it must fulfill the requirement of supporting MultiModalNetworks. The second algorithmic problem is then about the visualization of the obtained components. This problem is related to our requirements of LowVisualComplexity, SeparationProperty, and Schematization.

For computing the reachable part of the transportation network, previous work has considered various types of networks and routing algorithms specially engineered for solving this task in real-time applications. Bauer et al. (2008) present a simple routing algorithm that considers both a road network and a bus network. Gamper et al. (2011) proposed a formal definition, which describes an isochrone as the (possibly disconnected) subgraph of a transportation network that can be reached from a given starting point in a specific time. Based on Dijkstra's algorithm for finding shortest paths in graphs, they presented an algorithm for computing isochrones in MultiModalNetworks. Gamper et al. (2012) and Krismer et al. (2017) improved that algorithm with respect to its running time and memory consumption. Moreover, Krismer et al. (2014) considered the computation of multiple isochrones for different travel times but the same starting point. Baum et al. (2019) adapted customized route planning to computing isochrones in dynamic road networks. They specially engineered their algorithms to deploy them in interactive scenarios, taking large networks, i.e., spanning entire continents, into account.

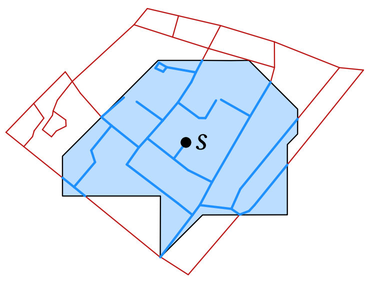
Less research has been conducted on the visualization of isochrones so far. O'Sullivan et al. (2000) proposed an approach deployed in a geoinformation system for visualizing isochrones that consider public transportation. They suggested computing for each station the remaining walking time and visualizing the possible pedestrian movement either by concentric circles around the stations or more complex regions modeling the area that is accessible on foot. These regions are joined, and inaccessible parts are cut out. Still, their approach does not enforce the



(a) outline of network



(b) minimal bends



(c) schematic

Figure 9.5: Sketches of different methods for creating travel-time maps.

SeparationProperty as the concentric circles can overlap unreachable areas. Krajzewicz and Heinrichs (2016) split the map into cells and, for each cell, computed the required travel time for a given starting point and multi-modal transportation. They used this information to aggregate and color the cells regarding the required travel time. However, based on the cells, this easily yields structures with branched outgrowths in the visualization. Marciuska and Gamper (2010) presented two approaches for visualizing isochrones in road networks. The first creates a buffer with a user-specific size around each reachable edge in the road network, and the second (see Figure 9.5a) constructs a single polygon enclosing the reachable part, which is then enlarged by buffering. Neither of the two approaches enforces the SeparationProperty, but when choosing a small buffer size, the time zones only contain a small number of unreachable roads. Baum et al. (2018) presented an approach for visualizing isochrones in road networks with a strong focus on algorithm engineering. They represented the isochrones by polygons that minimize the number of bends and satisfy the SeparationProperty; see Figure 9.5b. Their approach consists of the following steps:

1. Determine the reachable and unreachable parts of the road network.
2. Planarize the road network resolving bridges and tunnels.
3. Determine the faces in the planarization that separate the reachable parts from the unreachable parts. These faces are merged into one face, separating reachable and unreachable regions.
4. For each reachable region, compute a polygon that encloses that region and separates the unreachable from the reachable part. All of these polygons combined form the time zone.

In our approach, we apply similar steps in a slightly different order. We planarize the road network only once and reuse this information for multiple queries. Further, we consider multi-modal transportation and represent the isochrones by schematized polygons; see Figure 9.5c. An overview of related work for creating travel-time maps is given in Table 9.1 by categorizing the methods towards fulfilling the requirements introduced at the beginning of this chapter.

Schematization has also been investigated in other contexts. Most prominently, schematization is used to represent transit networks such as metro systems. The interested reader is referred to surveys on automated layout methods for transit maps by Nöllenburg (2014) and Wu et al. (2019, 2020). Moreover, Buchin et al. (2016) studied the automated simplification and

	MultiModal Networks	*LowVisual Complexity*	*Separation Property*	*Schema-tization*
Melbourne (1910)	no	no	no	circles
O'Sullivan et al. (2000)	yes	no	no	no
Bauer et al. (2008)	yes	-	yes	-
Marciuska and Gamper (2010)	no	no	no	no
Krajzewicz and Heinrichs (2016)	yes	no	no	no
Baum et al. (2018)	no	min. bends	yes	no
our approach	yes	min. bends	yes	octilinear

Table 9.1: Overview of different methods for creating isochrones towards fulfilling our requirements.

schematization of territorial outlines preserving area and topology constraints. To capture the shape of arbitrary polygons, Bouts et al. (2016a) presented a mapping to simple grid polygons based on Hausdorff or Fréchet distance. Recently, Bonerath et al. (2019) presented an approach for visualizing points annotated with time stamps based on schematized, octilinear α-shapes. In order to construct the octilinear polygons, they create a shortcut graph, which represents all candidate edges of the time zone. Similar graph structures are used for line simplification (Imai and Iri, 1988) and the simplification of footprints of buildings (Haunert and Wolff, 2010).

Empirical evaluations of LowVisualComplexity have been conducted in the context of graph drawings. Purchase (2000) identifies the number of bends and the number of crossings in a graph drawing as the most significant factors for the usability of graph drawings. Further research on edge crossings showed that the number of crossings should be minimized and their crossing angle should be maximized (Huang et al., 2008). Ware et al. (2002) highlight the importance of path continuity for shortest path perception. We incorporate these findings into our approach by minimizing the number of bends in our time zones and using octilinear polygons with a fixed smallest opening angle of 45°, thus improving path continuity.

We provide travel-time maps at different zoom levels, selecting the most important roads in small-scale maps based on their attributes. Other methods for selective omission in road networks can be applied. Brewer et al. (2013) presented an automated approach for thinning road networks by removing features. For network generalization, Chimani et al. (2014) introduced an approach that successively deletes the edges, preserving the graph's connectivity. Zhou and Li (2016) proposed a method for empirically determining geometric parameters to decide which roads should be retained or eliminated at a specific scale.

9.3 Methodology

We present a workflow for creating travel-time maps consisting of three components, which can be enhanced by two extensions; see Figure 9.6. The input is public transportation data consisting of timetable information and station coordinates, a road network, an origin within the road network, the start time of the query, and a set T of travel times.

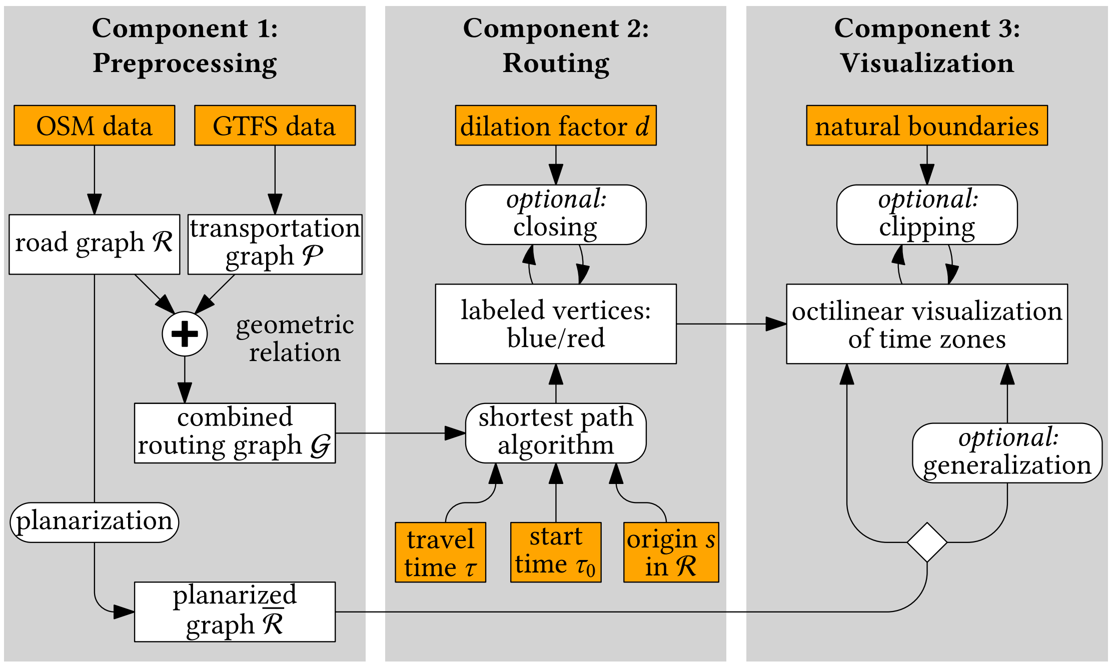

Component 1: Preprocessing
OSM data
GTFS data
road graph $\mathcal{R}$
transportation graph $\mathcal{P}$
geometric relation
combined routing graph $\mathcal{G}$
planarization
planarized graph $\overline{\mathcal{R}}$
Component 2: Routing
dilation factor d
optional: closing
labeled vertices: blue/red
shortest path algorithm
travel time τ
start time τ_0
origin s in $\mathcal{R}$
Component 3: Visualization
natural boundaries
optional: clipping
octilinear visualization of time zones
optional: generalization

Figure 9.6: Illustration of workflow. The Preprocessing Component creates different graph structures used in the Routing Component and the Visualization Component. The Routing Component marks all reachable vertices blue and all unreachable vertices red. The Visualization Component creates a time zone for each travel time.

Preprocessing component. The input data is prepared by preprocessing graph structures that can be reused for multiple travel-time maps with different origins. Hence, we execute this component only once in advance, investing some computation time to accelerate queries made by the user. In our experiments, we used General Transit Feed Specification (GTFS) data for public transportation and OpenStreetMap[1] (OSM) data for modeling the road network; other data sources may also be integrated. We build a graph $\mathcal{P}$ modeling the public transportation by applying the time-expanded model by Pyrga et al. (2008). Further, we represent the road network as a geometric graph $\mathcal{R}$. We merge both graphs to one graph $\mathcal{G}$ used in Component 2 for routing and determining all reachable vertices of $\mathcal{R}$. Further, we *planarize* $\mathcal{R}$ to obtain a graph $\overline{\mathcal{R}}$ whose embedding is planar, i.e., there are no crossings between two edges. The planarization is

[1] www.openstreetmap.org

detailed in Section 9.3.1. We only use $\overline{\mathcal{R}}$ in Component 3 to compute the visualization polygons, but not for routing in Component 2. The details are discussed in Section 9.3.1.

Routing component. Using a shortest path algorithm on $\mathcal{G}$, we compute the locations in $\mathcal{R}$ that are reachable from a given starting point s and start time τ_0 within travel time $\tau \in T$. We subdivide each edge e that is only partly reachable from s by an additional vertex at the point with time distance τ from s. Hence, afterward, the reachability of $\mathcal{R}$ corresponds to a coloring of the edges: all reachable edges are *blue*, and all unreachable edges are *red*. The details are in Section 9.3.2.

Visualization component. This component creates the time zones of the travel-time map. It expects the planarized road network $\overline{\mathcal{R}}$ and for each travel time $\tau \in T$, the edge coloring created by the previous component. For each travel time, it creates a set of schematic polygons containing exactly all reachable edges in $\overline{\mathcal{R}}$. In case of multiple travel times, the approach considers the creation of the time zones in an integrated way. The details are in Section 9.3.3.

Extensions. Three optional extensions can enhance the workflow. These aim to further simplify the shape of the created time zones by removing small holes and considering natural boundaries and zoom levels. The details are explained in Section 9.3.4.

9.3.1 Preprocessing component

This section explains how the input data is preprocessed, obtaining the graph structures used in the Routing Component and the Visualization Component.

Road network. We model the road network as a geometric graph $\mathcal{R} = (V_{\mathcal{R}}, E_{\mathcal{R}})$, i.e., each vertex corresponds to a point, and each edge corresponds to a line segment whose endpoints are the incident vertices of the edge. We assume that the course of a road between two junctions is described by a path consisting of degree-2 vertices. In order to accelerate routing, we can contract degree-2 vertices, but for the Visualization Component, we also need to consider these vertices to maintain the shape of the network. As we consider pedestrians navigating through the city in a combination of public transportation and walking, we only include paths that are accessible by pedestrians; in particular, we exclude all highways and speedways. Further, we annotate each edge $e \in E_{\mathcal{R}}$ with the time necessary to traverse it; in our experiments, we assumed a constant walking speed of 5 km/h. Note, however, that we make no assumptions outside the road network.

Due to tunnels and bridges, the given embedding of the road graph $\mathcal{R}$ is not necessarily plane, i.e., there may be edges that cross each other. However, we do not need this additional information for the Visualization Component. We can make the embedding plane by subdividing the edges with additional vertices at their intersections, as done in (Baum et al., 2018). The result is a plane graph $\overline{\mathcal{R}}$, which we utilize for creating the time zones in the Visualization Component. We note that we use the road graph $\mathcal{R}$ still containing all non-planarized bridges and tunnels for routing.

Public transportation network. We assume that the timetable for the public transportation network is given. It is described by a set S of *stations* and a set Γ of *trips*. Each trip $t \in \Gamma$ is a sequence of stations served by a vehicle in chronological order. Hence, we describe t as a sequence of *stops* such that each stop is defined by a station $\sigma \in S$ and two times, namely the arrival and departure times of the bus or train. Based on this timetable we construct a graph $\mathcal{P} = (V_{\mathcal{P}}, E_{\mathcal{P}})$ following the time-expanded model for timetable information presented by Pyrga et al. (2008); as we introduce some problem-specific adaptions we describe this model for the convenience of the reader; see also Figure 9.7.

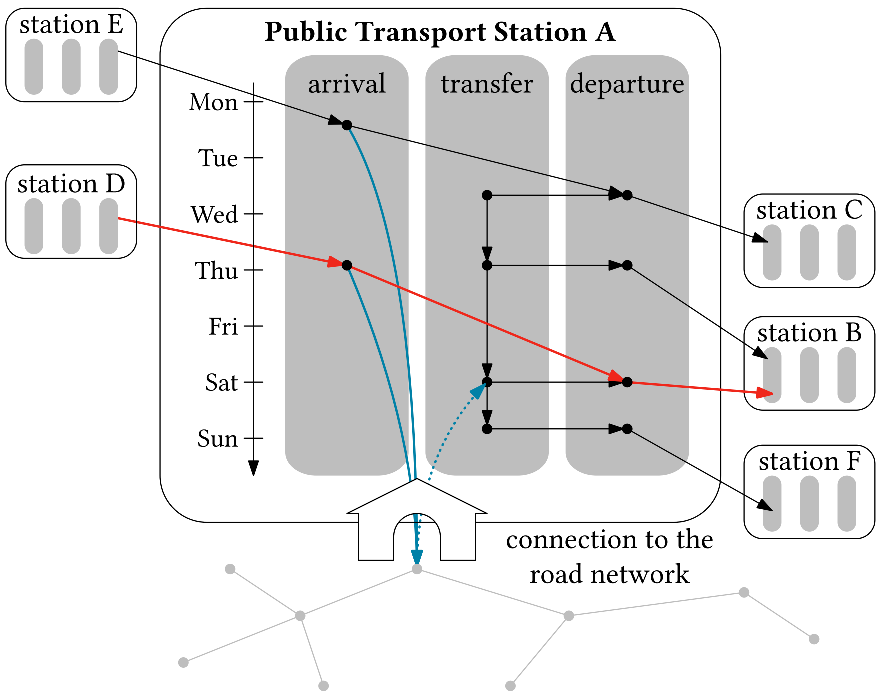


Figure 9.7: Time-expanded model for an example station A. Four trips occur at the station: two starting in station A and two passing through. Transfer vertices model the change of a connection. The connection to the road network is marked blue.

For each stop h of each trip, we introduce two vertices u and v that represent the arrival and the departure of the means of transportation at a certain station σ, respectively. We call u the *arrival vertex* and v the *departure vertex* of the stop and annotate u with its *arrival time* τ_u and v with its *departure time* τ_v. Further, we insert a directed edge from u to v. Per trip, each pair of consecutive stops h, h' are concatenated by adding an edge from the departure vertex of h to the arrival vertex of h'.

In order to model transfers from one trip to another, we introduce a *transfer vertex* w for each stop h of each trip. We annotate w with a *transfer time* τ_w equal to the departure time of h. We introduce a directed edge from w to v, modeling the access to a trip. Finally, we connect the transfer vertices of the same station with each other. To that end, let H_σ be the set of all stops at a station σ. We sort H_σ in increasing order with respect to their transfer times. We introduce directed edges between each pair of consecutive transfers in H_σ. These edges model necessary waiting times at σ.

Further, each edge is annotated with the time difference between its target and source vertices. This difference models the cost of traveling along this edge. The model allows us to find the best journey through the public transportation network, e.g., with Dijkstra's algorithm. Further, it supports minimal transfer times and a cyclic repetition of the timetable; for more details, see Pyrga et al. (2008).

Joining road graph and public transportation graph. For the Routing Component, we join the road graph $\mathcal{R}$ and the public transportation graph $\mathcal{P}$ to one graph $\mathcal{G}$ as follows. For each station σ we use its geographic location to identify the closest vertex u in $\mathcal{R}$; we call u an *access point* of the public transportation system. To keep the experiments simple we merely used the geographic locations of the stations. However, more sophisticated approaches based on the entrances of the stations could be used to make the routing more accurate. To model that the user leaves the public transportation system, we connect each arrival vertex v of σ with the access point u via a directed edge from v to u. As each trip has a pre-defined departure time at each station, we cannot simply insert edges between u and all transfer vertices of σ modeling that the user enters the public transportation system. Instead, we do this on demand when computing the shortest journey in $\mathcal{G}$ in the following component.

9.3.2 Routing component

We use the graph $\mathcal{G}$ defined in the previous component to determine all reachable vertices in the road network. For a starting point s in the road network of $\mathcal{G}$ we use Dijkstra's algorithm to determine all vertices that are reachable in time τ. From a technical point of view, the algorithm determines for each vertex u the shortest journey time necessary to reach u. Thus, when routing in the part of $\mathcal{G}$ representing the road network, we determine for each vertex u at which time we arrive earliest, assuming that we are given a start time for the journey, e.g., Monday 2 pm. We call this the *earliest arrival time* at u. Further, each time when the algorithm reaches an access point u of a station σ in the public transportation system, it temporarily introduces an edge to the transfer vertex of σ whose transfer time suits the earliest arrival time of u best; see the dotted blue arc in Figure 9.7. With this, we obtain for each vertex of $\mathcal{G}$ its earliest arrival time. Using a standard termination criterion, we let Dijkstra's algorithm only consider vertices that are reachable in time τ such that we do not consider the entire network. We mark all reachable vertices blue and all others red. Further, for each edge e that is incident to both a blue and a red vertex, we determine the location on e that is still reachable from s in time τ; see Figure 9.8a. We subdivide e at this location by a blue *dummy vertex*. A special case occurs when the sum of the remaining distances at two adjacent, reachable vertices u and v is smaller than the length of edge $e = \{u, v\}$, see Figure 9.8b. In this case, e is subdivided by two additional dummy vertices, with the middle part being colored red. We further transfer the coloring and the newly inserted vertices into the planarized graph $\overline{\mathcal{R}}$, additionally coloring the vertices that have been introduced to planarize $\mathcal{R}$; see Figure 9.8c. Each such vertex becomes blue if it subdivides an edge e in $\mathcal{R}$ that is only incident to blue vertices; otherwise, it becomes red. Altogether, we obtain a coloring of $\overline{\mathcal{R}}$ defining all vertices either blue or red. Due to the insertion of the dummy vertices, this vertex coloring induces an edge coloring: edges that are incident to two reachable vertices are also reachable, and all other edges are unreachable.

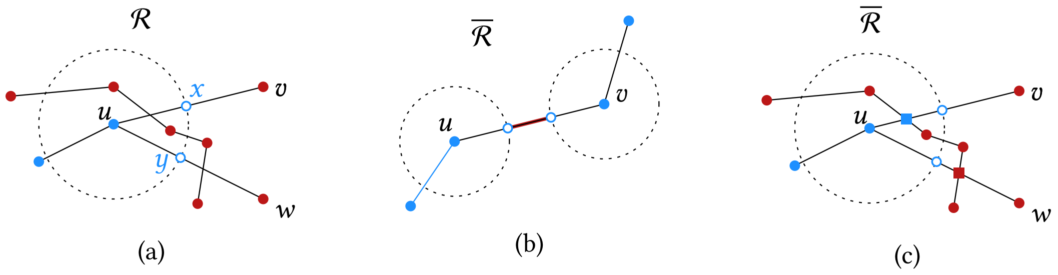


Figure 9.8: The reachability is modeled by subdividing edges. (a) The vertex x (resp. y) subdivides the edge (u, v) (resp. (u, w)) at the last reachable position. (b) Special case: The vertices u and v are are reachable from different directions. (c) The coloring of $\mathcal{R}$ is transferred to $\overline{\mathcal{R}}$, and the vertices (squares) introduced for the planarization are colored.

9.3.3 Visualization component

In this section, we describe how to compute the time zones of the travel-time map based on the coloring of the previous step (see Section 9.3.2). We first explain this for a single travel time and then for multiple travel times.

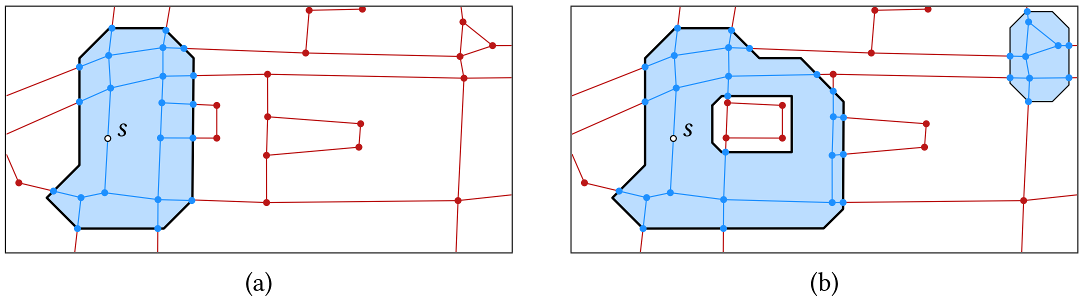


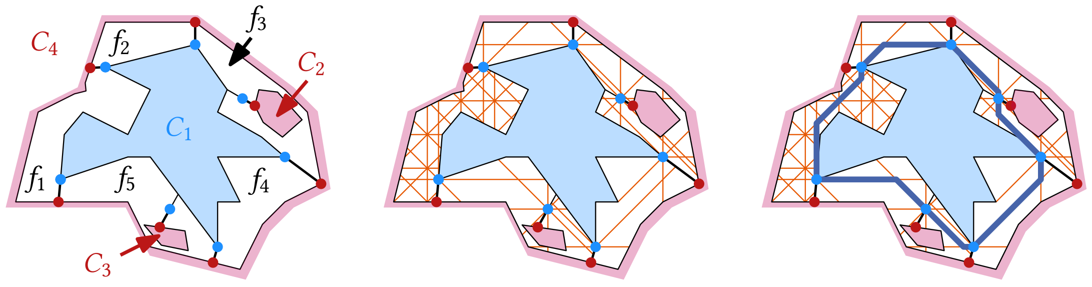
Figure 9.9: Reachabilities for a single travel time. (a) The blue vertices and edges form a connected component. (b) The blue and red vertices and edges form multiple connected components.

Single travel time

In this section, we assume a single travel time. Thus, we are given the planarized road network $\overline{\mathcal{R}}$ and a coloring such that each edge and each vertex are either blue (reachable) or red (unreachable). Further, we assume that the outer face only consists of red vertices but no blue vertices; we can always insert an unreachable bounding box and connect it to the road network.

We first observe that $\overline{\mathcal{R}}$ has faces that have both red and blue edges; see Figure 9.9. We call an edge that is incident to both a blue and a red vertex a *gate*. Further, we call the reachable vertex of a gate its *port*. Removing the gates from $\overline{\mathcal{R}}$ decomposes the graph into a set of components $C_1, \dots, C_\ell$ such that each component is either completely blue or red. We call them the *colored components* of $\overline{\mathcal{R}}$.

Base case. We first consider that $C_2, \dots, C_\ell$ have the same color, but C_1 has the opposite color; see Figure 9.10a. We describe a procedure that creates a single octilinear polygon such that C_1 lies inside and $C_2, \dots, C_\ell$ lie outside of the polygon. Without loss of generality, we assume that C_1 is blue and the other components are red. Let $v_1, \dots, v_k$ be the ports of $\overline{\mathcal{R}}$ on the boundary of C_1 in the order in which we encounter them when going along the boundary of C_1; we choose an arbitrary starting point and define $v_{k+1} := v_1$. We observe that any two consecutive ports v_i, v_{i+1} are incident to the same face of $\overline{\mathcal{R}}$, which we denote by f_i.



(a) (b) (c)

Figure 9.10: Creating an octilinear polygon that encloses the component C_1 (blue) of all reachable vertices and edges. (a) The faces $f_1, \dots, f_5$ surround the component C_1. (b) Each face is subdivided by an octilinear grid (Step 1). Furthermore, these grids are connected to one large grid G that is split by the port between f_1 and f_5 (Step 2). (c) An octilinear polygon is constructed by computing a bend-minimal path through G.

The base case consists of three steps; see Figure 9.10. In the first step, we create for each pair v_i, v_{i+1} of consecutive ports an octilinear grid G_i contained in f_i. In the second step, we fuse these grids to one large grid G. In the final step, we use G to determine an octilinear polygon by finding an optimal path through G.

Step 1. For each pair v_i, v_{i+1} of consecutive ports with $1 \leq i \leq k$ we first compute an octilinear grid G_i that is contained in f_i and connects v_i and v_{i+1}; see Figure 9.11. To that end, we shoot from both ports octilinear rays and compute the line segment arrangement $\mathcal{L}$ of these rays restricted to the face f_i; see Figure 9.11a. If $\mathcal{L}$ forms one component, we use it as grid G_i. Otherwise, we refine $\mathcal{L}$ as follows. We uniformly subdivide the bounding box of f_i by further vertices from which we shoot additional octilinear rays; see Figure 9.11b. We insert them into $\mathcal{L}$, restricting them to f_i. We call the number of vertices on the bounding box the *degree of refinement d*. We double the degree of refinement until $\mathcal{L}$ is connected or a threshold $d_{\max}$ is exceeded; see Figure 9.11c. In the latter case, we also insert the boundary of f_i into $\mathcal{L}$ to guarantee that $\mathcal{L}$ is connected. Later on, when creating the octilinear polygon, we only use the boundary edges of f_i if necessary.

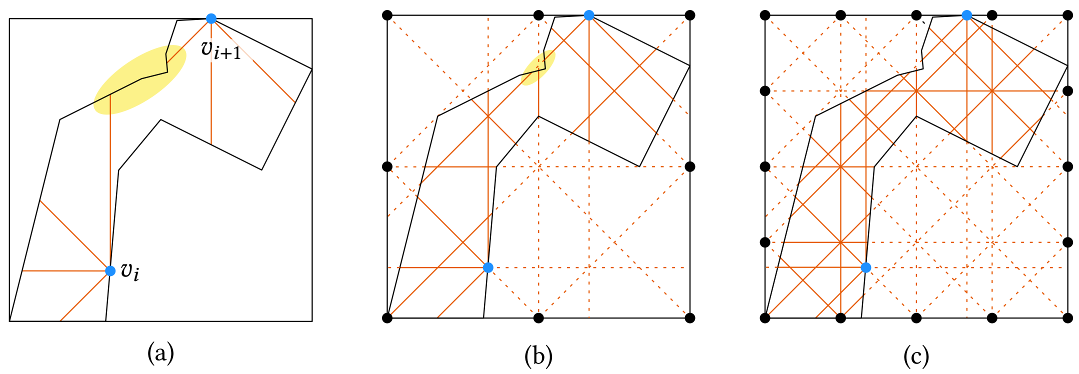


Figure 9.11: The face f_1 of the example shown in Figure 9.10 is subdivided by octilinear rays based on vertices on the bounding box. (a) Shooting octilinear rays from v_i and v_{i+1} does not yield a connected line arrangement (see yellow highlight). (b)–(c) The bounding box of f_1 is successively refined by vertices shooting octilinear rays until they connect v_i and v_{i+1}.

Step 2. In the following, we interpret each grid G_i as a geometric graph such that the grid points are the vertices of the graph and the segments connecting the grid points are the edges of the graph. We union these graphs to one large graph G. More precisely, G is the graph that contains all vertices and edges of the grids $G_1, \ldots, G_k$. In particular, each port v_i is represented by two vertices x_i and y_i in G such that x_i stems from G_{i-1} and y_i stems from G_i; for $i = 1$ we define $x_1 = v_k$. We connect two grids G_{i-1} and G_i in G by introducing the directed edge (x_i, y_i) in G for $2 \leq i \leq k$. An example is shown in Figure 9.12a.

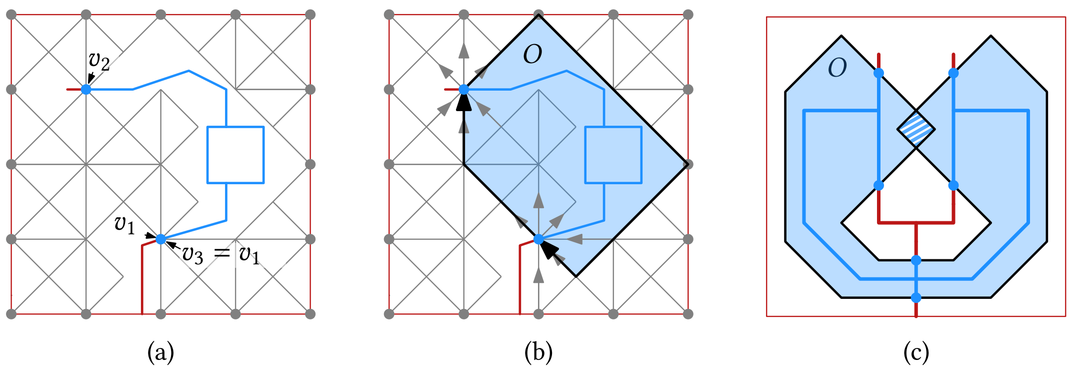


Figure 9.12: Routing step. (a)–(b) The polygon O must not contain unreachable parts. For the routing step the edges incident to ports are directed accordingly. (c) The path O is not necessarily simple.

Step 3. In the following let $s := y_1$ and $t := x_1$. We compute a path P from s to t in G such that P has the minimum number of bends, i.e., there is no other path from s to t that has fewer bends; see Figure 9.12b. To that end, we use Dijkstra's algorithm on the linear dual graph of G, allowing us to penalize bends in the cost function. If the choice of P is not unique because there are multiple paths with the same number of bends, we use the geometric length of the path as a tie-breaker, preferring shorter paths. As s and t have the same geometric location, the path P forms an octilinear polygon O. By using directed edges at the ports, we guarantee that P crosses the port so that no unreachable component can be enclosed by O; see Figure 9.12b.

If we cannot find a connected octilinear grid for a face in the second step, we relax the requirement that P is octilinear. In that case, P might also contain edges of the according face. In order to minimize the number of these possibly non-octilinear edges, we penalize them with high costs when searching for P. We note that O is not necessarily simple, as we individually calculate each segment between two ports. This can lead to two segments of the same face crossing each other; see Figure 9.12c. In this exceptional case, we transform O into a simple polygon with holes by joining the overlapping parts. This can change the number of bends in the polygon's outline. We consider this appropriate as we do not search for a global optimum solution towards the number of bends but only an optimal solution in our model.

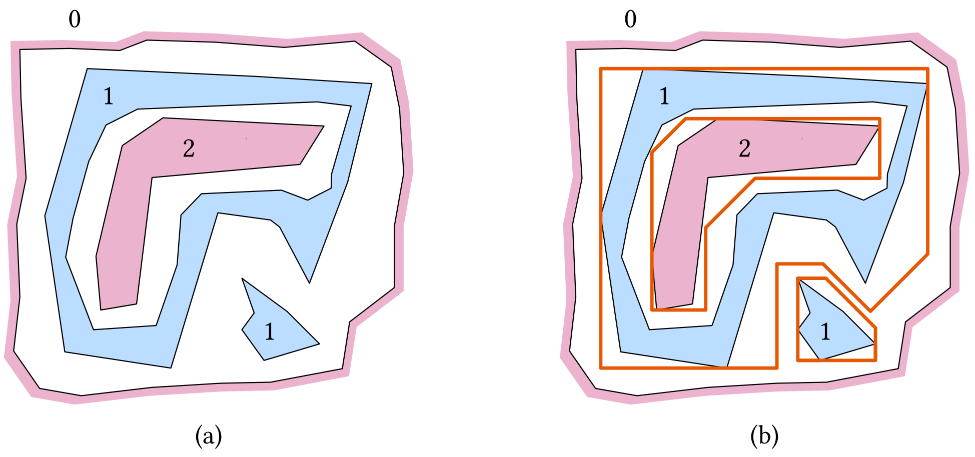


Figure 9.13: The reachable (blue) and unreachable (red) parts of the road graph consist of multiple components. (a) Each component has a level. (b) Each internal component is enclosed by a schematized polygon.

General case. If $\overline{\mathcal{R}}$ has multiple reachable and unreachable colored components, we enclose the components in multiple polygons. To that end, we assign to each component a level that states to which degree it is nested into other components; see Figure 9.13a. The idea is that blue components of an odd level and red components of an even level are enclosed by time zones. To assign the levels, consider the graph $\overline{\mathcal{R}}'$ that we obtain from $\overline{\mathcal{R}}$ by removing its gates. A component in the outer face of $\overline{\mathcal{R}}'$ has level 0. A component C that lies in the internal face of another component C' with level i gets assigned to level $i + 1$. For each blue component with an even level and each red component with an odd level a time zone is computed; see Figure 9.13b. Hence, we obtain a time zone so that blue and red edges and vertices of $\overline{\mathcal{R}}$ are separated. Further, time zones of different levels are either disjoint or nested. However, time zones of the same level might partly intersect. We resolve this by simply computing the union of all time zones enclosing components of the same level. The resulting set of time zones is then this union.

Multiple travel times

For multiple travel times, we compute the time zones of the travel times in decreasing order. However, we cannot simply combine them, as routing through the octilinear grids of the different travel times may result in partly intersecting time zones. We observe that the edges and vertices of $\overline{\mathcal{R}}$ are correctly separated for each travel time, but the map may contain unnecessary overlaps between time zones. Therefore, we proceed as follows. Let $\tau_1, \dots, \tau_\ell$ be the travel times in increasing order, i.e., $\tau_1 < \dots < \tau_\ell$. For τ_ℓ, we compute the set $\mathcal{Z}_\ell$ of time zones as described in Section 9.3.3. For τ_i with $i < \ell$ we slightly adapt the procedure of Section 9.3.3. When computing the grid graph G of a single colored component, we restrict the graph to the union of the time zones in $\mathcal{Z}_{i+1}$. More specifically, we intersect its geometric embedding with the union of the time zones in $\mathcal{Z}_{i+1}$. The result is a smaller grid graph G' that we use for routing and computing the time zone enclosing the corresponding component. Altogether, we obtain a set of time zones for all travel times. We draw them in the order as their area decreases and color each time zone differently.

9.3.4 Extensions

In this section, we briefly describe three extensions that we use to enhance the visualization. The first can be plugged into the Routing Component, and the second and third can be plugged into the Visualization Component.

Closing

Initial experiments showed that the Routing Component typically returns a colored graph that consists of a few large colored components and a variety of small colored components.

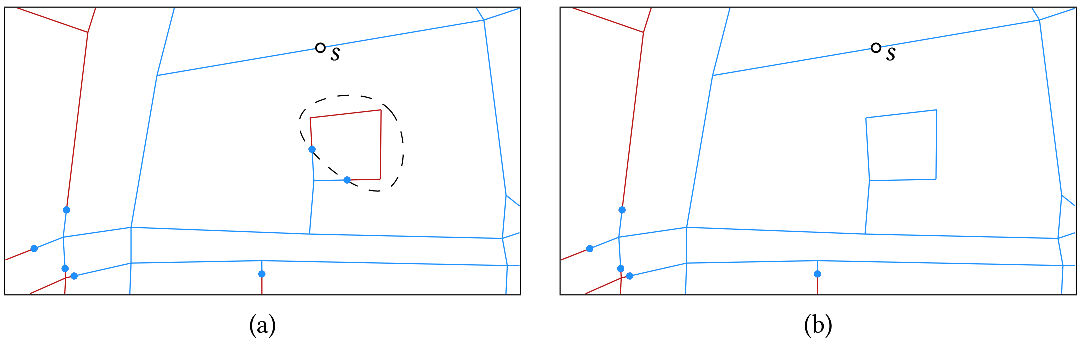


Figure 9.14: Closing operation. (a) A small group of unreachable (red) vertices in the blue area. (b) After the closing operation the hole is filled.

For example, due to the travel time choice, a few vertices and edges may be unreachable while all others in the direct surroundings are reachable; see Figure 9.14. These small unreachable *islands* “rupture” holes into the time zones, looking unpleasant. To improve the visualization, the closing extension removes these holes. A time zone created with this extension thus does not strictly fulfill the separation property anymore! Nonetheless, we deem it reasonable to fill these small holes, as the time zones depend on the assumption that the user has a walking speed of precisely 5km/h, which introduces an exaggerated accuracy. To that end, we utilize the technique of *morphological closing* applied in image processing to suppress small patches of noise (Serra, 1983). The idea is first to extend the area of interest, which removes the holes (dilation), and then to reduce the area to restore the original outer boundaries (erosion).

We adapt this method for our problem as follows. For dilation, we mark every vertex that is reachable in time τ with blue; see also Figure 9.14. Then we use an increased time $\tau_d = \tau \cdot (1+d)$ with $d > 0$ to calculate a buffer region, i.e., we compute all vertices that are reachable in time τ_d and mark all vertices *yellow* that are not already colored blue; we call d the *dilation factor* of the closing operation. All remaining uncolored vertices become red. For erosion, we determine for every yellow vertex adjacent to a red vertex the yellow component that contains that vertex. We color all vertices in that component red. Thus, the original boundaries of the blue components are restored, while the vertices in small holes of the blue components remain yellow. Finally, we color the remaining yellow vertices blue, which closes the holes. The greater d is chosen, the larger the hole closed by this operation.

Travel-time maps at different zoom levels

Depending on the concrete zoom level, digital maps typically show different levels of detail. For example, all roads are shown in large-scale maps, while only the most important roads are displayed to the user in small-scale maps. We implement this concept in travel-time maps as follows. While for large-scale maps, we aim at an outline of the time zone that separates the reachable from the unreachable part of the road network, for small-scale maps, we deem a rough outline preferable, showing the most important features of the time zone. More specifically, we compute all reachable and unreachable parts of the road network once at the beginning. When creating the time zone for a specific zoom level, we remove all roads not displayed for this one and construct the time zone based on the remaining roads. Hence, the separation property still holds for all drawn roads, but it may be violated for the removed roads. Further, the time zone directly adapts to the current selection of roads.

Figure 9.15 shows an example of a travel-time map for two zoom levels. Two areas with many bends are highlighted in Figure 9.15a. The first shows a graveyard with small footpaths, while the second shows a field path parallel to a residential road. A detailed outline with many bends is necessary in both cases to satisfy the separation property. Figure 9.15b shows the generalized time zones of the same example. The visualization for both marked regions is simplified, and the octilinear structure of the polygons is easily recognizable. The blue area remains almost unchanged in the given example, as no roads have been generalized here. In particular, improving the schematization relies on a good selection of roads to be removed.

Clipping

We observe that time zones may overlay important natural barriers such as rivers and lakes; see Figure 9.4. However, this creates the impression that the region is easily accessible by foot. Furthermore, schematizing the outline of these natural barriers may compromise the “mental map” a user has of the area. Therefore, we clip the time zones by subtracting important natural

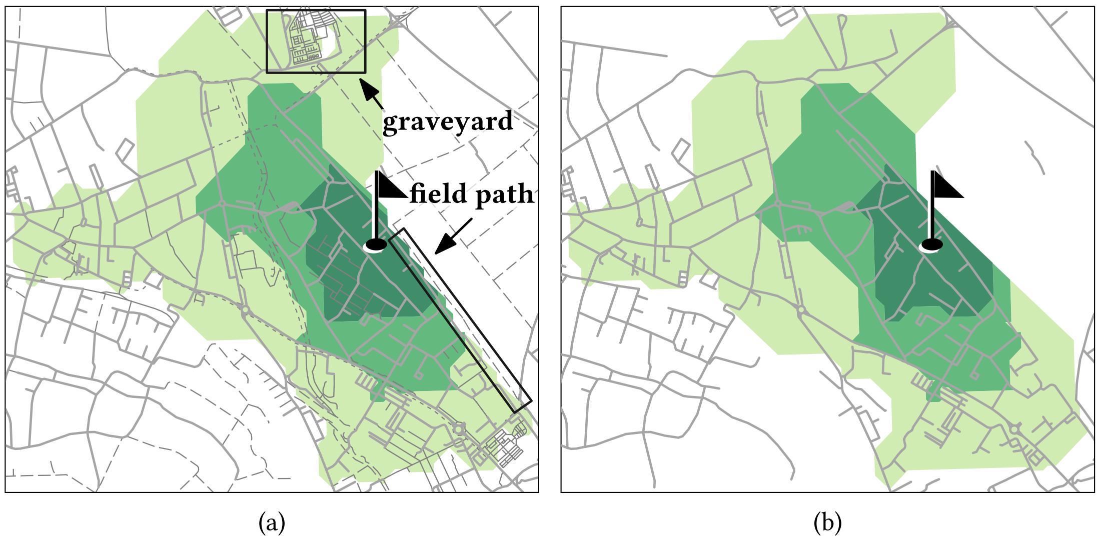


Figure 9.15: Generalization of time zones for different zoom levels. (a) Small streets yield an octilinear polygon with many bends. (b) Minor roads are ignored, resulting in a schematization with fewer bends.

barriers. Sometimes, this creates small scratches of the polygon on the wrong side of the river in areas with no roads. As we do not have information about accessibility to regions without roads, we remove these shreds. With this, we obtain an appearance of the time map that blends in with the natural features of the map.

9.4 Evaluation

In this section, we describe the experiments applied to real-world data. We present the experimental setup in Section 9.4.1. In Section 9.4.2, we discuss results based on a single travel time, comparing them to other visualization approaches for travel-time maps. Finally, in Section 9.4.3, we present a small case study featuring a travel-time map with multiple time zones.

9.4.1 Experimental setup

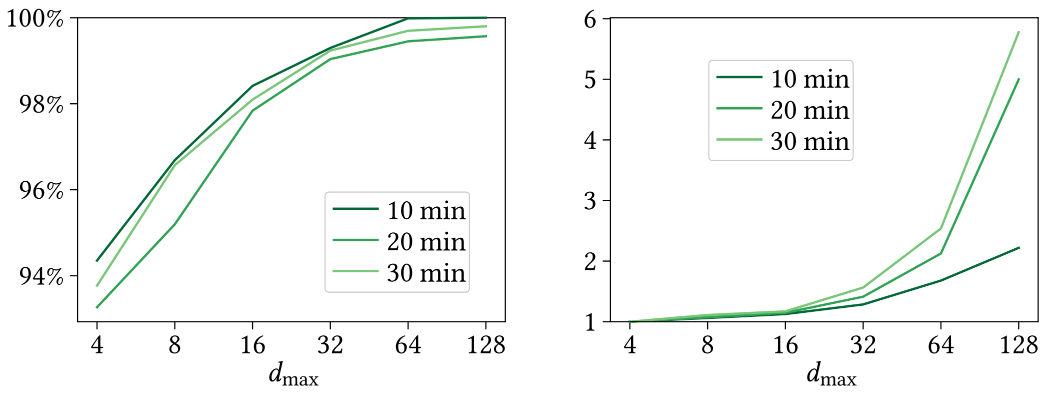
We have created travel-time maps for the metropolitan area of Cologne and Bonn, Germany. The area has a densely developed public transportation network comprising local, regional, and long-distance transport. Moreover, the Rhine River, a major waterway in central Europe, passes through the region and separates the considered region into an eastern and western part. Both parts are connected by eleven bridges and six ferries spread out over 60 kilometers along the river. Thus, the river constitutes a significant barrier to accessibility, making the region interesting for our evaluation.

The input data stems from two sources: for the road network, we use OpenStreetMap (OSM) data[2] extracted by Geofabrik[3]. The extent of the used data is approximately 40 kilometers in latitudes and 60 kilometers in longitudes, centered on the metropolitan area of Cologne and Bonn. The data of the public transportation network is taken from OpenData VRS[4] in the General Transit Feed Specification (GTFS) format and contains timetable information for 2018/2019 for local transportation and regional trains. Further, we inserted data about ferries crossing the Rhine River for our experiments.

We applied the algorithm to 20 different starting locations dispersed around the experimental region for our evaluation. We used Monday at 9:00 a.m. as the starting time and computed travel-time maps for eight travel times (in minutes), namely $T = \{5, 10, 15, 20, 25, 30, 35, 40\}$.

The implementation was written in Java, and the experiments were performed on an AMD EPYC™ 7402P processor clocked at 2.8 GHz with 128 GB RAM. The planarization of the graph took 62 seconds, and creating the remaining travel-time map took 18 seconds on average.

In order to decide on a suitable value for the maximum degree $d_{\max}$ of refinement, we conducted initial experiments. To this end, we calculated time zones for three different travel times with $d_{\max}$ varying from $d_{\max} = 4$ to $d_{\max} = 128$; see Figure 9.16.



(a) octilinearity (b) running time

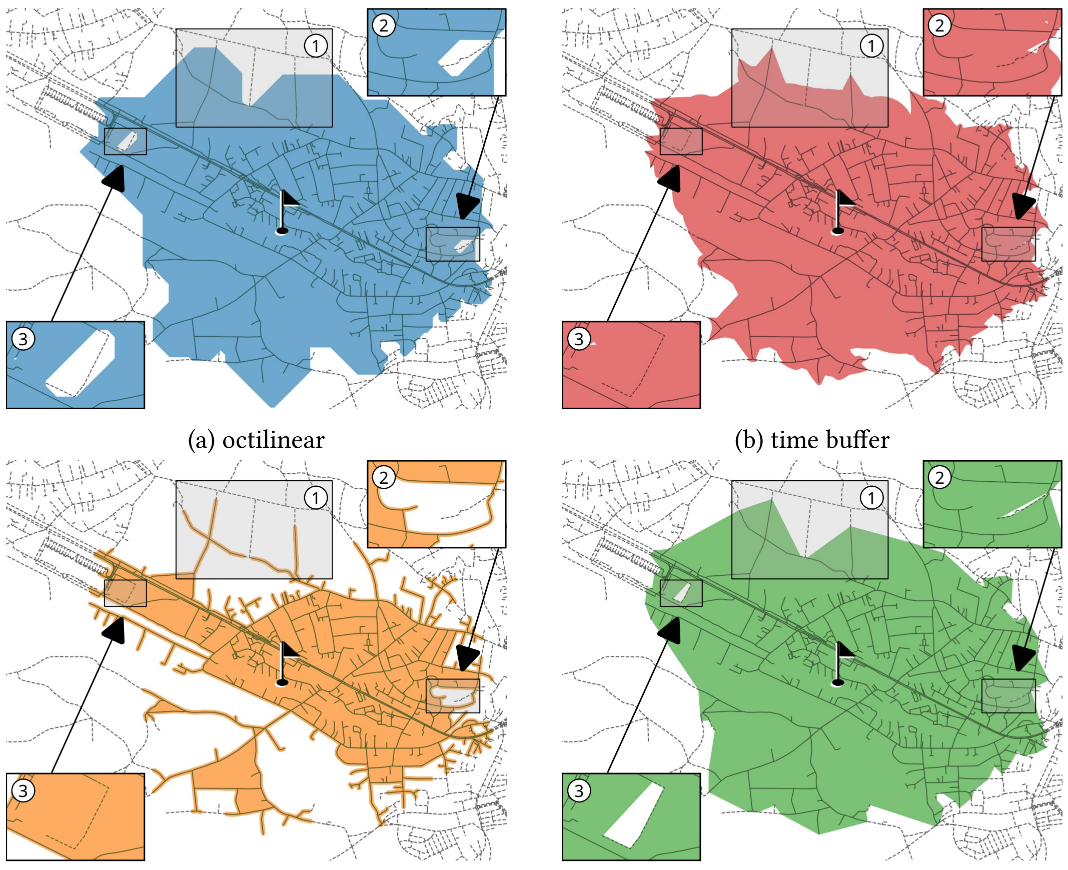
Figure 9.16: Evaluation of the maximum degree of refinement $d_{\max}$. (a) Share of the boundary's length that is visualized octilinearly. (b) Running time compared to the running time using the lowest $d_{\max}$.

The higher the maximum degree of refinement is, the finer is the octilinear grid. Hence, the share of the octilinear edges of the resulting polygon increases with an increasing maximum degree of refinement. For $d_{\max} = 32$ over 99% of all edges are octilinear; see Figure 9.16a. Moreover, for $d_{\max} = 32$ the running time is increased by a factor of at most 1.56 compared to $d_{\max} = 4$; see Figure 9.16b. Thus, for the experiments, we have chosen $d_{\max} = 32$ as a suitable compromise between accuracy and running time.

[2] ©OpenStreetMap contributors [3] www.geofabrik.de [4] www.vrs.de

9.4.2 Result comparison

We evaluate the results of our approach by comparing them with the results of other visualization techniques for travel-time maps. In specific, the techniques we compare our approach to are the *time buffered visualization* inspired by our initial example for the city of Melbourne (see Figure 9.2), the *arboreal visualization* as presented by Marciuska and Gamper (2010) using a very small buffer size to ensure best the separation property and the *minimum-perimeter visualization* which is a slight adaptation of the technique introduced by Baum et al. (2018), minimizing the perimeter instead of the number of bends. However, we note that we are using our implementations of the respective works, and results can differ from those generated by the original solution. We discuss the results concerning their basic properties and visual complexity. As simple measures for the visual complexity of a polygon, we use its perimeter, area, the type of angles (acute angle or obtuse angle), and the directions of its edges. These measures are a common tool to assess the complexity of geometric shapes (Chen and Sundaram, 2005).



(a) octilinear

(b) time buffer

(c) arboreal

(d) minimum perimeter

Figure 9.17: Single time zone for τ = 15 min. for different visualization types. ① and ③ show areas where the separation property is violated in (b) and (c). ② highlights a hole, which is hard to recognize using some visualizations such as (d).

Octilinear visualization vs time buffered visualization. First, we compare the octilinear visualizations with *time buffered visualizations.* At the core of this visualization technique is a constant off-road speed, which is assumed to be smaller than the walking speed on roads. For each point of the road network (no matter whether it is a vertex or a point on an edge), the remaining travel time at this point is visualized by a buffer computed utilizing the off-road speed; see Figure 9.17b. Consequently, the separation property is not enforced by this visualization technique. While reachable parts of the road network are guaranteed to be contained in the time zone, also unreachable parts may be covered by the zone, violating our requirement SeparationProperty; as seen in Figure 9.3a. Another example is shown in Figure 9.17b ① and ②, where the zone covers parts of the road network that are not reachable. Similar examples are easily found all along the zone's boundary. For the given example, 72 vertices of the road network are wrongly represented as reachable. In contrast, as our algorithm requires, the octilinear time zone shown in Figure 9.17a precisely separates the roads. Moreover, the time-buffered visualization tends to become frayed at the boundary as highlighted in Figure 9.17b ②, also violating the LowVisualComplexity.

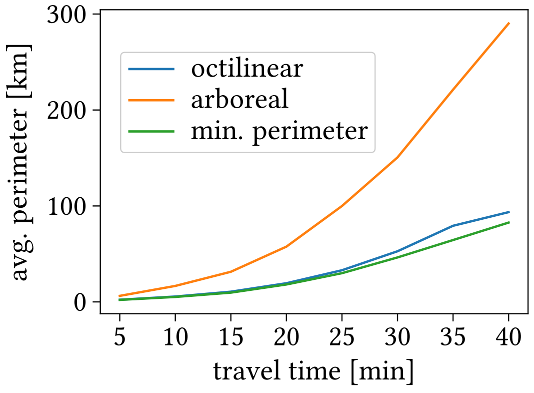
We insist on the SeparationProperty as the main requirement for the travel-time maps. Therefore, we only evaluate approaches that fulfill this requirement in the following.

Octilinear visualization vs arboreal visualization. The second approach visualizes the outer boundary of the reachable zone, which we call *arboreal visualization* due to the ramified shape of the time zones; see Figure 9.17c. We emphasize that this representation gets cluttered along the zone's boundaries since the last reachable roads form tree-like dangles attached to a polygon. Hence, in contrast to our approach, this kind of visualization tends to have substantially larger perimeters; see Figure 9.18a. We observe that, especially for larger travel times, the perimeter differs from our approach, e.g., for a travel time of 30 minutes, the perimeter is increased by a factor of 2.9.

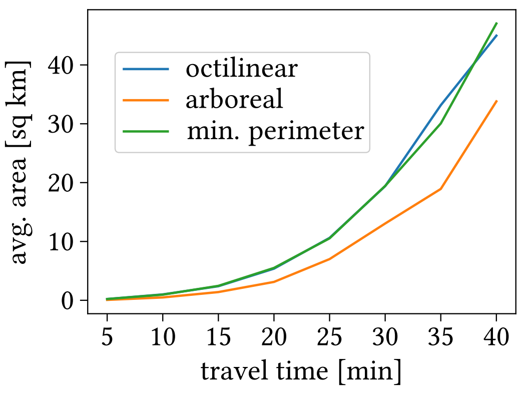
To analyze the types of angles and the direction of the time zones' edges, we create a histogram of the outer angles of the polygons; see Figure 9.19. To that end, we split the angle interval $[0, 2\pi]$ into 16 equi-sized bins such that the i-th bin is defined by the interval $I_i = [\frac{2i-3}{16}\pi, \frac{2i-1}{16}\pi)$ for $1 \leq i \leq 16$. With this definition, the octilinear directions do not form the boundaries of these intervals but split them into equally sized subintervals. For each bin i, we count how many bends of the polygon have an outer angle in I_i.

We observe that the number of acute angles (three backward facing bins, outer angle smaller 33.75° or greater 326.25°) is significantly higher for the arboreal visualization than for the octilinear visualizations; see Figure 9.19b. The share of acute angles at the total amount of angles is 12.2% while it is only 0.07% for our approach. This supports the visual impression of Figure 9.17 that the complexity for arboreal visualizations is much higher than for octilinear visualizations.

Octilinear visualization vs minimum-perimeter visualization. Finally, we compare our approach with visualizations that minimize the perimeter of the travel-time zones; see Figure 9.17d. The approach presented by Baum et al. (2018) yields visually similar results. We first observe that the minimum-perimeter and octilinear visualization lead to similar perimeters;



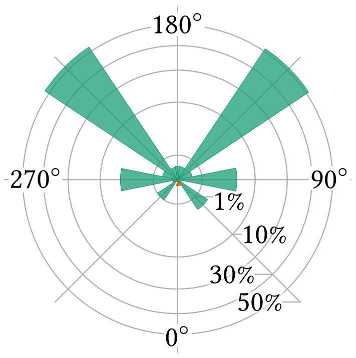
(a) perimeter



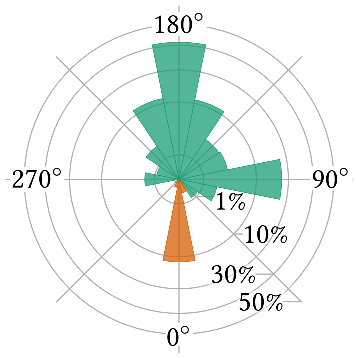
(b) area

Figure 9.18: Perimeter and area for different visualization types.

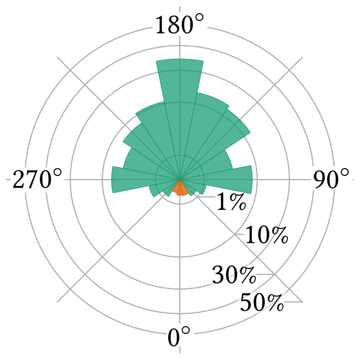
see Figure 9.18a. For our approach, the perimeters are 11% longer on average and 23% at maximum. Moreover, the area covered by the time zones is comparable to each other, indicating relatively small differences between the time zones. However, evaluating the distribution of the outer angles, the effect of the schematization in our approach becomes evident; see Figure 9.19. Over 98.5% of all outer angles lie in bins corresponding to the octilinear directions. In particular, the vast majority of 83.7% lies in the bin representing the two obtuse angles 225° and 135°. In contrast, for the minimum-perimeter visualization, the angles are spread across the whole histogram, leading to a higher visual complexity concerning this criterion.



(a) octilinear



(b) arboreal



(c) minimum perimeter

Figure 9.19: Histogram of the time zone's outer angles, averaged over all starting points for a travel time $\tau = 30$ min. Orange bins relate to acute angles. The quantity of bends per bin is proportional to the bin's area.

Similar to the arboreal visualization, the number of acute angles is larger for the minimum-perimeter visualization than for the octilinear visualization. The share of acute angles is over nine times higher than for our approach. These acute angles can create artifacts in the visualization, where parts of the polygon are hard to spot. In Figure 9.17d, the time zone has a hole at ②, but as it is visualized with a thin polygon having acute angles at its bends, it can hardly be spotted. In contrast, the hole is clearly visible in the octilinear time zone; see Figure 9.17a.

The analysis shows that the octilinear visualization has a lower visual complexity than the minimum-perimeter visualization. This coincides with the visual impression of the maps shown in Figure 9.17. Nonetheless, there can be cases where the octilinear visualization suffers from stair-shaped lines in graph faces with unfavorable geometry. An example is in Figure 9.20. These unfavorable geometries occur for long stretched faces whose main direction is not one of the octilinear directions. In contrast, the minimum-perimeter visualization has one straight edge at this location. Combining the two visualization approaches thus could lead to an even better result. For example, we could check if the number of bends in an octilinear face exceeds a threshold k and then fall back to the minimum-perimeter visualization for this face. An investigation of the combination of both algorithms is future work.

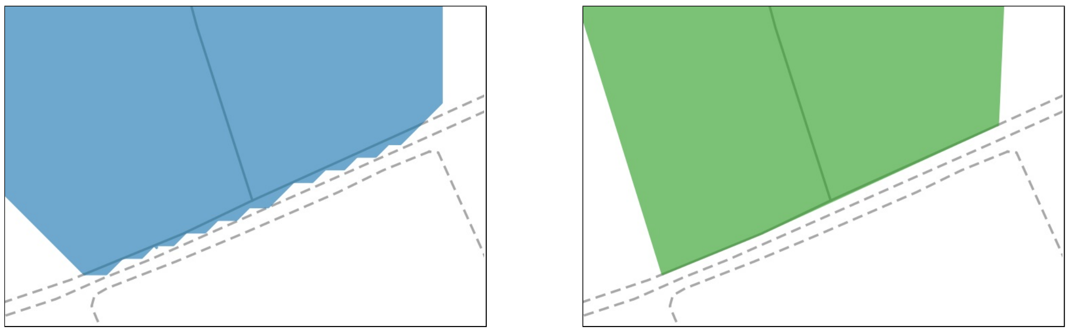

(a) octilinear visualization (b) minimum-perimeter visualization

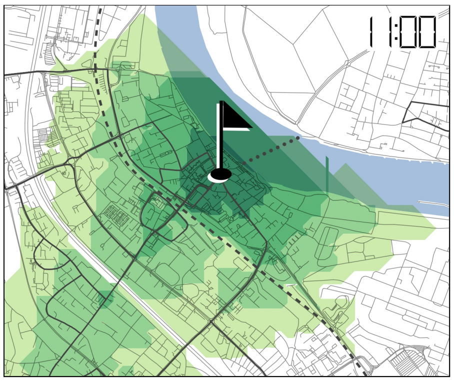
Figure 9.20: In rare cases, the octilinear visualization has a step-like outline, while the minimum-perimeter visualization does not suffer from this.

9.4.3 Case study

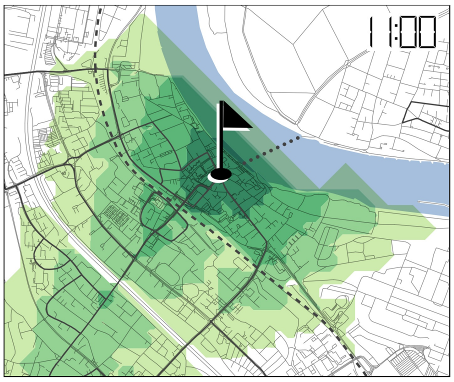
We conclude the evaluation with a small case study highlighting the important steps for creating a travel-time map with multiple time zones; see Figure 9.21 and Figure 9.22. We used a starting location in Wesseling between Cologne and Bonn to obtain four time zones.

Creating each time zone separately might cause inner time zones to overlap the outer ones if their boundaries share the same graph faces; see Figure 9.21a. The outer time zones are used as a spatial limit for the inner ones to eliminate this issue. With this, the inner zones strictly lie inside the outer ones, which leads to a less cluttered map (see Figure 9.21b). Nonetheless, it remains non-intuitive that time zones are differentiated on the Rhine River. To remove this issue, we deploy our clipping extension (see Section 9.3.4), removing all parts of the time zones covering such natural boundaries. As a result, the time zones represent the topography better and thus become more comprehensible; see Figure 9.21c.

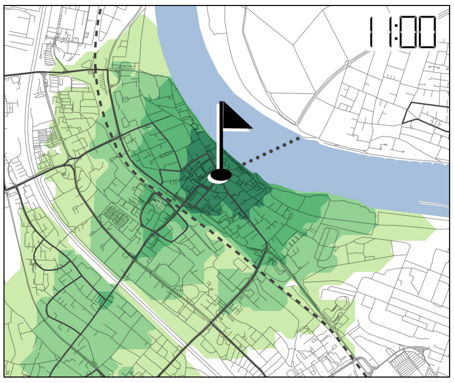
As a last processing step, we apply the closing extension (see Section 9.3.4) to eliminate small holes in the time zones; see Figure 9.21d. We have used a dilation factor of 10%, i.e., a hole is closed if it is entirely covered by a travel time increased by 10% of the original travel



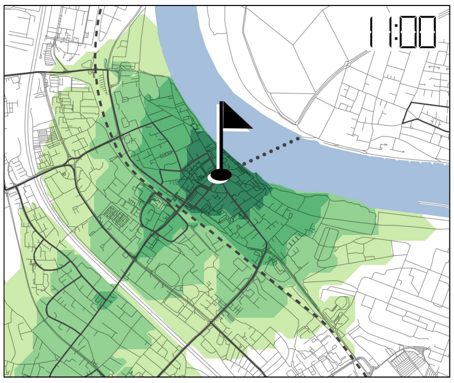
(a)



(b)

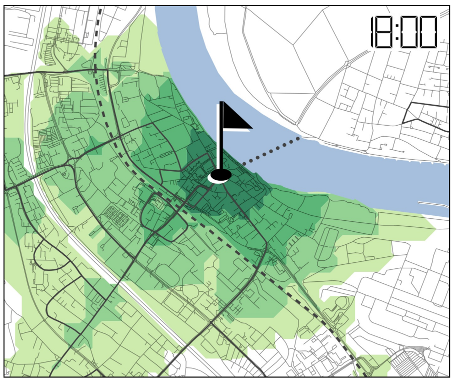


(c)

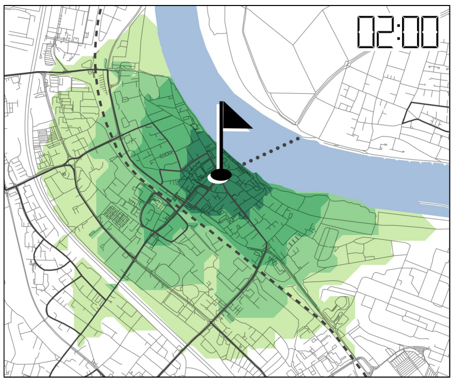


(d)

Figure 9.21: Case study. The travel-time map is created for four time zones with $T = \{5, 10, 15, 20\}$ in minutes and start time Monday 11 a.m. (a) Independently generated time zones. (b) Time zones nested in the next bigger time zone. (c) Clipping to the river. (d) Result after applying the closing operation.



(a)



(b)

Figure 9.22: Continuation of the case study. (a) Travel times with start time Monday, 6pm. (b) Travel times with start time Tuesday, 2am.

time. This further reduces the visual complexity of the resulting travel-time map. In public transportation, this closing operation can be interpreted as a modeled vagueness in the travel-time map due to delays or varying walking speeds.

Thanks to the integration of time-table information of public transportation, the travel-time map can be generated for different starting points, travel times, and starting times. Examples of the variation in travel-time zones depending on the time of the day are shown in Figure 9.22a.

9.5 Isochrones and routing profiles

In Chapter 6, we presented an algorithm to infer the routing preference of a subject (e.g., a cyclist) given their past trajectory recordings. The routing preference is quantified by a numerical balancing factor within a mathematical model. However, this output of the algorithm is hard or even impossible to interpret for non-expert users. To address this issue, we bridge the gap between expert and non-expert users by implementing isochrones as a visual tool for illustrating routing profiles. To that end, we use the visualization component as explained in Section 9.3.3. However, we replace the routing component with one utilizing the routing model of Chapter 6.

Adapted routing component In Chapter 6, the street network is modeled as a directed graph G with two edge cost functions, c_0 and c_1, representing distinct routing criteria. We employ a bicriterial routing model to assess the trade-off between these routing criteria by introducing a personalized edge cost function c_α. This personalized cost ist a linear combination of c_0 and c_1, controlled by a balance factor $\alpha \in [0, 1]$ (see also Equation 6.1):

$$c_\alpha(e) = (1 - \alpha) \cdot c_0(e) + \alpha \cdot c_1(e)$$

Recall that we used Dijkstra's algorithm in the original routing component (Section 9.3.2) on the routing graph to determine all vertices reachable within a given maximum travel time. Analogously, in graph G, we can use Dijkstra's algorithm with the personalized cost $c_{\bar{\alpha}}$ of a specific user to determine all vertices that are reachable within a maximum cost $c^*_{\bar{\alpha}}$. Unlike the original scenario, the maximum cost $c^*_{\bar{\alpha}}$ lacks specific semantic interpretation. Nonetheless, higher values of $c^*_{\bar{\alpha}}$ equate to a more significant effort required to reach the destination. Therefore, we call $c^*_{\bar{\alpha}}$ the *maximum effort* and analogously $c_{\bar{\alpha}}$ the *effort*. As the maximum effort is not a time-based metric, we will refer to the visualization as *isolines* instead of isochrones in this section.

Example: Preference of official bicycle routes. In Section 6.4, the preference for official bicycle routes is examined using the bicriterial routing model and a dataset of 1016 cycling trajectories. To this end, the following edge cost functions were defined (compare Equation 6.3):

$$c_0(e) = \begin{cases} 0, & \text{if } e \text{ is part of an official bicycle route} \\ d(e), & \text{else.} \end{cases}$$

$$c_1(e) = \begin{cases} d(e), & \text{if } e \text{ is part of an official bicycle route} \\ 0, & \text{else.} \end{cases}$$

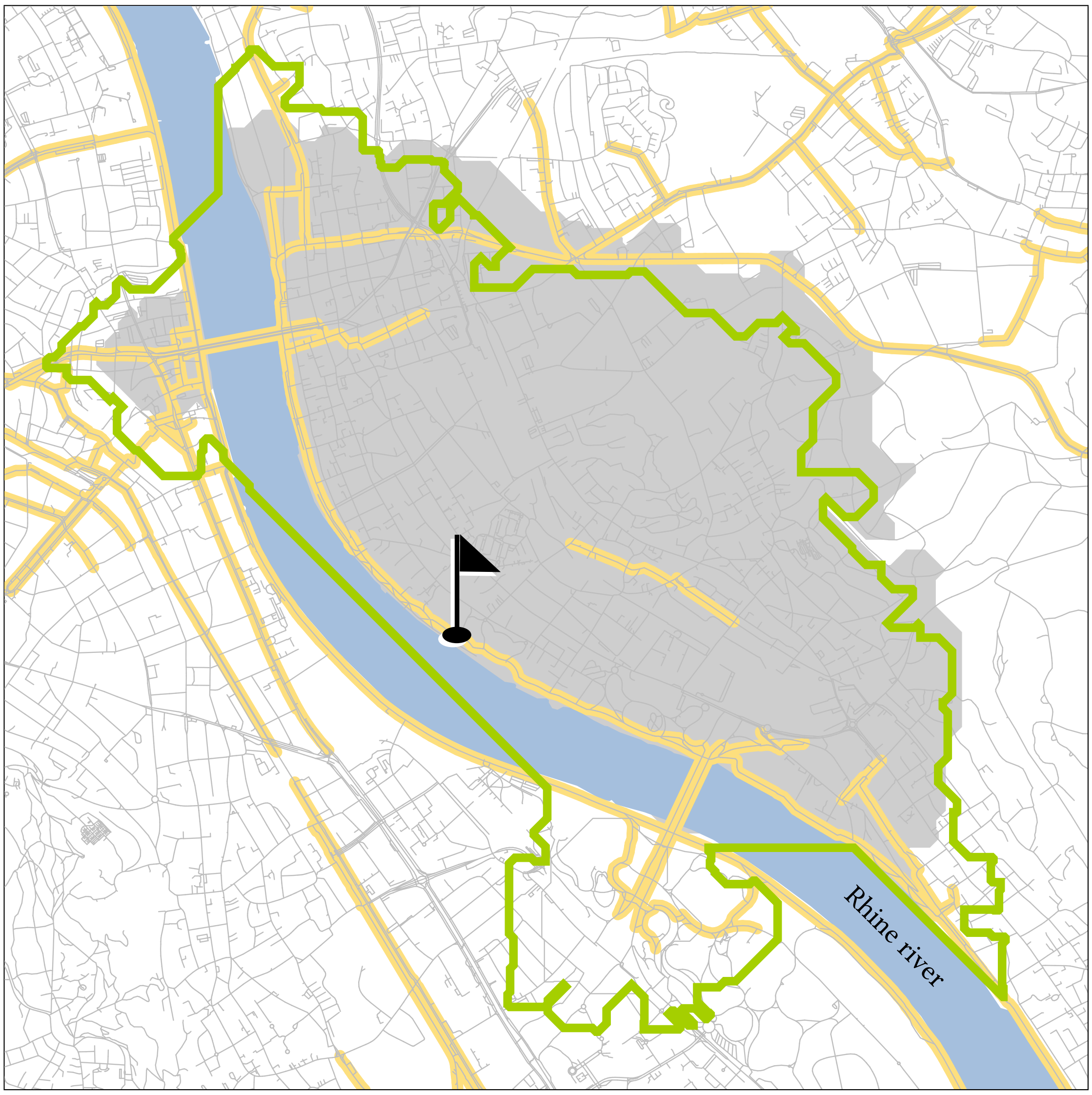


Figure 9.23: Visualizing different routing profiles. The gray area indicates the region accessible by a cyclist I of the group Indiff ($\alpha = 0.5$) covering a maximum distance of 3km from the specified starting point. In contrast, the green outline shows the reachable area for a cyclist P of the group Pro ($\alpha = 0.5721$) with the same maximum effort and starting point. Notably, the dedicated cycle paths (yellow) running along the Rhine River extend P's reach to exceed that of I along the river while the absence of cycle paths away from the river limits P's reach in those directions.

Using the algorithm for inferring routing preferences from the trajectories and the bicriterial routing model (Equation 6.1), the cyclists are classified into three groups: Pro ($\alpha_{\text{Pro}} = 0.5721$), Indiff ($\alpha_{\text{Indiff}} = 0.5$), and Con ($\alpha_{\text{Con}} = 0.4974$).

Previously, we highlighted that the maximum effort lacks semantic interpretation. However, $\alpha = 0.5$ is an exception for the given cost functions. In this case, the effort is defined as $c_{\bar{\alpha}}(e) = 0.5c_0(e) + 0.5c_1(e)$. As either $c_0(e)$ or $c_1(e)$ is $d(e)$ and the respective other is zero, it follows that $c_{\bar{\alpha}}(e) = 0.5d(e)$. Thus, for $\alpha = 0.5$, cycling for a specific effort equates to traversing twice the geometric distance.

In Figure 9.23, two isolines for different routing profiles are shown: one using the routing profile Indiff (gray region) and the other one using the routing profile Pro (green outline). The maximum effort for both isolines is $c^*_{\bar{\alpha}} = 1500$ meters. This results in the Indiff isoline illustrating the area reachable from the starting point within 3000 meters using the road network. For the routing profile Pro, traversing official cycle paths (highlighted in yellow in Figure 9.23) costs less effort than traversing regular roads. This effect is visible along the Rhine River, where paths of the official cycling network are located, resulting in a farther reach in those directions. On the contrary, no cycle paths lead away from the river at the specified starting location, resulting in a reach smaller than the initial three kilometers.

We intentionally opted not to illustrate the routing profile Con, as the resulting isoline closely mirrors the routing profile Indiff. There are two primary reasons for this. Firstly, the balance factors for these two routing profiles are very similar ($\alpha_{\text{Indiff}} = 0.5$ and $\alpha_{\text{Con}} = 0.4974$). Secondly, the conventional road network is much denser than the official cycle path network, with official cycle paths often running adjacent to regular roads. Consequently, alternative routes outside official cycle paths typically involve minimal to no detours compared to the geometric shortest path.

9.6 Conclusion

This chapter presented an automatic approach for the generation of schematic travel-time maps. Outlines of reachable regions are generated from a user location for a specific travel time. Following the spirit of schematized metro maps, our method generates octilinear polygonal zones. This reduces the visual complexity and leads to a higher map legibility attributed to optimizing the number of polygon bends.

We conclude that the computation of schematic travel-time maps can be modeled as a search for a minimum-weight path in an appropriately defined graph. In particular, using the *line graph* of a graph obtained from augmenting a regular octilinear grid with additional segments through ports (i.e., points where the unreachable and reachable part of a network meet) allowed us to heuristically minimize the number of bends of the isochrones. We further showed how to apply this approach to generate travel-time maps for multiple travel times and showed its feasibility through a case study.

Future Work. The evaluation of the results has shown that the different visualization approaches for travel-time maps have advantages and disadvantages. Future research could combine these approaches to benefit from these respective advantages. For example, one could combine our approach with the approach by Baum et al. (2018) to reduce saw tooth patterns in the visualization. Moreover, we deem combining our approach with schematized network maps fruitful. To that end, we extend our approach to create schematized isolines in schematized metro maps. This augmented approach is detailed in the following Chapter 10.

Our approach allows us to visualize different travel times or routing profiles and can, for example, be used to identify deficiencies in the existing road network. For a more advanced analysis of the road network, it may be suitable to compare more than two map states to each other. One way to achieve this is to use animated transitions between the states, so-called *morphs*. In our ongoing research, we aim to create morphs between schematized input lines that preserve the schematic properties of the input. Preliminary results are presented by Forsch et al. (2022b).

Chapter 10

Metrochrones: Schematic Isochrones for Schematic Metro Maps

This chapter is mainly taken from joint work with Jan-Henrik Haunert (Forsch and Haunert, 2023). It presents an algorithm for creating schematic isochrones for schematic input maps. The presented approach is an adaptation to the visualization component presented in Chapter 9, thoroughly optimized to suit the use case of metro maps. A significant modification lies in using a pre-computed grid rather than dynamically creating the grid. This adjustment allows for a formal guarantee that the isochrone can be consistently generated and always maintains its schematic nature.

In the context of public transportation, travel-time maps display the area accessible from a starting point within a given travel time. In these maps, isochrones connect locations of equal travel time. Transit networks are often displayed using a schematic representation, where each edge is restricted to a given set of orientations, e.g., the octilinear directions. Methods for generating isochrones for topographic maps unnecessarily constrain the isochrones to fixed positions, impeding the visual quality. In this chapter, we present an algorithm for generating Metrochrones, which are schematic regions enclosing the reachable area of a transit map. The regions are restricted to the same set of orientations as the underlying transit map. Our approach guarantees that the regions correctly separate the stations in the network into reachable and unreachable stations. Moreover, the result is optimized towards a simple visualization by (a) minimizing the number of bends in the isochrone, (b) forcing the isochrone away from lines in the transit network, and (c) avoiding small crossing angles. The approach can easily be extended to other isolines, such as travel fares.

10.1 Introduction

Transit maps are an important tool for exploring the public transportation network of large cities or metropolitan regions. Traditionally, these maps focus on the topological connectivity of stations by transit lines instead of a geometrically accurate representation of the network. For this, schematic layouts are often employed to improve the readability of the connectivity. A widely applied schematization type is octilinearity. In an octilinear transit map, the transit lines only run at angles that are restricted to being either horizontal, vertical, or 45° diagonal.

While transit maps allow the reader to understand the connectivity of the transportation network, they do not give information about the transit schedule and travel times. One frequently applied approach to visualize travel times in cartography is using isochrones. An *isochrone* is a line connecting all locations that are reachable within a given travel time. Traditionally, geographic maps are used as the base map for displaying isochrones. This chapter presents an algorithm for generating *Metrochrones*: schematic isochrones in octilinear transit maps. Figure 10.1 shows two outputs of our approach generated for the tram network of Cologne, Germany. The reachability within a travel time of 10 minutes is displayed for two example start locations. Our approach guarantees that the isochrones include all reachable stations and none of the unreachable stations.

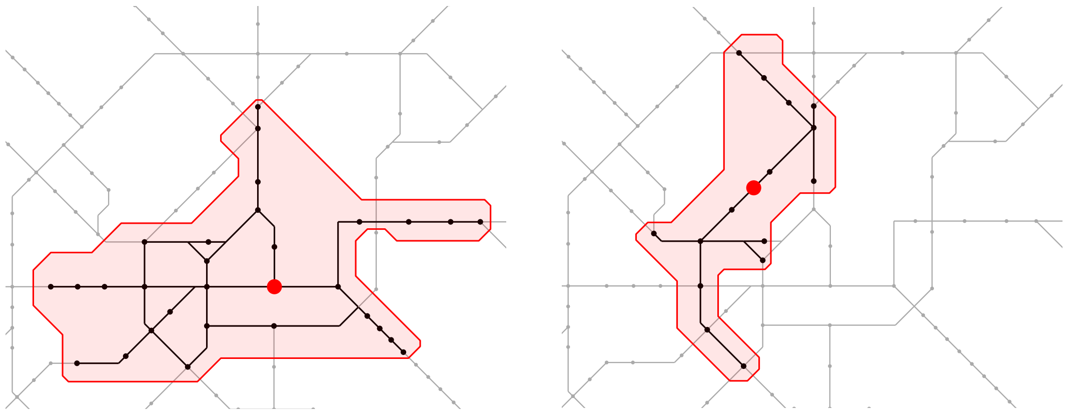


Figure 10.1: Shown are two schematized regions that are generated with our approach. The reachability from the start location (red) within 10 minutes is displayed. The regions are guaranteed to contain all reachable (black) stations and none of the unreachable (grey) stations.

Our approach is based on the work by Forsch et al. (2021), who generate isochrones inside of a road network. Schematic transit maps have structural properties different from road networks, allowing us to use a more specialized approach for generating the isolines. Firstly, in a transit network, the reachability is only interesting for stations, as one can only enter and leave the network at these locations. In contrast, in a road network, the reachability is of interest along the road segments, i.e., the user wants to know how far they can get along a road. Thus, drawing an isoline in a transit network is more flexible, as the separation point between two stations can be arbitrarily chosen. In contrast, in road networks, exactly one location separates reachable and unreachable road segments. Hence, when applied to transit maps, methods for creating isolines for topographic maps unnecessarily constrain the isochrones to fixed positions on the road network, impeding the visual quality. The second difference is related to the network size. Transit maps are schematized and display the transit network in a single map frame suitable, e.g., for printing. Therefore, they are much less detailed than road networks and, due to being schematized, they are much more structured. These properties allow us to use a pre-computed grid for generating the isochrone instead of setting up a grid on the fly, as done in Chapter 9. This greatly increases the performance of the approach. We evaluate our approach based on transit maps of the cities of Cologne, Germany and Paris, France. For this, we use a Java implementation of our algorithm. The implementation and example instances are available as open source at https://gitlab.igg.uni-bonn.de/forsch/metrochrones.

The chapter is structured as follows. Section 10.2 discusses related work on transit maps, schematization, and the generation of isochrones. In Section 10.3, the approach to creating isochrones is described, and design goals, which are integrated into the process, are derived. The performed evaluation is discussed in Section 10.4. Section 10.5 concludes the chapter.

10.2 Related work

In cartography, schematization is an extreme type of abstraction and simplification to maximize the utility of a map for a specific function or task (Burghardt et al., 2014). Schematization can, among others, be differentiated by its geometric style and the type of the schematized object (Meulemans, 2014).

Geometric style. Different geometric styles have been applied in schematization. An important property is the shape of drawn lines. Examples include using curves and circular arcs (van Dijk et al., 2014; van Goethem et al., 2015) and straight-line schematization with an orientation restriction (Stott et al., 2011; Nöllenburg and Wolff, 2006). A popular concept in this context is k-linearity (Nickel and Nöllenburg, 2019), where each line in the map must have an angle that is a multiple of π/k. For $k = 4$, this results in octilinear visualizations where each line is either horizontal, vertical, or 45° diagonal.

Schematization has been applied to different types of geometries, such as point sets (Bonerath et al., 2019), lines (Delling et al., 2010, 2014), networks (Nöllenburg and Wolff, 2011), and regions (Buchin et al., 2016). Most relevant for this chapter is the schematic representation of networks, which forms the base map of our approach and the schematization of regions, in our case, accessibility regions.

Schematizing networks. The schematization of networks is often considered in the context of transit maps. Since the publication of Henry Beck's map design for the London Tube Map of 1933 (Garland, 1994), schematic transit maps have been widely used to support commuters. In Beck's map, the transit lines are restricted to follow either horizontal, vertical, or 45° diagonal orientations. Such maps are referred to as *octilinear* maps. Over the years, many types of schematization have been applied to transit maps, such as curvilinear maps, concentric circle maps, and multilinear maps. For a detailed overview of the different schematization types, we refer to Wu et al. (2020).

Schematizing regions. Cicerone and Cermignani (2012) describe a heuristic approach for generating a rectilinear or octilinear schematization of polygons. However, their approach does not guarantee that the area of the schematized polygon is similar to the original one. Buchin et al. (2016) present an algorithm for an area-preserving schematization of polygonal subdivisions. In contrast to the schematization of single polygons, the topology of neighboring polygons must be preserved in their approach. While the previous approaches relied on octilinear or rectilinear visualizations, van Goethem et al. (2015) explored curved schematization for polygons. Similar to the approach by Buchin et al. (2016), their approach preserves the area and topology of the schematized polygons. In the context of transit maps, Galvão et al. (2020) consider simultaneous schematization of a route with given context information, such as polygonal landmarks. For this, the authors present an algorithm that creates octilinear schematizations, preserving the topology of the route and the landmarks.

Grid-based for schematization. Multiple approaches exist that use a regular grid to perform the schematization. According to Löffler and Meulemans (2017), using regular grids as a basis for schematization has several advantages: (1) the grid can easily model different constraints, (2) the grid promotes collinear edges and uniform edge lengths, (3) the grid avoids visual collapses of the schematized shape. Grid-based approaches have successfully been applied to the schematization of networks (Tamassia, 1987; Bast et al., 2020, 2021) as well as the schematization of regions (Bouts et al., 2016b; Löffler and Meulemans, 2017). In this chapter, we also adopt a grid-based approach to schematize the accessible region in a transit network. In contrast to the aforementioned publications, we are not given the non-schematized input geometry. Instead, we directly generate the schematized accessible region from the classification into reachable and unreachable parts of the transit network.

Travel-time maps. Travel-time maps provide the user with information about the accessibility of a region. Different approaches exist for visualizing the travel time on these maps. Denain and Langlois (1998) present anamorphosis maps, where the base map is distorted such that the geometric distance between two points on the map is proportional to their travel time. Similarly, Buchin et al. (2014) visualize the travel time by edges drawn as sinusoids such that the edges' lengths are proportional to the travel time.

Isochrone maps. The most relevant type of travel-time maps in the context of this chapter are isochrone maps. An isochrone map displays areas that are reachable within a given set of travel times by polygons. The generation of travel-time maps can be divided into two steps: determining the reachable area and visualizing it (Forsch et al., 2021). Determining the reachable area is usually done by applying a routing algorithm such as Dijkstra (1959)'s algorithm on a transportation network graph. For routing in public transportation networks, Pyrga et al. (2008) introduce the time-expended graph model that includes information on the schedule. Baum et al. (2016) present speed-up techniques for routing in the context of isochrones. In this chapter, we focus on the visualization of the reachable area. Creating a separating polygon between two sets of points (e.g., reachable and unreachable) has been investigated in computational geometry, where minimum-perimeter polygons (Edelsbrunner and Preparata, 1988) and minimum-link polygons (Eades and Rappaport, 1993) have been considered.

An early example of visualizing isochrones on a map is the Tramway Time Zones map of Melbourne (Melbourne and Metropolitan Tramways Board, 1922). In this map, concentric circles around reachable railway stations indicate the remaining travel time at this location. The circles only approximate the reachable area; thus, the time zone might cover unreachable areas and vice versa. Since then, different approaches for the automatic visualization of isochrones have been developed that enforce that the time zones correctly separate reachable and unreachable areas. In the context of road networks, Baum et al. (2018) present an approach for displaying isolines, i.e., lines connecting locations of equal value. Forsch et al. (2021) present an approach for visualizing octilinear isochrones for multimodal transportation, i.e., public transportation and walking. Both approaches guarantee that the road network is correctly separated into reachable and unreachable roads. Additionally, both approaches use minimizing the number of bends in the isoline as the optimization criterion.

10.3 Methodology

We are given a schematic transit map consisting of a set of stations S and a set of connecting lines L. The transit map is schematized using rectilinearity or octilinearity, meaning that all lines in L are restricted in their orientation to a set of rectilinear or octilinear orientations, e.g., an octilinear transit map only contains horizontal, vertical, and 45° diagonal lines. We will describe the octilinear case in the following for ease of notation. The approach can easily be adapted for the rectilinear case.

Next to the layout of the transit map, we are given a classification of all stations into *reachable* or *unreachable*. This classification could, for example, stem from the output of a routing algorithm or result from fare zone regulations. Take Figure 10.2a as an example: the black stations could represent the stations reachable with a specific tariff. We aim to create a polygon that encloses all reachable stations while not containing any unreachable stations, following the same schematization as the transit map. We call the outline of the polygon the *isoline*.

In Section 10.3.1, an algorithm for creating such an isoline is introduced. The algorithm allows for the integration of different design goals using different edge weightings in the graph, such as a minimal number of bends or avoiding an isoline close to existing transit lines. Appropriate design goals are reviewed in Section 10.3.2, and their assembly into the framework is discussed in Section 10.3.3. A result generated with our approach is displayed in Figure 10.2b.

10.3.1 Algorithm for generating a valid isoline

Our approach for generating an isoline is based on minimum-cost routing to generate a closed path around the reachable part of the network. A pre-computed grid is used to set up an appropriate graph for the routing such that the computed path is guaranteed to correctly separate reachable and unreachable parts of the network.

Input. The transit map consists of a geometric graph $G = (S, L)$ with the stations being its vertices and the connecting lines being its edges. In general, G can contain lines crossing each

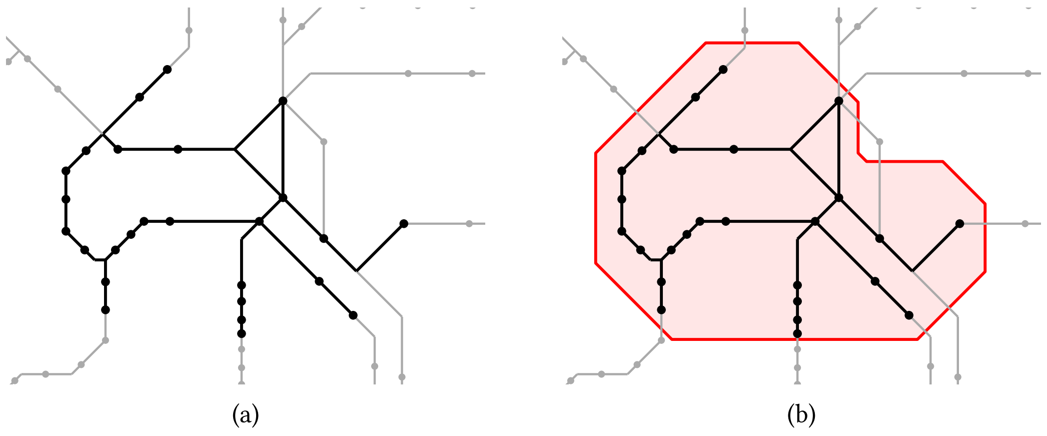


Figure 10.2: (a) Drawing of a classified, octilinear metro network. The black stations are reachable, while the grey stations are unreachable. (b) Our algorithm creates an octilinear isochrone, which separates the reachable and unreachable stations.

other outside of the stations. Additionally, a line connecting two stations can have bends. We planarize G into a graph $\overline{G} = (\overline{S}, \overline{L})$ that is crossing free by subdividing crossing edges in L with additional vertices at their intersections.

As stated before, the stations are classified into reachable and unreachable stations. This classification can easily be extended to the edges: an edge is reachable if it is incident to two reachable stations; otherwise, it is unreachable. Additionally, we call an intersection vertex reachable if one of the two lines intersecting at its location is reachable. As a result, all graph elements are classified as either reachable or unreachable.

Gates and ports. The planarized graph $\overline{G}$ has edges that are incident to both a reachable and unreachable vertex. We call these edges *gates*. Further, we call the reachable vertex of a gate its *port*. By removing all gates from $\overline{G}$, we decompose the graph into a set of components $C_1, \ldots, C_\ell$, where each component consists entirely of reachable parts or entirely of unreachable parts.

In the following, we assume that we are given only one reachable component C_1 that does not contain a hole, i.e., C_1 is reachable and $C_2, \ldots, C_\ell$ are unreachable. We call this the *base case*. At the end of the section, we show how to solve the general case based on the base case.

Let $g_1, \ldots, g_k$ be the gates of $\overline{G}$ on the boundary of C_1 in clockwise order. We can identify these by following the outline of C_1, starting at an arbitrary location. Moreover, we define $g_{k+1} := g_1$. We observe that any two consecutive gates (g_i, g_{i+1}) are incident to the same face of $\overline{G}$, which we denote f_i. Collecting the faces for all gates results in a set $F = (f_1, \ldots, f_k)$ of *reachability-splitting faces*. Note that a face can appear multiple times in F if more than two gates are incident to it; see the faces f_1 and f_{10} in Figure 10.3 as an example.

Figure 10.3 shows the gates and the reachability-splitting faces for the example shown in Figure 10.2. Each face in F separates the reachable and unreachable parts of $\overline{G}$. Therefore, the isoline must split each face into two parts. Recall that the two gates g_i and g_{i+1} are incident to the face f_i. These gates denote the separation point between the two parts of the network. Thus, the section of the isoline running through f_i must connect these two gates.

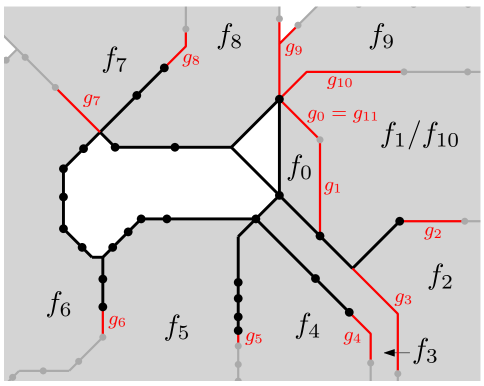


Figure 10.3: The faces in F that separate the reachable component from the unreachable parts of the network are highlighted (grey). They are identified in clockwise order around the component.

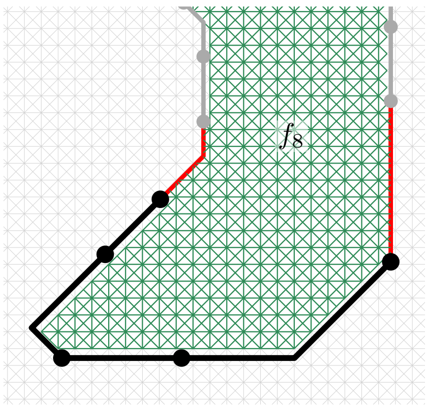


Figure 10.4: For each face (here f_8), a grid graph (green) is extracted from a pre-computed grid (light grey). At gates (red), the grid graph can touch the outer boundary of the face.

Our procedure for the base case consists of three steps. In the first step, we identify all reachability-splitting faces, and for each of these faces, we extract a grid graph from a pre-computed, octilinear grid that connects its incident gates. We join the individual grid graphs at the gates to create an octilinear routing graph. In the final step, we perform a minimum-cost routing in the routing graph to get an octilinear path. This resulting path is the region outline.

Grid graphs. We define an infinitely-large octilinear grid graph from which we generate a solution space for the segments of the isoline. A grid graph $\mathcal{G}_w = (\mathcal{V}, \mathcal{E})$ with grid spacing w is constructed as follows. For every point $(i \cdot w, j \cdot w)$ with $i, j \in \mathbb{Z}$, we add a grid vertex $v_{i,j}$ to $\mathcal{V}$. Each vertex $v_{i,j}$ is connected to each of its eight octilinear neighbors by a grid edge $e \in \mathcal{E}$. Using this definition, any path $p = (e_1, \ldots, e_n)$, with $e_i \in \mathcal{E}$ is an octilinear line. At the end of this section, we explain how to choose w such that $\mathcal{G}_w$ contains a valid solution for the isoline for any classification of the stations in $\overline{G}$ into reachable and unreachable stations.

Step 1: Extracting a grid graph for each face. For each reachability-splitting face f_i in F, we extract a subgraph $\mathcal{F}_i = (V_i, E_i)$ from $\mathcal{G}_w$ that connects its two gate edges g_i and g_{i+1}. We use the pre-computed grid $\mathcal{G}$ to initialize V_i and E_i. Every grid point in $\mathcal{G}_w$ inside of f_i becomes a vertex in $\mathcal{F}_i$. Every grid segment in $\mathcal{G}_w$ that is completely contained in f_i, i.e., that is inside of f_i and does not touch its boundary, becomes an undirected edge in $\mathcal{F}_i$. Additionally, for every segment in $\mathcal{G}_w$ that crosses one of the gates of f_i, we insert this crossing point as a vertex and the part of the segment that is inside of f_i as an undirected edge to $\mathcal{F}_i$. Figure 10.4 shows the grid graph created for face f_4 of Figure 10.3. Notice how the grid graph only touches the face's boundary at the two gate edges and creates a connection between them.

Step 2: Creating the routing graph. In Step 1, we created grid graphs for the individual faces in F. We combine the grid graphs into one large routing graph $\mathcal{R}$ by joining for $i = 2, \dots, k$ every two consecutive grid graphs $\mathcal{F}_{i-1}$ and $\mathcal{F}_i$ by edges. Recall that the grid graphs $\mathcal{F}_{i-1}$ and $\mathcal{F}_i$ share the common gate g_i; see Figure 10.5a. By construction, both grid graphs have vertices on the same locations on g_i, namely the intersection points of g_i with the grid $\mathcal{G}$. These are the *gate vertices* of g_i. For each such pair of coincident vertices, we connect the vertices with a directed edge from the vertex in $\mathcal{F}_{i-1}$ to the vertex in $\mathcal{F}_i$. Figure 10.5b shows an enlarged version of g_i highlighting the inserted edges. We refer to these edges as *gate edges*. Later on, we can apply different edge weightings to the gate edges to favor specific gate crossings over others. An important observation is that the two consecutive grid graphs $\mathcal{F}_{i-1}$ and $\mathcal{F}_i$ are only connected along their common gate g_i. As a result, the gates can only be used in sequence when routing in the combined routing graph, even when a face appears multiple times in the set of reachability-splitting faces F.

Notice that we did not connect the grid graph $\mathcal{F}_k$ to $\mathcal{F}_1$ along gate g_1, see Figure 10.5c. We insert an additional dummy vertex into the routing graph for each vertex of $\mathcal{F}_1$ along g_1. These vertices serve as sources for the routing performed in Step 3. The source vertices are connected to their corresponding gate vertex by a directed edge, which serves as the gate edge for the first gate.

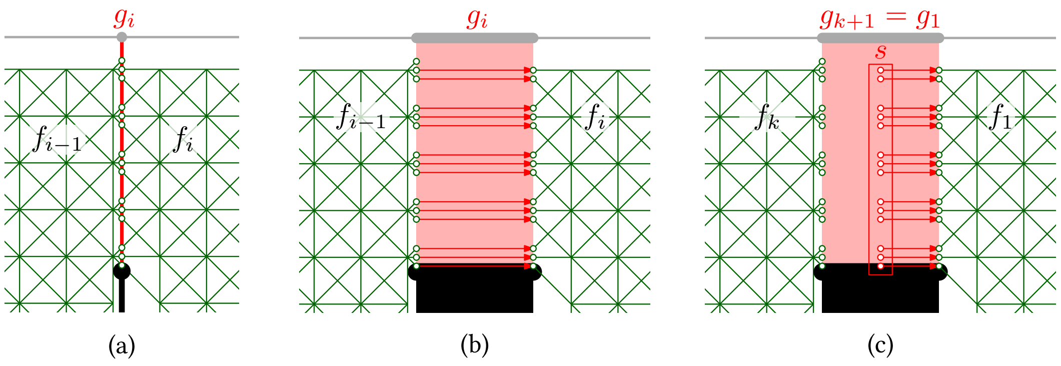


Figure 10.5: Creating the routing graph $\mathcal{R}$. (a) Along gates (red), the grid graph is extracted so its edges touch the gate. (b) Detailed view of the gate. Two gate vertices of consecutive grid graphs are connected by a gate edge (red). (c) The first and last grid graphs are not connected. Instead, the source vertices s are added.

Step 3: Routing. In the previous step, we created a routing graph $\mathcal{R}$ that connects all gates. Figuratively speaking, the routing graph forms a ring around the reachable component of the transit network. Even though this ring is closed geometrically, the graph itself is separated at the gate g_1, where the coincident vertices of $\mathcal{F}_1$ and $\mathcal{F}_k$ are not connected. We assume that we are given a cost function c that expresses how suitable an edge in $\mathcal{R}$ is for being part of the isoline. The lower the cost, the better the edge represents the reachability. The details on setting up cost function c are outlined in Section 10.3.3. We generate the isoline by performing a minimum-cost path query regarding c in $\mathcal{R}$. For example, the algorithm of Dijkstra (1959) can be used for this task. The isoline is only valid if it is closed; its source vertex must coincide with the target vertex. Therefore, for each vertex v_1 on g_1 that has a coincident vertex v_{k+1} on g_{k+1}, we perform a minimum-cost path query starting at the dummy source of v_1 and ending at v_{k+1}. This will generate a set of candidates for the isoline. From this set, we select the minimum-cost solution regarding c as the final isoline.

Ensuring a valid solution. In the following, we show how to choose a grid spacing w^* that ensures that a valid solution for the isoline can be found in $\mathcal{G}_{w^*}$. In particular, we need to choose a sufficiently fine grid to ensure that the graphs $\mathcal{F}_i$ and $\mathcal{F}_{i+1}$ for $i = 1, \dots, k$ are geometrically connected via the incident gate g_i. However, choosing a very small value for w^* will increase the algorithm's running time.

Therefore, to choose w^* appropriately, we use the following result by Brimkov et al. (2011, Theorem 12): Given a connected surface A in $\mathbb{R}^2$ and a unit grid (i.e., an integer grid with spacing 1), the set of grid cells whose centers are within distance $1/2$ of A is connected via the 8-neighborhood. Since connectivity is not affected by scaling, it also holds for any surface A in $\mathbb{R}^2$ and grid with spacing w that the set of grid cells whose centers are within distance $w/2$ of A is connected via the 8-neighborhood.

Let f_i and f_{i+1} be two adjacent faces in G that share a common gate g_i. Additionally, let p_i be a point inside of f_i furthest away from f_i's boundary. Analogously, p_{i+1} is a point inside of f_{i+1} furthest away from f_{i+1}'s boundary. This situation is shown in Figure 10.6a. By removing g_i, we get a single face P. Further, let d be the largest distance one can buffer the boundary of P to the inside so that the resulting geometry A is connected, includes a point on g_i, and includes the points p_i and p_{i+1}, see Figure 10.6b. By construction, the area within distance d of A is guaranteed to be contained inside of P. Additionally, following the findings of Brimkov et al. (2011), all grid points within distance d of A are octilinearly connected on a grid with spacing $w = 2d$. To extend this guarantee to all faces of $\overline{G}$, we have to compute the minimum distance d^* over all of its faces. We can select $w^* = 2d^*$ to ensure octilinearly connected grid graphs.[1]

General case. Until now, we assumed that $\overline{G}$ only has one reachable component that does not contain unreachable parts. If $\overline{G}$ has multiple reachable components, we enclose each individually. For this, we first select one reachable component and generate the isoline using our solution to the base case. If a reachable component C_r contains an unreachable component C_u, we enclose both C_r and C_u. In theory, reachable and unreachable components can be nested arbitrarily deep, requiring an iteration of this process. In practice, we never encountered more than a nesting level of one (i.e., an unreachable component being enclosed by a reachable component). We refer to Chapter 9 for a more formal definition of the general case.

[1] Brimkov et al. (2011) report a minimum distance of $\sqrt{2}/2$ to guarantee connectivity via the 4-neighborhood on a unit grid, resulting in a minimum grid spacing of $w^* = \sqrt{2}d^*$ in the rectilinear case.

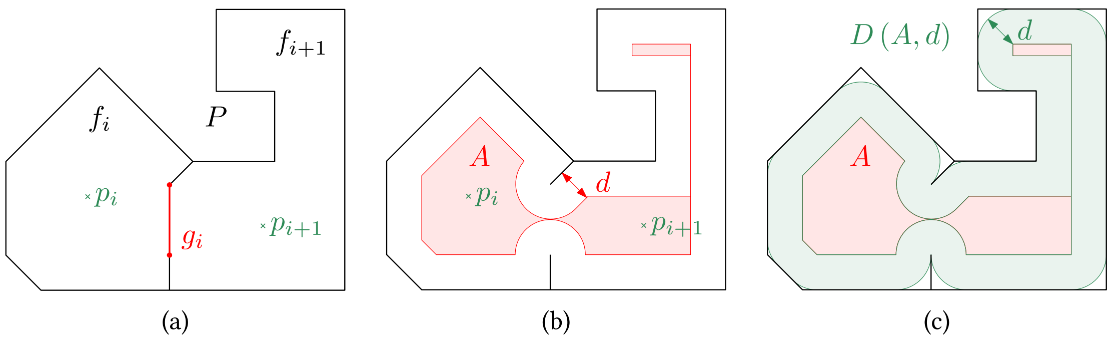


Figure 10.6: (a) Two faces f_i and f_{i+1} that are incident to each other and share a common gate g_i. By removing g_i, we get a face P. (b) The buffer distance d is the largest distance that can be buffered inside of P such that the resulting geometry A is connected, contains a point on g_i, and contains p_i and p_{i+1}. (c) The area within distance d of A (green) is contained inside of P, and all grid points covered by it are octilinearly connected if the underlying grid has a grid spacing of $w = 2d$.

10.3.2 Design criteria

In previous research, isochrones have often been generated using basic geometric principles, leaving little room for design choices. For example, a circle is drawn for each reachable station whose radius corresponds to the remaining travel-time budget. In the following, we review design choices from cartography and information visualization domains to adopt them for a more flexible approach to generating isochrones.

When visualizing information, the simplicity of shapes is known to be a key factor influencing the quality of perception (Van Lier, 1996). In this regard, many researchers focused on deriving measures that express the aesthetics of a shape (Birkhoff, 1933; Eysenck, 1941; Boselie and Leeuwenberg, 1984). The list of characteristics used in the proposed formulas is rich and includes measures of complexity, symmetry, angles, repetition, pattern regularity, and compactness. Especially when dealing with geographic shapes, compactness is considered the most crucial spatial property (MacEachren, 1985). Compact shapes typically feature a less complex boundary and are more space-efficient than non-compact ones. Therefore, we aim for compact shapes when computing schematic isochrones for metro maps.

Another frequently applied design criterion is minimizing the number of bends in polylines, also often referred to as *straightness* (Tamassia, 1987; Hong et al., 2006). Studies in the context of graph drawing have shown that reading graphs with a lower number of bends results in fewer errors than reading graphs with a higher number of bends (Purchase, 1997, 2000). Sometimes, bends or crossings in the lines cannot be avoided. In these cases, having larger crossing angles has been proven to increase the readability of the graph layout (Ware et al., 2002; Huang et al., 2008, 2014). Di Giacomo et al. (2011) study the trade-off between, among others, having fewer bends and increasing the minimum angle formed between two crossing segments. They present algorithms for drawing non-planar embeddings with constrained minimum edge crossing angles and the number of bends. In this chapter, we aim to increase the crossing angle between line intersections of our isoline and existing lines while minimizing the number of bends in the isoline.

Design criteria are also considered in the context of external labeling (Bekos et al., 2019). In external labeling, textual annotations are placed on the margins of an illustration and connected to the annotation location by so-called *leaders*. Next to the aforementioned criteria, such as minimizing the number of bends, the *distinctiveness* of these leaders was also identified as an important design criterion (Niedermann et al., 2017). For example, leaders running close to each other should not be parallel. Therefore, we aim to avoid lines running close to each other and lines running close to stations.
As a conclusion, we have identified the following design criteria:

C1 *minimizing the number of bends,*

C2 *avoiding acute crossing angles,*

C3 *avoiding lines close to existing features,*

C4 *increasing the compactness.*

In the following section, we will explain our approach. Afterward, we explain how to incorporate these design criteria into the approach.

10.3.3 Optimization

In Section 10.3.1, we outlined an approach for generating a valid isoline separating the reachable and the unreachable part of the transit network. This approach relies on a cost function c that quantifies a segment's suitability for representing the isoline. This section introduces appropriate cost functions to quantify the design goals identified in Section 10.3.2. We then describe how to integrate these cost functions into a combined cost function c.

Bends. In Design Criterion C1, we aim at minimizing the number of bends in the isoline. To be able to penalize bends, we perform the routing step on the pseudo-dual graph $\partial\mathcal{R}$ of $\mathcal{R}$ as introduced by Winter (2002). This allows us to apply a cost between two consecutive edges e_i, e_j in the grid graph $\mathcal{R}$. We use $\theta(e_i, e_j)$ to denote the angle between e_i and e_j, with $\theta = 0$ indicating a straight line having no bend.

We suggest two different cost functions to penalize bends. The first one penalizes all bends uniformly:

$$c_b^u(e_i, e_j) = \begin{cases} 1 & \text{if } \theta(e_i, e_j) > 0, \\ 0 & \text{else.} \end{cases}$$

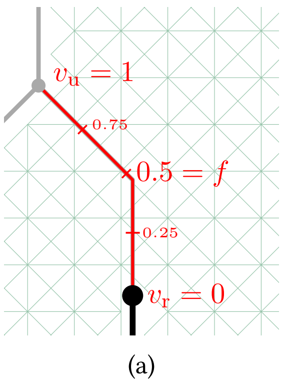
Implicitly, this cost function favors very sharp bends because these bends cover a larger turn angle at the same cost. These sharp bends reduce the compactness of the resulting shape, which contradicts Design Criterion C4. Therefore, the second bend cost function s_b^w weights the bends according to their bend angle θ:

$$c_b^w(e_i, e_j) = \theta(e_i, e_j) \cdot \frac{C}{2\pi},$$

where C is the schematization number (i.e., the number of allowed directions). For octilinear grids, i.e., $C = 8$, this results in $c_b^w(e_i, e_j) = 1$ for 45° bends, $c_b^w(e_i, e_j) = 2$ for 90° bends, and $c_b^w(e_i, e_j) = 3$ for 135° bends.

The cost functions c_b^u and c_b^w are alternatives to each other. In the following, if the differentiation between c_b^u and c_b^w is not relevant, we refer to the cost function penalizing the bends as c_b.

Passage location. For best visibility, the location where the isoline crosses a gate should be far away from the two stations the gate is incident to. While this is, strictly speaking, not in our design criteria, it is closely related to Design Criterion C3. The optimization goal *gate passage location* favors specific crossing locations farther away from stations over closer ones. In particular, the optimal location is specified by a fraction f along the gate starting at the reachable station. Let $g = (v_\mathrm{r}, v_\mathrm{u})$ be a gate connecting the reachable vertex v_r to the unreachable vertex v_u. Further, let e_g be a gate edge in $\mathcal{R}$ that crosses g at location p_i. The



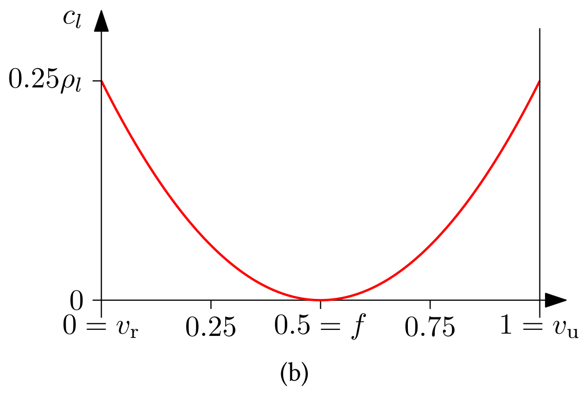


Figure 10.7: (a) A gate $g = (v_\mathrm{r}, v_\mathrm{u})$ with visualised fractions. For $f = 0.5$, the optimal passage location is precisely in the middle of the gate. (b) The gate passage location cost function c_l for $f = 0.5$. At f, the optimal passage location, the cost is zero.

intersection point p_i divides g into two sub edges g_r and g_u, with g_r being incident to v_r and g_u being incident to v_u. The gate passage location cost c_l is defined as:

$$c_l = \left(\frac{\ell(g_\mathrm{r})}{\ell(g)} - f \right)^2 ,$$

where ℓ returns the geometric length of the edge. c_l evaluates to zero at location f and increases quadratically with the distance to f. In the remainder of this chapter, we will use $f = 0.5$, i.e., the central location on the gate between the reachable and the unreachable stations as the optimal passage location. Figure 10.7 visualises the passage location cost. The optimal passage for the gate shown in Figure 10.7a is close to its bend at $f = 0.5$, where s_l is almost zero (see Figure 10.7b).

Passage orthogonal crossing. Large crossing angles are more readable than smaller ones (Design Criterion C2). The optimization goal *gate passage angle* is applied to favor orthogonal or near orthogonal crossing angles between the isoline and the gates over more acute ones. Let g be a gate and e_g be a gate edge in $\mathcal{R}$ that crosses g at an angle $\phi(e_g, g)$. The gate passage angle cost c_a is defined as:

$$c_a(e_g) = \begin{cases} 1 & \text{for } \left|\phi(e_g, g) - \frac{\pi}{2}\right| > \phi_{\max}, \\ 0 & \text{else,} \end{cases}$$

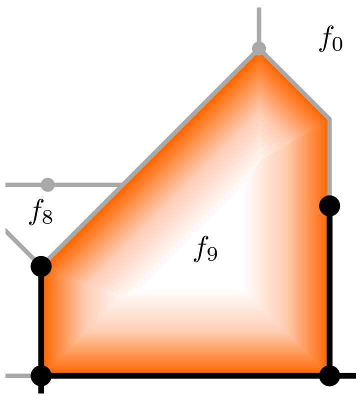
where $\phi_{\max}$ is an angle threshold to decide whether an angle is favorable. In the octilinear variant, there can only be gate crossings of 90°, i.e., orthogonal crossings and crossings of 45°. We only want to favor orthogonal crossings. Therefore, we set $\phi_{\max} = 0$.

Distance from face boundary. The isoline should be easily distinguishable from the network lines. Following Design Criterion C3, this is not the case if the isoline runs close to the face boundary. Therefore, we want to force the isoline away from existing network lines. We scale the length ℓ of a grid edge according to its distance to the face boundary d to retrieve the distance to boundary cost c_d:

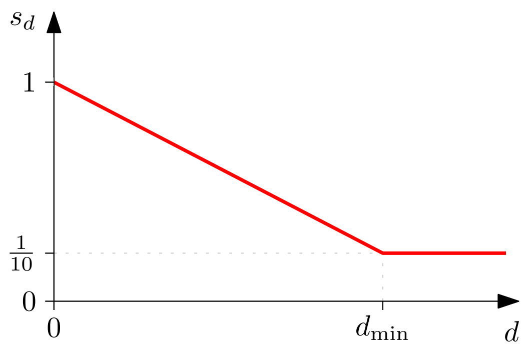
$$c_d(e) = \ell(e) \cdot s_d(e),$$

where

$$s_d(e) = \begin{cases} 1 - \frac{9}{10} \cdot \frac{d}{d_{\min}} & \text{for } d < d_{\min}, \\ \frac{1}{10} & \text{else.} \end{cases}$$



(a)



(b)

Figure 10.8: (a) Visualization of the distance to boundary cost (orange). The cost is the highest at the face boundary, while it is neutral in the face center. (b) The distance to boundary scale factor s_d. Edges closer to the face boundary than $d_{\min}$ are penalized.

Here, $d_{\min}$ is the distance an edge needs to be away from the face boundary to not be penalized. In the remainder of this chapter, we will select $d_{\min} = 30$. In Figure 10.8, the scale factor s_d is shown in detail.

Combined cost function. We defined four different cost functions. We combine these factors to a combined cost function $c\colon E \longrightarrow \mathbb{R}$ that is applied to all edges in $\mathcal{R}$. To weigh the importance of the individual factors, we introduce four weighting factors ρ_d, ρ_l, ρ_a, and ρ_b. The combined, weighted cost function c is defined as:

$$c(e) = \begin{cases} \rho_d \cdot s_d(e) & \text{if } e \text{ is a grid edge,} \\ \rho_l \cdot c_l(e) + \rho_a \cdot c_a(e) & \text{if } e \text{ is a gate edge.} \end{cases}$$

In the pseudo-dual graph $\partial\mathcal{R} = (\partial V, \partial E)$, the edge cost function for two consecutive edges e_i and e_j in the primal graph $\mathcal{R}$ is joined with the edge cost function of $\mathcal{R}$. The final cost function $\partial c : E \times E \longrightarrow \mathbb{R}$ thus is:

$$\partial c(e_i, e_j) = \rho_b \cdot c_b(e_i, e_j) + c(e_i).$$

For details, we refer to Winter (2002).

Summing up this section, we have a cost function ∂c for our pseudo-linear dual routing graph ∂R. This cost function is a weighted sum of five different cost functions c_b^u, c_b^w, c_l, c_a, and c_d, whereas c_b^u and c_b^w are mutually exclusive to each other. Each cost function has a weighting term ρ associated with it that defines the relative importance of the specific cost compared to the other costs. The higher ρ, the more important the respective cost factor is.

10.4 Evaluation

In this section, we perform a quantitative analysis of our approach based on the optimization criteria defined in Section 10.3.3. As described in Section 10.3.2, minimizing the number of bends is a common optimization goal in schematization that has also previously been applied in related work dealing with the visualization of travel times. As such, we select Design Criterion C1 *minimizing the number of bends* as our primary criterion for the analysis. In our evaluation, we want to analyze how strongly the other criteria influence this optimization goal. For this, we perform a case study based on the transit network of Cologne, Germany.

Evaluation instances. For our evaluation, we are using instances generated for the tram network of the city of Cologne, Germany. The underlying transit map reflects the layout of the tram lines in the current rail network plan of the local transportation agency (Verkehrbund Rhein-Sieg, 2023b). The reachability in these instances is hand-generated using a public journey planner (Verkehrbund Rhein-Sieg, 2023a) to represent travel times in the network. Note that this process can be automated by including a routing engine in the code. As our focus is on visualization, we refrained from doing so.

Baseline Solution. We use the instance shown in Figure 10.9 as our baseline solution. Figure 10.9a shows the tram network (grey) with reachable stations and lines colored black. Additionally, the used grid is shown. Figure 10.9b shows the solution of our approach when only applying c_b^u, i.e., minimizing the number of bends. The minimum number of bends needed to correctly separate the reachable and unreachable stations on the chosen grid is 10. A visual inspection of the result reveals some weaknesses in the visualization. For example, in the lower left, the generated region runs very close along the transit lines, making it hard to see the transit line. Another example is along the top, where the region cuts off very close along the reachable stations, making it hard to distinguish whether these stations are included.

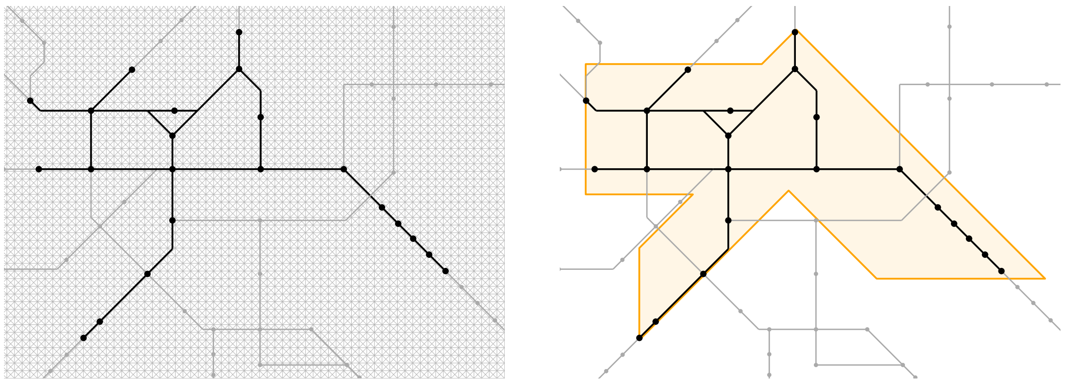
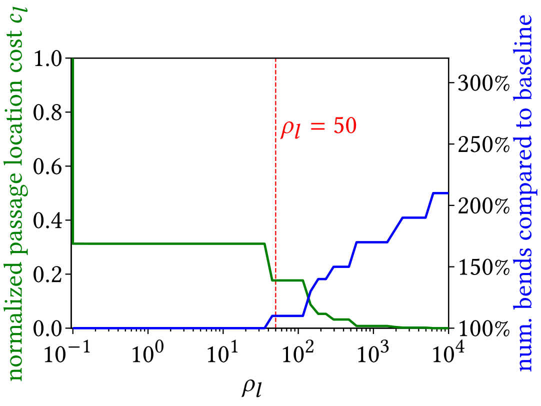

(a) (b)

Figure 10.9: Evaluation instance showing the tram network of Cologne. (a) Reachable stations (black) and grid (light grey). (b) Polygon enclosing the reachable stations with a minimal number of bends.

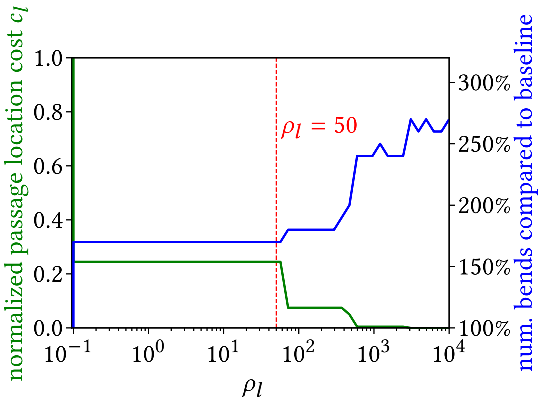
Parameter tuning. In the first step, we analyze the influence of the weighting factors ρ of the four optimization criteria. To analyze the effect of the individual cost functions, we perform a pairwise comparison of our secondary optimization criteria with the number of bends in the polygon. We do this by fixing ρ_b, the weighting factor for minimizing the number of bends to $\rho_b = 10$ and iteratively increasing the weighting factor of the second considered criteria from $\rho = 10^{-1}$ to 10^4 (totaling 50 logarithmically spaced values) while not considering the other criteria (i.e., setting their weighting factor to zero).

Figure 10.10 shows the results obtained for the test instance displayed in Figure 10.9. Figure 10.10a displays the value of the passage location cost function c_l on the y-axis over the weighting factor ρ_l for this cost function. The value is scaled to the interval $[0, 1]$. Additionally, on the second y-axis, the increase in the number of bends compared to our baseline solution is shown. Analogously, Figure 10.10b displays the data for the passage angle cost c_a with ρ_a and Figure 10.10c displays the data for the distance to face cost c_d with ρ_d.

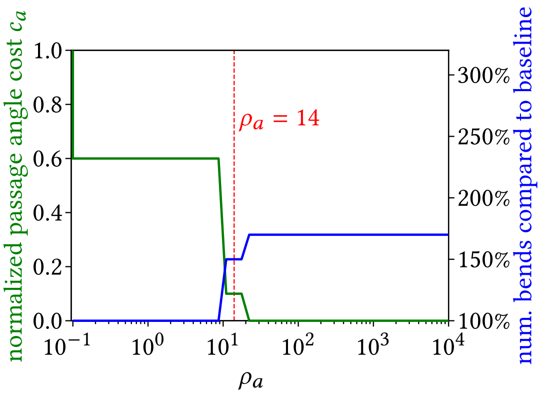
For all three optimization criteria, a substantial decrease in the individual cost can be achieved even without increasing the number of bends. This decrease is the largest for the passage location cost, down to a remaining cost of $\approx$ 30% compared to the maximum, and the lowest for the passage angle, down to a remaining cost of $\approx$ 60% compared to the maximum. Subsequently, there is a small interval for which the cost function can be further reduced while not increasing the number of bends by more than 50% compared to our baseline solution. We argue that this increase in the number of bends is reasonable. Thus, we select the corresponding weighting factor for the following analysis. This results in the following set of weighting parameters: $\rho_b = 10$, $\rho_l = 50$, $\rho_a = 14$, and $\rho_d = 1$.



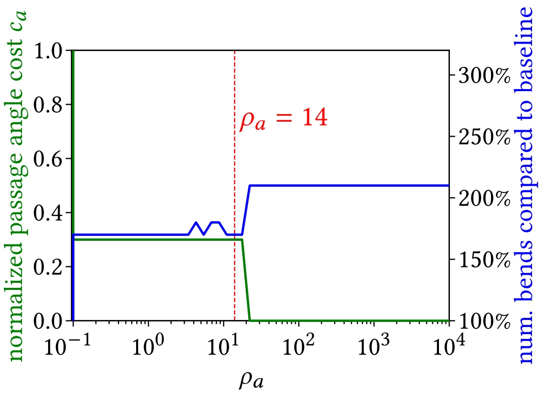
(a) passage location c_l



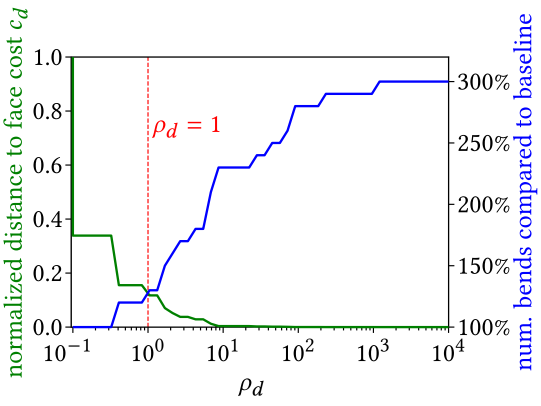
(a) passage location c_l



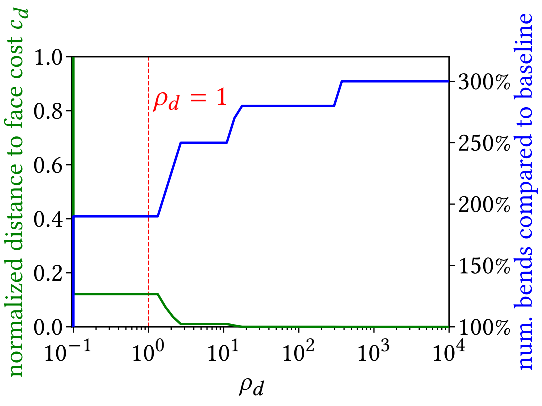
(b) passage angle c_a



(b) passage angle c_a



(c) distance to existing lines c_d



(c) distance to existing lines c_d

Figure 10.10: Comparing c_b^u to the other criteria. For all results, $\rho_b = 10$ is fixed.

Figure 10.11: Comparing c_b^w to the other criteria. For all results, $\rho_b = 10$ is fixed.

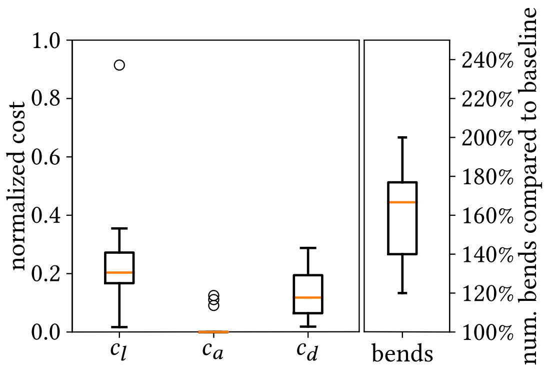


Figure 10.12: Quality for the individual cost functions using c_b^u with $\rho_b = 10$, $\rho_l = 50$, $\rho_a = 14$, and $\rho_d = 1$.

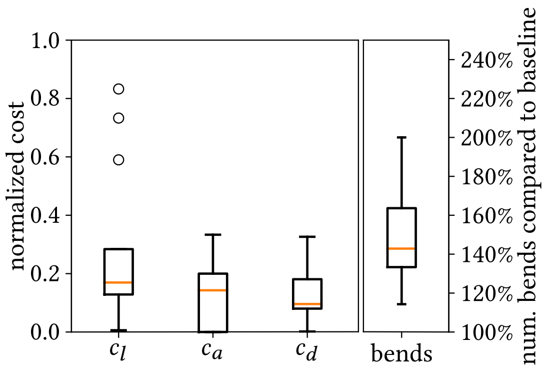


Figure 10.13: Quality for the individual cost functions using c_b^w with $\rho_b = 10$, $\rho_l = 50$, $\rho_a = 14$, and $\rho_d = 1$.

To confirm that these results achieved on a single instance are representative of our approach, we apply this set of parameters to different instances and analyze whether the influence is analogous. For this, we use 13 different instances and compute the enclosing region with the selected set of parameters. Additionally, we determine the minimum and maximum possible value for every cost function by setting the corresponding weighting factor to $\rho = 10000$ and $\rho = 10^{-6}$, respectively ($\rho = 10^{-6}$ is used to ignore the specific cost function except for using it as a tiebreaker among equally good solutions in the other objectives). This allows us to analyze the relative cost for each cost function. Figure 10.12 shows the results of our analysis averaged over all instances. For the three secondary cost functions, the average cost is decreased to 0.2 or lower with only a few outliers with a cost greater than 0.8 of the maximum cost. Similarly, the increase in bends averages around $\approx$ 65%, which aligns with our findings for the initial instance. We conclude that our results generalize well, independent of the specific instance.

	average	maximum
number of vertices/edges in $\mathcal{R}$	10,220 / 28.643	24,248 / 68,890
number of vertices/edges in $\partial\mathcal{R}$	57,099 / 339,285	137,483 / 820,347
time needed to set up $\partial\mathcal{R}$	2.06 sec	3.92 sec
time needed to find the best path in $\partial\mathcal{R}$	3.15 sec	6.91 sec

Table 10.1: Performance of the algorithm. The computation times mainly depend on the size of the (pseudo-dual) routing graph $\partial\mathcal{R}$. Displayed are the measures for the evaluation instances of Cologne.

Table 10.1 shows the running time of our algorithm during the evaluation of the 13 evaluation instances for Cologne. The routing graph's size is the most important factor influencing the running time. The experiments show that the individual zones are, on average, computed within $\approx$ 5 seconds. The largest instance took approximately 10 seconds to compute. These running times make the approach suitable for manually fine-tuning the parameters for a given instance. In case faster running times are needed, we suggest two methods for improving the running times: Firstly, in our experiments, we used a grid width of d^* as introduced in Section 10.3. While this gives us a formal guarantee that a valid solution can be found, we observed that even for grid widths up to $3d^*$, a solution was found in our instances. This substantially reduces the size of $\partial\mathcal{R}$ and thus speeds up the algorithm. In conclusion, an adaptive approach could be applied, starting with a coarse grid and refining it if no valid solution is found. Secondly, our current implementation uses a generic variant of Dijkstra's algorithm without using the routing graph's grid structure. More specialized routing algorithms can speed up the routing part, e.g., the algorithms developed by Wenzheng et al. (2019) or Blum et al. (2021, 2023).

Comparing c_b^u and c_b^w. Until now, we only considered a uniform weighting (c_b^u) for weighting the number of bends. As explained in Section 10.3.2, the compactness of a shape is an essential aspect in the esthetic of a geometric shape. Rounder shapes are more compact than other shapes, which motivated the introduction of the alternative cost function c_b^w, penalizing sharp bends more harshly than multiple minor bends. To assess the applicability of c_b^w, we first confirm that the parameter set found in our previous evaluation still applies for c_b^w. We then evaluate whether using c_b^w over c_b^u increases the compactness of our polygons. For this, we use the iso-perimetric quotient to quantify the compactness of our polygons. The iso-perimetric quotient for a polygon P is defined as $\gamma(P) = 4\pi\frac{\text{area}(P)}{\text{perimeter}(P)^2}$ (Osserman, 1978). γ is scale-invariant and ranges from 0 to 1. The most compact shape is a circle with $\gamma = 1$, while less compact shapes have a lower value.

Figure 10.11 shows the results of our parameter tuning when using c_b^w over c_b^u for the baseline solution. The baseline solution for c_b^w has 15 bends, an increase of 50% over the baseline solution obtained with c_b^u. Again, we argue that this increase is reasonable. Similar to the previous results, the individual costs for ρ_l, ρ_a, and ρ_d can be decreased even without increasing the number of bends. Additionally, the selected values from our previous parameter tuning still yield reasonable results, only increasing the number of bends by a maximum of 100%. Even though this is a more significant increase than with c_b^u, we argue that this is reasonable, as the actual number of bends is not the main optimization goal, in this case, anymore. Again, we analyze the generalisability of the result by applying the parameters to further instances. The results are shown in Figure 10.13 and are very similar to our findings for the uniform weighting of bends. This observation supports our claim that the choice of parameters generalizes very well. Figure 10.14 shows the resulting regions generated using c_b^u and c_b^w, respectively. The two results are similar in general appearance, only slightly differing at the corners. As expected, c_b^w creates more leveled corners than c_b^u. Compared to the baseline solution (Figure 10.9b), the issues with the polygon outline running close to existing lines or stations are resolved.

We evaluate the compactness of the resulting areas for all 13 test instances with our selected parameter set. For the uniform cost of bends, an average compactness of ≈ 0.45 is achieved while the compactness is increased to ≈ 0.49 when using the weighted cost of bends. The difference in the average compactness between the two approaches is minor. However, considering the earlier observation that the resulting regions are similar, this is unsurprising. In conclusion, the quantitative analysis between c_b^u and c_b^w does not allow for a clear recommendation

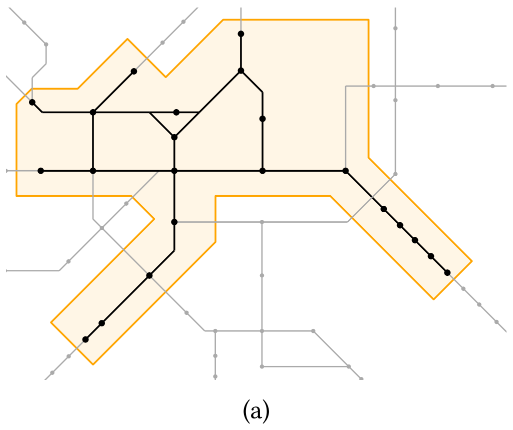


Figure 10.14: Results for the evaluation instance (a) using c_b^u and (b) using c_b^w.

towards one of the two optimization goals. The differences are minor and are possibly prone to the subjective assessment of the map reader. Therefore, we perform a qualitative analysis based on existing regions in transit maps.

Qualitative analysis. In most cases, transit maps display more than one region. One frequently encountered example is the display of fare zones. In this section, we perform a qualitative analysis by comparing results for fare zones generated with our approach with existing depictions of fare zones hand-drawn by a map designer. For our study, we use the transit map of Paris and the three central fare zones. Figure 10.15 shows the results of our study. The original transit map, together with the original depiction of the fare zones, is shown in Figure 10.15a. Figure 10.15b and Figure 10.15c show the results of our approach using c_b^u and c_b^w, respectively. We used the parameter set identified above (i.e., $\rho_b = 10$, $\rho_l = 50$, $\rho_a = 14$, $\rho_d = 1$). Remarkably, the original fare zones have much fewer bends than the solutions we generated. This implies that our selected ρ_b compared to the other weighting factors is too small for the given example. We increased ρ_b from 10 to 30 and recalculated the fare zones to verify this hypothesis. The results are displayed in Figure 10.15d. In particular, the outermost fare zone resembles the fare zones presented on the official map very closely. Our approach now has even fewer bends than the official version. Another observation is that the original fare zones only contain bend angles that are 45°. The same applies to our result generated with c_b^w.

In our approach, the single fare zones are generated individually. This has two negative implications. Firstly, this can result in overlaps of the individual zones as visible in the lower-left corner of Figure 10.15b (marked by the red circle). Secondly, this prevents inter-zone relationships from being considered, such as having equally spaced regions. Take the original fare zones as an example: they are more or less spaced equally and look similar to concentric circles. This is not the case for our generated examples. However, our approach can easily be extended to consider previously generated zones. For example, the zones could be generated starting from the innermost zone and in every iteration, the previously computed zone is treated as part of the transit network. That way, intersections are prevented. Additionally, appropri-

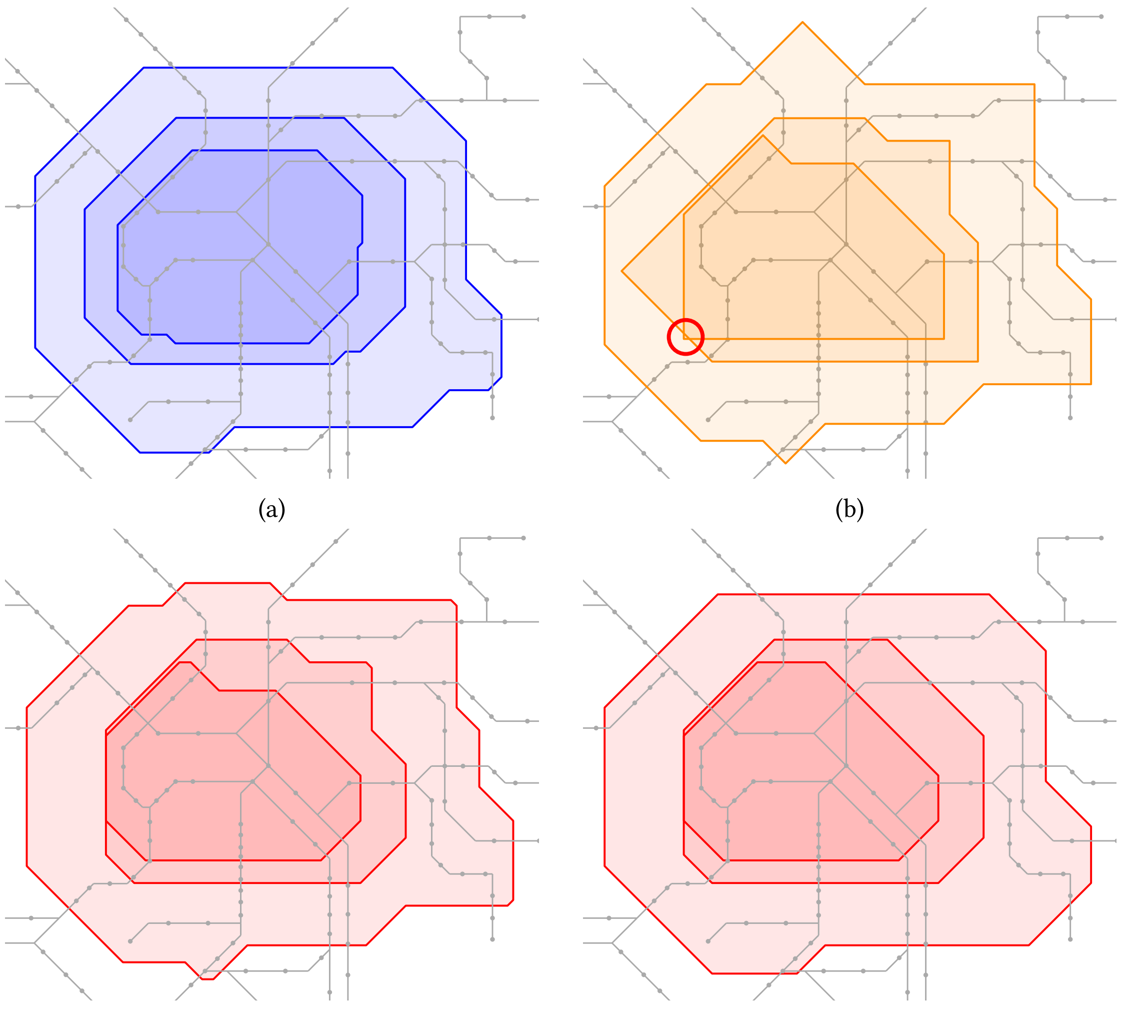


Figure 10.15: The three central fare zones of the transit network in Paris are shown, manually reproduced based on the official metro map (Régie Autonome des Transports Parisiens, 2023). (a) Fare zones of the official transit map. (b) Fare zones generated with our approach using c_b^u. (c) Fare zones generated with our approach using c_b^w. (d) Fare zones generated with our approach using c_b^w and increased $\rho_b = 30$.

ate cost factors could be added to favor equal spacing of the zones. These options should be explored in future work.

Summarizing the qualitative evaluation based on the transit map of Paris, we can verify that our approach creates zones that are very similar to the given transit map. Nonetheless, the parameter choice is crucial for generating a pleasing visualization. For the given example, a stronger focus on bend minimization compared to our found parameter set and using cost function c_b^w over c_b^u seem to yield the closest results to the given zones.

Evaluation summary. In this chapter, we outlined an empirical method for performing the parameter tuning based on a pairwise comparison of the optimization criteria. We applied this method to the transit network of Cologne, identifying $\rho_b = 10$, $\rho_l = 50$, $\rho_a = 14$, and $\rho_d = 1$ as a suitable parameter set. In a second experiment, we tested whether our approach could reproduce the hand-drawn fare zones for Paris's transit network. Our approach can generate visually pleasing results, even when using the parameters optimized for the transit network of Cologne (i.e., without performing the empirical parameter tuning for the network of Paris). A manual adaptation of the parameters is necessary to match the hand-drawn zones more closely. This method is suitable for fine-tuning the parameters thanks to the short computation times.

10.5 Conclusion

This chapter presented an automated method for generating schematized region boundaries in transit networks based on classifying stations into reachable and unreachable stations. The region boundaries created with our method correctly separate the stations into unreachable and reachable stations. Typical examples of such regions in transit networks are fare zones and isochrones, displaying the area that is reachable within a specific tariff or travel time, respectively. We use a grid-based approach that uses a pre-computed grid to generate a solution space for the generated region. We ensure the generated solution space contains a valid solution that correctly separates reachable and unreachable stations.

Enforcing the separation into reachable and unreachable parts only on the station level introduces an additional degree of freedom, which we use to optimize the regions towards popular design criteria, such as minimizing the number of bends and maximizing crossing angles with existing lines. These design criteria are incorporated into the approach by cost functions applied to the grid, allowing for a straightforward extension of the criteria. In an evaluation based on the transit network of Cologne, Germany, we study the influence of the different design criteria on the generated regions. The result shows that a suitable trade-off between the considered design criteria can be found. Nonetheless, when comparing the results generated with our approach to existing regions in official transit maps, two shortcomings of our approach become apparent. Firstly, while quantitatively yielding good results, the parameter choice we selected in the parameter tuning may not always create optimal results for a given instance. We argue that manually fine-tuning the parameters is not a problem, as computing times for a single region are within a few seconds. A second shortcoming is that in the current version, regions are generated independently of each other. In cases where multiple zones are overlayed on the same transit map, this can lead to unexpected results, such as overlaps in the individual zones. An easy fix for this issue could be incorporating previously generated regions into the computation of new regions. A more sophisticated solution could incorporate generating multiple zones at once. Next to preventing overlaps in the generated regions, this would also allow design criteria like an even spacing of regions or symmetry. Both of these approaches should be explored in future work.

This chapter analyzed the efficacy of the algorithm for generating isolines based on quantitative measures and a qualitative evaluation that was conducted. The question remains whether the produced visualizations are visually pleasing for a broader audience. An empirical study involving daily public transportation users should be conducted to verify the applicability of our approach in practice.

Conclusion & Future Work

In this thesis, algorithms for processing crowd-sourced trajectory data with a focus on analyzing the routing behavior of the users are presented. The algorithms are structured into three parts, each representing a component of a possible processing pipeline: preprocessing, analysis, and visualization. Working with crowd-sourced information provides many benefits, such as the availability of datasets not collected by authoritative sources or the inherent local perspective, offering insights into user behavior. However, the inherent heterogeneity of crowd-sourced data also introduces new challenges for subsequent analysis. In Section 1.1, the following challenges were identified for working with VGI trajectories:

1. **Unknown spatial quality**: The data is collected in an uncontrolled environment, so data accuracy is unknown. Similarly, there is no guarantee that the data is complete.
2. **Lack of metadata**: The data often lacks metadata, such as the user intent for recording the trajectory or the travel mode.
3. **Participation inequality**: The data does not represent all regions and demographic groups equally, resulting in a spatial and social bias. For some regions, no data will be given at all. Similarly, many individuals will only contribute few or no trajectories at all.
4. **Ethics**: The data can reveal sensitive information about the user, and the analysis can breach the users' privacy.

First, the thesis is concluded by reviewing the design choices implemented into the presented algorithms to address the challenges of crowd-sourced data. Then, potential directions for further research based on the presented work are discussed.

11.1 Conclusion

Chapter 4 presented an algorithm for matching the raw trajectory data to the road network, commonly called map matching. This is a crucial process for handling the heterogeneous spatial quality of the raw input. The energy-based approach adaptively manages measurement noise, with the flexibility to adjust energy terms for significant noise levels, ensuring reasonable matches. Further, modeling missing road segments addresses challenges related to incomplete road network data (which is often also based on crowd-sourced data). Last, explicitly modeling off-road movement alleviates constraints introduced by the map matching on user movement, which could otherwise skew analysis results.

Contrasting similar algorithms, the algorithm to infer routing preferences presented in Chapter 6 relaxes the constraint that a trajectory as a whole needs to be optimal for a specific routing preference. This allows handling all kinds of trajectories, even trips that did not aim at minimizing a specific objective, such as round trips. As such, missing metadata on the user intent is mitigated. Further, the algorithm neither requires complete coverage of the road network with trajectories nor multiple trajectories per user, thus addressing the participation inequality, especially the spatial bias. However, caution is required in interpreting the results of large-scale experiments due to potential social bias in the data, which leads to the results only representing a limited user group's perspective.

In Chapter 8, an algorithm computing glyphs for visualizing the off-screen evolution of trajectories is presented. As this algorithm works on raw trajectory input, an outlier ratio is introduced to account for data inacurracies. Additionally, this method facilitates the exploration of large trajectory datasets, aiding in identifying regions with varying data density and, hence, the spatial bias.

Chapters 9 and 10 apply schematic visualizations to communicate travel times and distances. Given the widespread use of schematic metro maps, it is reasonable to assume that most individuals recognize that geometric information presented in schematic visualizations should be treated cautiously. While not directly targeted at data accuracy, this effect is used to communicate the potential inaccuracies in the displayed travel times.

In conclusion, the methodological pipeline presented in this thesis comprehensively addresses most challenges associated with crowd-sourced trajectories. The only challenge not extensively targeted in a separate chapter is ethics. However, Chapter 3 presents several existing algorithms that can be applied to ensure user privacy. Furthermore, visualization methods allowing feedback to contributors enable them to benefit from their contributions, increasing user engagement and hopefully building a healthy VGI environment.

11.2 Future research

This thesis raised several research questions that could be followed up upon. The following discusses some directions for future research.

Empirical evaluation. Chapter 6 introduced an algorithm to infer the balance factor in a bicriterial routing model. The central motivation for developing this algorithm was to gain insight into the routing behavior of cyclists and pedestrians. While it was shown that the algorithm can be used to classify cyclists into specific routing archetypes, it remains unclear if the learned balance factor accurately reflects the routing preferences of the users. Empirical evaluations through controlled user studies are necessary to analyze this aspect. A challenge lies in the discrepancy between implicit and explicit routing preferences, as discussed in Section 5.2: users may behave differently from what they explicitly state as their preference. Therefore, any potential study should consider this and may even involve analyzing the relationship between these two preference types.

Multi-criterial routing models. The algorithm presented in Chapter 6 can efficiently infer the balance factor between two distinct routing criteria to characterize a given trajectory best. However, empirical studies have revealed that cyclists often consider more than two criteria when selecting a route (e.g., Casello and Usyukov, 2014). A simple approach to extend the bicriterial method presented in Chapter 6 to consider more than two criteria has been considered (Forsch, 2019). The method, sketched in Figure 11.1, integrates multiple criteria through an iterative process, merging two criteria based on the inferred balance factor between them. However, this approach is impractical due to long computation times, and the sequence in which the different criteria are integrated heavily influences the result.

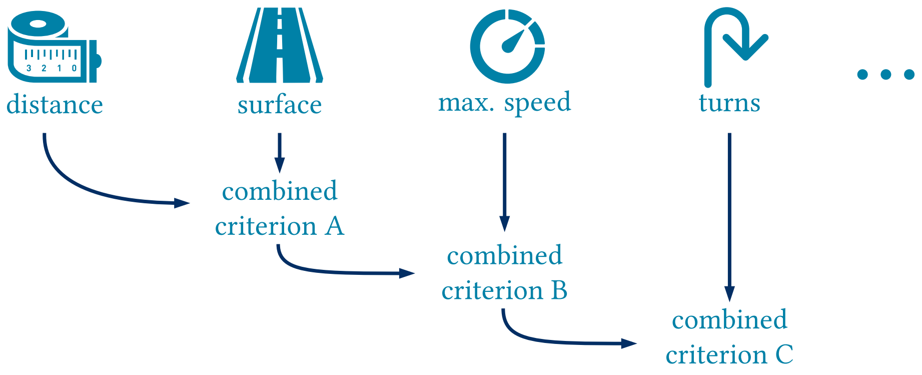


Figure 11.1: Iterative scheme to consider an arbitrary number of routing criteria. First, the balance factor for two initial criteria is learned and used to create a combined criterion. This combined criterion can iteratively be merged with additional criteria.

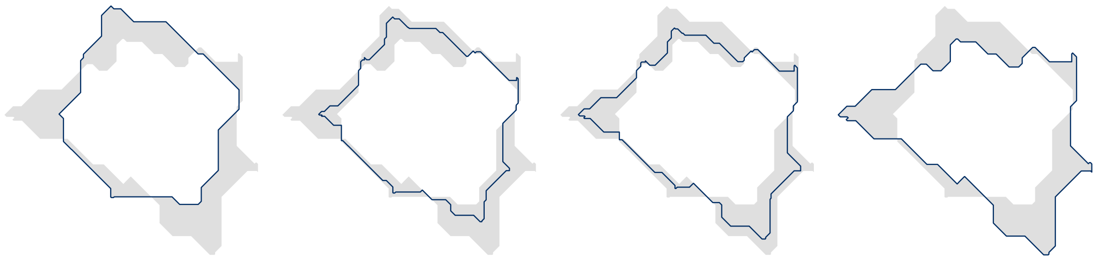
There exist algorithms for learning the balance factor in routing models with an arbitrary number of criteria, such as the algorithms presented by Funke et al. (2016) and Barth et al. (2021). However, these algorithms assume that the trajectory is optimal for at least one balance factor, a condition that often does not hold for real-world trajectories. To address such scenarios, the authors refer to trajectory segmentation approaches like the one presented in this thesis. Consequently, future research endeavors could combine these approaches to learn multi-criteria routing models efficiently.

Polygon simplification. This thesis aimed to gain insights into the routing behavior of cyclists by learning their trade-offs between different routing criteria. However, similar trade-offs commonly appear in other contexts as well.

One possible application is polygon simplification, which involves a trade-off between complexity reduction and shape preservation. Bonerath et al. (2023) introduce *shortcut hulls* for polygon simplification, where shortcut segments replace a chain of segments in the original polygon. Each shortcut e is assigned two costs: c_L specifying its complexity reduction and c_A quantifying its shape preservation. The shortcut hull without holes is computed using an optimal-path query in the shortcut graph with respect to the cost $c(e) = \lambda \cdot c_L(e) + (1-\lambda) \cdot c_A(e)$, where λ serves as a balance factor between shape preservation and complexity reduction.

The choice of λ relies on manual selection based on the application. Alternatively, the algorithm presented in Chapter 6 could infer an optimal balance factor based on existing simplifications. Professional cartographers' designs in existing maps provide a valuable input source for this purpose: numerous polygon simplifications are present in maps, such as variations in building representations across different zoom levels.

Morphing of isolines. Chapter 9 and Chapter 10 introduced algorithms for generating schematic isolines for reachability analysis. A common task performed in these analyses is comparing two map states, such as the state before and after implementing an infrastructure measure. Animated transitions between states can support detecting map changes but also add complexity to the visualization, which in extreme cases may overwhelm the user, thus limiting the information the user processes. Thus, animated maps should be generalized to the most important information to simplify the visualization.



(a) $t = 0.00$ (b) $t = 0.33$ (c) $t = 0.66$ (d) $t = 1.00$

Figure 11.2: Schematic morph animation between two routing-profile isolines. The intermediate polylines are schematic, self-intersection free, and contained in the area between the initial ($t = 0.00$) and final ($t = 1.00$) isoline (gray).

Ongoing research (Forsch et al., 2022b) deals with generating schematic morph animations for schematic isolines, as shown in Figure 11.2. To reduce the visual complexity, the intermediate shapes generated during the animation follow the same schematization as the input isolines, and self-intersections are prohibited. Additionally, the semantic meaning of the morph is preserved by requiring that the intermediate shapes stay between the two input isolines.

End-user-friendly analytics platform. Finally, it is crucial to advance the technical implementation of the algorithms further. This thesis has introduced a methodological pipeline for processing crowd-sourced trajectory data. Although implementations of the individual components are available as open-source code, these are primarily research prototypes designed for testing methods and performing experimental evaluations. To enable urban planners to enhance their decision-making processes, there is a need for an end-user-friendly analytics platform that seamlessly integrates these algorithms into a cohesive workflow. This platform could also be used for closing the loop with VGI contributors by providing feedback and increasing transparency in data processing, thereby improving user engagement. The provided implementations will serve as a solid starting point for this task.

Bibliography

P. Aggarwal, D. Thomas, L. Ojeda, and J. Borenstein. Map matching and heuristic elimination of gyro drift for personal navigation systems in GPS-denied conditions. *Measurement Science and Technology*, 22(2):025205, 2011. doi:10.1088/0957-0233/22/2/025205. 37

S. P. A. Alewijnse, K. Buchin, M. Buchin, A. Kölzsch, H. Kruckenberg, and M. A. Westenberg. A framework for trajectory segmentation by stable criteria. In *Proc. 22nd ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems (SIGSPATIAL'14)*, pages 351–360. ACM, 2014. doi:10.1145/2666310.2666415. 20, 60, 66

S. P. A. Alewijnse, K. Buchin, M. Buchin, S. Sijben, and M. A. Westenberg. Model-based segmentation and classification of trajectories. *Algorithmica*, 80(8):2422–2452, 2018. doi:10.1007/s00453-017-0329-x. 1, 60

L. O. Alvares, V. Bogorny, B. Kuijpers, J. A. F. de Macedo, B. Moelans, and A. Vaisman. A model for enriching trajectories with semantic geographical information. In *Proc. 15th Annual ACM International Symposium on Advances in Geographic Information Systems (GIS'07)*. ACM, 2007. doi:10.1145/1341012.1341041. 29

G. F. Anahid Basiri, Mordechai Haklay and P. Mooney. Crowdsourced geospatial data quality: challenges and future directions. *International Journal of Geographical Information Science*, 33(8):1588–1593, 2019. doi:10.1080/13658816.2019.1593422. 2

G. Andrienko and N. Andrienko. A general framework for using aggregation in visual exploration of movement data. *The Cartographic Journal*, 47(1):22–40, 2010. doi:10.1179/000870409X12525737905042. 90

N. Andrienko and G. Andrienko. Visual analytics of movement: An overview of methods, tools and procedures. *Information Visualization*, 12(1):3–24, 2013. doi:10.1177/1473871612457601. 1, 82

V. Antoniou and A. Skopeliti. Measures and indicators of VGI quality: An overview. *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, II-3/W5:345–351, 2015. doi:10.5194/isprsannals-II-3-W5-345-2015. 83

V. Antoniou, C. Capineri, and M. Haklay. VGI and beyond: From data to mapping. In *The Routledge Handbook of Mapping and Cartography*, pages 475–488. Routledge, 2017. 1

J. Arellana, M. Saltarín, A. M. Larrañaga, V. I. González, and C. A. Henao. Developing an urban bikeability index for different types of cyclists as a tool to prioritise bicycle infrastructure investments. *Transportation Research Part A: Policy and Practice*, 139:310–334, 2020. doi:10.1016/j.tra.2020.07.010. 53

B. Aronov, A. Driemel, M. V. Kreveld, M. Löffler, and F. Staals. Segmentation of trajectories on non-monotone criteria. *ACM Transactions on Algorithms*, 12(2):1–28, 2016. doi:10.1145/2660772. 1, 20, 21, 22, 60, 66

C. Bachechi, L. Po, and F. Rollo. Big data analytics and visualization in traffic monitoring. *Big Data Research*, 27:100292, 2022. doi:10.1016/j.bdr.2021.100292. 82

A. Balteanu, G. Jossé, and M. Schubert. Mining driving preferences in multi-cost networks. In M. A. Nascimento, T. Sellis, R. Cheng, J. Sander, Y. Zheng, H.-P. Kriegel, M. Renz, and C. Sengstock, editors, *Proc. 13th International Symposium on Advances in Spatial and Temporal Databases (SSTD'13)*, Lecture Notes in Computer Science (LNISA), pages 74–91, Berlin, Heidelberg, 2013. Springer. doi:10.1007/978-3-642-40235-7_5. 55

Y. Bang, J. Kim, and K. Yu. An improved map-matching technique based on the Fréchet distance approach for pedestrian navigation services. *Sensors*, 16(10):1768, 2016. doi:10.3390/s16101768. 37

M. Barragana, L. O. Alvares, and V. Bogorny. Unusual behavior detection and object ranking from movement trajectories in target regions. *International Journal of Geographical Information Science*, 31 (2):364–386, 2017. doi:10.1080/13658816.2016.1202415. 1

F. Barth, S. Funke, T. S. Jepsen, and C. Proissl. Scalable unsupervised multi-criteria trajectory segmentation and driving preference mining. In *Proc. 9th ACM SIGSPATIAL International Workshop on Analytics for Big Geospatial Data (BIGSPATIAL'20)*, pages 1–10. ACM, 2020. doi:10.1145/3423336.3429348. 55, 77

F. Barth, S. Funke, and C. Proissl. Preference-based trajectory clustering – an application of geometric hitting sets. In H.-K. Ahn and K. Sadakane, editors, *32nd International Symposium on Algorithms and Computation (ISAAC 2021)*, volume 212 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 15:1–15:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.ISAAC.2021.15. 55, 157

H. Bast, P. Brosi, and S. Storandt. Metro maps on octilinear grid graphs. *Computer Graphics Forum*, 39 (3):357–367, 2020. doi:10.1111/cgf.13986. 136

H. Bast, P. Brosi, and S. Storandt. Metro maps on flexible base grids. In *Proc. 17th International Symposium on Spatial and Temporal Databases (SSTD'21)*, pages 12–22. ACM, 2021. doi:10.1145/3469830.3470899. 136

P. Baudisch and R. Rosenholtz. Halo: A technique for visualizing off-screen objects. In *Proc. of the SIGCHI Conference on Human Factors in Computing Systems (CHI'03)*, pages 481–488. ACM, 2003. doi:10.1145/642611.642695. 91

V. Bauer, J. Gamper, R. Loperfido, S. Profanter, S. Putzer, and I. Timko. Computing isochrones in multi-modal, schedule-based transport networks. In *Proc. 16th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems (GIS'08)*, pages 1–2. ACM, 2008. doi:10.1145/1463434.1463524. 108, 110

M. Baum, V. Buchhold, J. Dibbelt, and D. Wagner. Fast exact computation of isochrones in road networks. In A. V. Goldberg and A. S. Kulikov, editors, *International Symposium on Experimental Algorithms*, Lecture Notes in Computer Science (LNTCS), pages 17–32. Springer International Publishing, 2016. doi:10.1007/978-3-319-38851-9_2. 136

M. Baum, T. Bläsius, A. Gemsa, I. Rutter, and F. Wegner. Scalable exact visualization of isocontours in road networks via minimum-link paths. *Journal of Computational Geometry*, 9(1):24–70, 2018. doi:10.20382/jocg.v9i1a2. 106, 109, 110, 112, 123, 124, 131, 137

M. Baum, V. Buchhold, J. Dibbelt, and D. Wagner. Fast exact computation of isocontours in road networks. *ACM Journal of Experimental Algorithmics*, 24, 2019. doi:10.1145/3355514. 108

L. Baur, S. Funke, and T. Rupp. Pathfindervis. In *Proc. 30th International Conference on Advances in Geographic Information Systems (SIGSPATIAL'22)*. ACM, 2022. doi:10.1145/3557915.3560990. 83

T. Behr, T. C. van Dijk, A. Forsch, J.-H. Haunert, and S. Storandt. Map matching for semi-restricted trajectories. In K. Janowicz and J. A. Verstegen, editors, *11th International Conference on Geographic Information Science (GIScience 2021) - Part II*, volume 208 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 12:1–12:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.GIScience.2021.II.12. 35

M. A. Bekos, B. Niedermann, and M. Nöllenburg. External labeling techniques: A taxonomy and survey. *Computer Graphics Forum*, 38(3):833–860, 2019. doi:10.1111/cgf.13729. 143

M. Ben-Akiva and M. Bierlaire. Discrete choice methods and their applications to short term travel decisions. In R. W. Hall, editor, *Handbook of Transportation Science*, pages 5–33. Springer, Boston, MA, 1999. doi:10.1007/978-1-4615-5203-1_2. 55

M. Ben-Akiva, M. S. Ramming, and S. Bekhor. Route choice models. In M. Schreckenberg and R. Selten, editors, *Human Behaviour and Traffic Networks*, pages 23–45, Berlin, Heidelberg, 2004. Springer. doi:10.1007/978-3-662-07809-9_2. 54, 55

M. Berg, O. Cheong, M. Kreveld, and M. Overmars. *Computational Geometry*. Springer, Berlin, Heidelberg, 3 edition, 2008. doi:10.1007/978-3-540-77974-2. 13

L. Bertolini. From "streets for traffic" to "streets for people": can street experiments transform urban mobility? *Transport Reviews*, 40(6):734–753, 2020. doi:10.1080/01441647.2020.1761907. 53

C. Bettini, X. S. Wang, and S. Jajodia. Protecting privacy against location-based personal identification. In W. Jonker and M. Petković, editors, *Secure Data Management*, pages 185–199, Berlin, Heidelberg, 2005. Springer. doi:10.1007/11552338_13. 31

P. Bimbao, J. Antonio, and S. J. Ou. #bikeparking now: Bike parking patterns in the landscape with instagram crowdsourced data. In *Proc. 2021 4th International Conference on Education Technology Management (ICETM'21)*, pages 291–296. ACM, 2022. doi:10.1145/3510309.3510354. 54

G. D. Birkhoff. *Aesthetic measure*. Harvard University Press, Cambridge, MA and London, England, 1933. doi:10.4159/harvard.9780674734470. 142

J. Blum, S. Funke, and S. Storandt. Sublinear search spaces for shortest path planning in grid and road networks. *Journal of Combinatorial Optimization*, 42(2):231–257, Aug. 2021. doi:10.1007/s10878-021-00777-3. 150

J. Blum, R. Li, and S. Storandt. Convexity hierarchies in grid networks. *Proc. International Conference on Automated Planning and Scheduling*, 33(1):52–60, July 2023. doi:10.1609/icaps.v33i1.27178. 150

W. Bohte and K. Maat. Deriving and validating trip purposes and travel modes for multi-day GPS-based travel surveys: A large-scale application in the netherlands. *Transportation Research Part C: Emerging Technologies*, 17(3):285–297, 2009. doi:10.1016/j.trc.2008.11.004. 1

J.-D. Boissonnat, O. Devillers, M. Teillaud, and M. Yvinec. Triangulations in CGAL. In *Proc. 16th Annual Symposium on Computational Geometry (SoCG'00)*, pages 11–18, 2000. 42

A. Bonerath, B. Niedermann, and J.-H. Haunert. Retrieving alpha-shapes and schematic polygonal approximations for sets of points within queried temporal ranges. In *Proc. 27th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems (SIGSPATIAL'19)*, pages 249–258. ACM, 2019. doi:10.1145/3347146.3359087. 110, 135

A. Bonerath, B. Niedermann, J. Diederich, Y. Orgeig, J. Oehrlein, and J.-H. Haunert. A time-windowed data structure for spatial density maps. In *Proc. 28th International Conference on Advances in Geographic Information Systems (SIGSPATIAL'20)*, pages 15–24. ACM, 2020. doi:10.1145/3397536.3422242. 83

A. Bonerath, J.-H. Haunert, J. S. Mitchell, and B. Niedermann. Shortcut hulls: Vertex-restricted outer simplifications of polygons. *Computational Geometry*, 112:101983, 2023. doi:10.1016/j.comgeo.2023.101983. 157

L. Bonomi and L. Xiong. A two-phase algorithm for mining sequential patterns with differential privacy. In *Proc. 22nd ACM International Conference on Information & Knowledge Management (CIKM'13)*, pages 269–278. ACM, 2013. doi:10.1145/2505515.2505553. 31

F. Boselie and E. Leeuwenberg. A general notion of beauty used to quantify the aesthetic attractivity of geometric forms. In *Cognitive Processes in the Perception of Art*, volume 19 of *Advances in Psychology*, pages 367–387. North-Holland, 1984. doi:10.1016/S0166-4115(08)62359-6. 142

Q. W. Bouts, T. Dwyer, J. Dykes, B. Speckmann, S. Goodwin, N. H. Riche, S. Carpendale, and A. Liebman. Visual encoding of dissimilarity data via topology-preserving map deformation. *IEEE Transactions on Visualization and Computer Graphics*, 22(9):2200–2213, 2016a. doi:10.1109/TVCG.2015.2500225. 110

Q. W. Bouts, I. Kostitsyna, M. van Kreveld, W. Meulemans, W. Sonke, and K. Verbeek. Mapping polygons to the grid with small Hausdorff and Fréchet distance. In *24th Annual European Symposium on Algorithms (ESA'16)*, volume 57 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 22:1–22:16. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2016b. doi:10.4230/LIPIcs.ESA.2016.22. 136

P. H. Bovy and S. Fiorenzo-Catalano. Stochastic route choice set generation: Behavioral and probabilistic foundations. *Transportmetrica*, 3(3):173–189, 2007. doi:10.1080/18128600708685672. 54

A. Brauer, V. Mäkinen, and J. Oksanen. Characterizing cycling traffic fluency using big mobile activity tracking data. *Comput. Environ. Urban Syst.*, 85:101553, 2021. doi:10.1016/j.compenvurbsys.2020.101553. 1, 4, 36

A. Brauer, V. Mäkinen, A. Forsch, J. Oksanen, and J.-H. Haunert. My home is my secret: concealing sensitive locations by context-aware trajectory truncation. *International Journal of Geographical Information Science*, 36(12):2496–2524, 2022. doi:10.1080/13658816.2022.2081694. 1, 30, 32

C. A. Brewer, L. V. Stanislawski, B. P. Buttenfield, K. A. Sparks, J. McGilloway, and M. A. Howard. Automated thinning of road networks and road labels for multiscale design of the national map of the united states. *Cartography and Geographic Information Science*, 40(4):259–270, 2013. doi:10.1080/15230406.2013.799735. 110

V. E. Brimkov, R. P. Barneva, and B. Brimkov. Connected distance-based rasterization of objects in arbitrary dimension. *Graphical Models*, 73(6):323–334, 2011. doi:10.1016/j.gmod.2011.06.002. 141

K. Buchin, A. van Goethem, M. Hoffmann, M. van Kreveld, and B. Speckmann. Travel-time maps: Linear cartograms with fixed vertex locations. In M. Duckham, E. Pebesma, K. Stewart, and A. U. Frank, editors, *Geographic Information Science*, pages 18–33, Cham, 2014. Springer. doi:10.1007/978-3-319-11593-1_2. 136

K. Buchin, W. Meulemans, A. V. Renssen, and B. Speckmann. Area-preserving simplification and schematization of polygonal subdivisions. *ACM Transactions on Spatial Algorithms and Systems*, 2 (1), 2016. doi:10.1145/2818373. 109, 135, 136

M. Buchin, A. Driemel, M. J. Van Kreveld, and V. Sacristan. Segmenting trajectories: A framework and algorithms using spatiotemporal criteria. *Journal of Spatial Information Science*, 3(3):33–63, 2011. doi:10.5311/JOSIS.2011.3.66. 20, 60, 67

M. Buchin, B. Kilgus, and A. Kölzsch. Group diagrams for representing trajectories. *International Journal of Geographical Information Science*, 34(12):2401–2433, 2020. doi:10.1080/13658816.2019.1684498. 37

D. Burghardt, C. Duchêne, and W. Mackaness, editors. *Abstracting Geographic Information in a Data Rich World*. Lecture Notes in Geoinformation and Cartography. Springer, Cham, 2014. doi:10.1007/978-3-319-00203-3. 135

D. Burghardt, E. Demidova, and D. A. Keim, editors. *Volunteered Geographic Information: Interpretation, Visualization and Social Context*. Springer, Cham, 2024. doi:10.1007/978-3-031-35374-1. 3

S. Burigat and L. Chittaro. Visualizing references to off-screen content on mobile devices: A comparison of arrows, wedge, and overview+detail. *Interacting with Computers*, 23(2):156–166, 02 2011. doi:10.1016/j.intcom.2011.02.005. 91

S. Burigat, L. Chittaro, and S. Gabrielli. Visualizing locations of off-screen objects on mobile devices: A comparative evaluation of three approaches. In *Proc. 8th Conference on Human-Computer Interaction with Mobile Devices and Services (MobileHCI'06)*, pages 239–246. ACM, 2006. doi:10.1145/1152215.1152266. 91

E. Cakmak, H. Schäfer, J. Buchmüller, J. Fuchs, T. Schreck, A. Jordan, and D. A. Keim. MotionGlyphs: Visual abstraction of spatio-temporal networks in collective animal behavior. *Computer Graphics Forum*, 39(3):63–75, 2020. doi:10.1111/cgf.13963. 90

L. Cao and J. Krumm. From GPS traces to a routable road map. In *Proc. 17th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems (GIS'09)*, pages 3–12. ACM, 2009. doi:10.1145/1653771.1653776. 28

M. B. Carmo, A. P. Afonso, and P. P. Matos. Visualization of geographic query results for small screen devices. In *Proc. 4th ACM workshop on Geographical information retrieval (GIR'07)*, pages 63–64. ACM, 2007. doi:10.1145/1316948.1316965. 90

J. M. Casello and V. Usyukov. Modeling cyclists' route choice based on GPS data. *Transportation Research Record*, 2430(1):155–161, 2014. doi:10.3141/2430-16. 157

A. Çöltekin, H. Janetzko, and S. I. Fabrikant. Geovisualization. *Geographic Information Science & Technology Body of Knowledge*, 2018(Q2), Apr. 2018. doi:10.22224/gistbok/2018.2.6. 81

P. Chao, Y. Xu, W. Hua, and X. Zhou. A survey on map-matching algorithms. In R. Borovica-Gajic, J. Qi, and W. Wang, editors, *Databases Theory and Applications*, Lecture Notes in Computer Science (LNISA), pages 121–133, Cham, 2020. Springer. doi:10.1007/978-3-030-39469-1_10. 37, 62

C. Chen, J. C. Anderson, H. Wang, Y. Wang, R. Vogt, and S. Hernandez. How bicycle level of traffic stress correlate with reported cyclist accidents injury severities: A geospatial and mixed logit analysis. *Accident Analysis & Prevention*, 108:234–244, 2017a. doi:10.1016/j.aap.2017.09.001. 53

D. Chen, D. Kim, L. Xie, M. Shin, A. K. Menon, C. S. Ong, I. Avazpour, and J. Grundy. Pathrec: Visual analysis of travel route recommendations. In *Proc. Eleventh ACM Conference on Recommender Systems (RecSys'17)*, pages 364–365. ACM, 2017b. doi:10.1145/3109859.3109983. 84

L. Chen, M. Lv, Q. Ye, G. Chen, and J. Woodward. A personal route prediction system based on trajectory data mining. *Information Sciences*, 181(7):1264–1284, 2011. doi:10.1016/j.ins.2010.11.035. 55

R. Chen, G. Acs, and C. Castelluccia. Differentially private sequential data publication via variable-length n-grams. In *Proc. 2012 ACM Conference on Computer and Communications Security (CCS'12)*, pages 638–649. ACM, 2012. doi:10.1145/2382196.2382263. 31

Y. Chen and H. Sundaram. Estimating complexity of 2D shapes. In *2005 IEEE 7th Workshop on Multimedia Signal Processing*, pages 1–4, 2005. doi:10.1109/MMSP.2005.248668. 123

M. Chimani, T. C. van Dijk, and J.-H. Haunert. How to eat a graph: Computing selection sequences for the continuous generalization of road networks. In *Proc. 22nd ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems (SIGSPATIAL'14)*, pages 243–252. ACM, 2014. doi:10.1145/2666310.2666414. 110

S. Cicerone and M. Cermignani. Fast and simple approach for polygon schematization. In B. Murgante, O. Gervasi, S. Misra, N. Nedjah, A. M. A. C. Rocha, D. Taniar, and B. O. Apduhan, editors, *Computational Science and Its Applications (ICCSA 2012)*, pages 267–279, Berlin, Heidelberg, 2012. Springer. doi:10.1007/978-3-642-31125-3_21. 136

S. Composto, J. Ingensand, M. Nappez, O. Ertz, D. Rappo, R. Bovard, I. Widmer, and S. Joost. How to recruit and motivate users to utilize VGI-systems? In *Proc. 19th AGILE International Conference on Geographic Information Science*, Helsinki, Finland, 2016. URL https://arodes.hes-so.ch/record/4671. 82

T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. MIT Press, Cambridge, MA, USA, 2009. 9, 11, 15, 17, 18, 66

B. Custers, M. van de Kerkhof, W. Meulemans, B. Speckmann, and F. Staals. Maximum physically consistent trajectories. In *Proc. 27th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems (SIGSPATIAL'19)*, pages 79–88. ACM, 2019. doi:10.1145/3347146.3359363. 28

B. Custers, W. Meulemans, B. Speckmann, and K. Verbeek. Coordinated schematization for visualizing mobility patterns on networks. In *Proc. 11th International Conference on GIS (GIScience'21) - Part II*, volume 208, pages 7:1–7:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.GIScience.2021.II.7. 90

C. E. de Rivera, L. L. Bliss-Ketchum, M. D. Lafrenz, A. V. Hanson, L. E. McKinney-Wise, A. H. Rodriguez, J. Schultz, A. L. Simmons, D. Taylor Rodriguez, A. H. Temple, and R. E. Wheat. Visualizing connectivity for wildlife in a world without roads. *Frontiers in Environmental Science*, 10, 2022. doi:10.3389/fenvs.2022.757954. 82

D. Delling, A. Gemsa, M. Nöllenburg, and T. Pajor. Path schematization for route sketches. In H. Kaplan, editor, *Algorithm Theory - SWAT 2010*, Lecture Notes in Computer Science (LNTCS), pages 285–296, Berlin, Heidelberg, 2010. Springer. doi:10.1007/978-3-642-13731-0_27. 135

D. Delling, A. Gemsa, M. Nöllenburg, T. Pajor, and I. Rutter. On d-regular schematization of embedded paths. *Computational Geometry*, 47(3, Part A):381–406, 2014. doi:10.1016/j.comgeo.2013.10.002. 135

U. Demšar, K. Buchin, F. Cagnacci, K. Safi, B. Speckmann, N. Van de Weghe, D. Weiskopf, and R. Weibel. Analysis and visualisation of movement: an interdisciplinary review. *Movement Ecology*, 3(1):1–24, 2015. doi:10.1186/s40462-015-0032-y. 82, 90

U. Demšar, J. A. Long, F. Benitez-Paez, V. B. Bastos, S. Marion, G. Martin, S. Sekulić, K. Smolak, B. Zein, and K. Siła-Nowicka. Establishing the integrated science of movement: bringing together concepts and methods from animal and human movement analysis. *International Journal of Geographical Information Science*, 35(7):1273–1308, 2021. doi:10.1080/13658816.2021.1880589. 82

J.-C. Denain and P. Langlois. Cartographie en anamorphose. *Mappemonde*, 49(1):16–19, 1998. doi:10.3406/mappe.1998.1347. 136

E. Di Giacomo, W. Didimo, G. Liotta, and H. Meijer. Area, curve complexity, and crossing resolution of non-planar graph drawings. *Theory of Computing Systems*, 49:565–575, 2011. doi:10.1007/s00224-010-9275-6. 142

E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1(1):269–271, 1959. doi:10.1007/BF01386390. 15, 55, 76, 136, 141

J. Dill and T. Carr. Bicycle commuting and facilities in major U.S. cities: If you build them, commuters will use them. *Transportation Research Record*, 1828(1):116–123, 2003. doi:10.3141/1828-14. 53

S. Dodge and E. Noi. Mapping trajectories and flows: facilitating a human-centered approach to movement data analytics. *Cartography and Geographic Information Science*, 48(4):353–375, 2021. doi:10.1080/15230406.2021.1913763. 82

A. Dunkel, M. Löchner, and D. Burghardt. Privacy-aware visualization of volunteered geographic information (vgi) to analyze spatial activity: A benchmark implementation. *ISPRS International Journal of Geo-Information*, 9(10), 2020. doi:10.3390/ijgi9100607. 3

C. Dwork. Differential privacy: A survey of results. In M. Agrawal, D. Du, Z. Duan, and A. Li, editors, *Theory and Applications of Models of Computation*, pages 1–19, Berlin, Heidelberg, 2008. Springer. doi:10.1007/978-3-540-79228-4_1. 31

J. Dykes, A. M. MacEachren, and M.-J. Kraak. Introduction exploring geovisualization. In J. Dykes, A. M. MacEachren, and M.-J. Kraak, editors, *Exploring Geovisualization*, International Cartographic Association, pages 1–19. Elsevier, Oxford, 2005. doi:10.1016/B978-008044531-1/50419-X. 81

P. Eades and D. Rappaport. The complexity of computing minimum separating polygons. *Pattern Recognition Letters*, 14(9):715–718, 1993. doi:10.1016/0167-8655(93)90140-9. Computational Geometry. 137

H. Edelsbrunner and F. Preparata. Minimum polygonal separation. *Information and Computation*, 77(3): 218–232, 1988. doi:10.1016/0890-5401(88)90049-1. 136

M. Ehrgott, J. Y. Wang, A. Raith, and C. van Houtte. A bi-objective cyclist route choice model. *Transportation Research Part A: Policy and Practice*, 46(4):652–663, 2012. doi:10.1016/j.tra.2011.11.015. 56

J. Eisner, S. Funke, A. Herbst, A. Spillner, and S. Storandt. Algorithms for matching and predicting trajectories. In *Proc. 13th Workshop on Algorithm Engineering and Experiments (ALENEX'11)*, pages 84–95, 2011. doi:10.1137/1.9781611972917.9. 37, 38

B. Elias. Pedestrian navigation – creating a tailored geodatabase for routing. In *2007 4th Workshop on Positioning, Navigation and Communication*, pages 41–47, 2007. doi:10.1109/WPNC.2007.353611. 38

C. R. Emond, W. Tang, and S. L. Handy. Explaining gender difference in bicycling behavior. *Transportation Research Record*, 2125(1):16–25, 2009. doi:10.3141/2125-03. 56

D. Eppstein and J. Erickson. Iterated nearest neighbors and finding minimal polytopes. *Discrete & Computational Geometry*, 11(3):321–350, 1994. doi:10.1007/BF02574012. 101

M. Ester, H.-P. Kriegel, J. Sander, and X. Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proc. Second International Conference on Knowledge Discovery and Data Mining (KDD'96)*, pages 226–231. AAAI Press, 1996. 30

H. J. Eysenck. The empirical determination of an aesthetic formula. *Psychological Review*, 48(1):83, 1941. doi:10.1037/h0062483. 142

L. Ferrari and M. Mamei. Identifying and understanding urban sport areas using Nokia Sports Tracker. *Pervasive and Mobile Computing*, 9(5):616–628, 2013. doi:10.1016/j.pmcj.2012.10.006. Special issue on Pervasive Urban Applications. 4

N. Ferreira, J. Poco, H. T. Vo, J. Freire, and C. T. Silva. Visual exploration of big spatio-temporal urban data: A study of New York City taxi trips. *IEEE Transactions on Visualization and Computer Graphics*, 19(12):2149–2158, 2013. doi:10.1109/TVCG.2013.226. 90

U. Feuerhake, C. Kuntzsch, and M. Sester. Finding interesting places and characteristic patterns in spatio-temporal trajectories. In *Proc. 8th International Symposium on Location-Based Services*, page 18, 2011. URL https://www.ikg.uni-hannover.de/fileadmin/ikg/Forschung/publications/lbs2011_feuerhake.pdf. 1, 60

J. A. W. Filho, W. Stuerzlinger, and L. Nedel. Evaluating an immersive space-time cube geovisualization for intuitive trajectory data exploration. *IEEE Transactions on Visualization and Computer Graphics*, 26(1):514–524, 2020. doi:10.1109/TVCG.2019.2934415. 91

G. Filomena and J. A. Verstegen. Modelling the effect of landmarks on pedestrian dynamics in urban environments. *Computers, Environment and Urban Systems*, 86:101573, 2021. doi:10.1016/j.compenvurbsys.2020.101573. 4

M. Fiore, P. Katsikouli, E. Zavou, M. Cunche, F. Fessant, D. Le Hello, U. M. Aivodji, B. Olivier, T. Quertier, and R. Stanica. Privacy in trajectory micro-data publishing: a survey. *Transactions on Data Privacy*, 13:91–149, 2020. URL https://inria.hal.science/hal-02968279. 30, 31

A. Forsch. Automatic learning of multi-criteria routing models from user-generated trajectories. Master's thesis, University of Bonn, 2019. 157

A. Forsch and J.-H. Haunert. Metrochrones: Schematic isochrones for schematic metro maps. *The Cartographic Journal*, -(-):–, 2023. doi:10.1080/00087041.2023.2284436. 133

A. Forsch, Y. Dehbi, B. Niedermann, J. Oehrlein, P. Rottmann, and J.-H. Haunert. Multimodal travel-time maps with formally correct and schematic isochrones. *Transactions in GIS*, 25(6):3233–3256, 2021. doi:10.1111/tgis.12821. 103, 134, 136, 137

A. Forsch, F. Amann, and J.-H. Haunert. Visualizing the off-screen evolution of trajectories. *KN - Journal of Cartography and Geographic Information*, 72(3):201–212, 2022a. doi:10.1007/s42489-022-00106-6. 87

A. Forsch, R. Kemna, E. Langetepe, and J.-H. Haunert. Morphing of schematized polygons for animated travel-time maps. In *3rd Schematic Mapping Workshop*, Bochum, apr 2022b. URL https://www.ruhr-uni-bochum.de/schematicmapping/papers/smw-fklh.pdf. Poster abstract. 131, 158

A. Forsch, J. Oehrlein, B. Niedermann, and J.-H. Haunert. Inferring routing preferences from user-generated trajectories using a compression criterion. *Journal of Spatial Information Science*, 26(5): 99–124, 2023. doi:10.5311/JOSIS.2023.26.256. 57

A. Forsch, S. Funke, J.-H. Haunert, and S. Storandt. Efficient mining of volunteered trajectory datasets. In D. Burghardt, E. Demidova, and D. A. Keim, editors, *Volunteered Geographic Information: Interpretation, Visualization and Social Context*, pages 43–77. Springer, Cham, 2024. doi:10.1007/978-3-031-35374-1_3. 3

M. L. Fredman and R. E. Tarjan. Fibonacci heaps and their uses in improved network optimization algorithms. *J. ACM*, 34(3):596–615, 1987. doi:10.1145/28869.28874. 17

M. Friese, M. Wänke, and H. Plessner. Implicit consumer preferences and their influence on product choice. *Psychology & Marketing*, 23(9):727–740, 2006. doi:10.1002/mar.20126. 54

S. Fuest and M. Sester. A framework for automatically visualizing and recommending efficient routes. *Proceedings of the ICA*, 2:34, 2019. doi:10.5194/ica-proc-2-34-2019. 84

S. Fuest, S. Grüner, M. Vollrath, and M. Sester. Evaluating the effectiveness of different cartographic design variants for influencing route choice. *Cartography and Geographic Information Science*, 48(2): 169–185, 2021. doi:10.1080/15230406.2020.1855251. 84

S. Funke and S. Storandt. Polynomial-time construction of contraction hierarchies for multi-criteria objectives. In *2013 Proc. Meeting on Algorithm Engineering and Experiments (ALENEX'13)*, pages 41–54, 2013. doi:10.1137/1.9781611972931.4. 63, 65, 77

S. Funke, S. Laue, and S. Storandt. Deducing individual driving preferences for user-aware navigation. In S. Ravada, M. E. Ali, S. D. Newsam, M. Renz, and G. Trajcevski, editors, *Proc. 24th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems (SIGSPATIAL'16)*, pages 1–9. ACM, 2016. doi:10.1145/2996913.2997004. 36, 55, 157

S. Funke, S. Laue, and S. Storandt. Personal routes with high-dimensional costs and dynamic approximation guarantees. In C. S. Iliopoulos, S. P. Pissis, S. J. Puglisi, and R. Raman, editors, *16th International Symposium on Experimental Algorithms (SEA 2017)*, volume 75 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 18:1–18:13, Dagstuhl, Germany, 2017. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.SEA.2017.18. 76, 77

S. Funke, T. Rupp, A. Nusser, and S. Storandt. PATHFINDER: storage and indexing of massive trajectory sets. In *Proc. 16th International Symposium on Spatial and Temporal Databases (SSTD'19)*, pages 90–99, 2019. doi:10.1145/3340964.3340978. 36, 37, 61

M. D. L. Galvão, J. Krukar, M. Nöllenburg, and A. Schwering. Route schematization with landmarks. *Journal of Spatial Information Science*, 21(11):99–136, 2020. doi:10.5311/JOSIS.2020.21.589. 136

J. Gamper, M. Böhlen, W. Cometti, and M. Innerebner. Defining isochrones in multimodal spatial networks. In *Proc. 20th ACM International Conference on Information and Knowledge Management (CIKM'11)*, pages 2381–2384. ACM, 2011. doi:10.1145/2063576.2063972. 106, 108

J. Gamper, M. H. Böhlen, and M. Innerebner. Scalable computation of isochrones with network expiration. In *Scientific and Statistical Database Management (SSDBM'12)*, volume 7338 of *Lecture Notes in Computer Science (LNCS)*, pages 526–543. Springer, 2012. doi:10.1007/978-3-642-31235-9_35. 108

S. R. Ganta, S. P. Kasiviswanathan, and A. Smith. Composition attacks and auxiliary information in data privacy. In *Proc. 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'08)*, pages 265–273. ACM, 2008. doi:10.1145/1401890.1401926. 31

K. Garland. *Mr Beck's underground map*. Capital Transport, London, England, 1994. 135

S. Gedicke, A. Jabrayilov, B. Niedermann, P. Mutzel, and J.-H. Haunert. Point feature label placement for multi-page maps on small-screen devices. *Computers & Graphics*, 100:66–80, 2021. doi:10.1016/j.cag.2021.07.019. 90

R. Geisberger, P. Sanders, D. Schultes, and C. Vetter. Exact routing in large road networks using contraction hierarchies. *Transportation Science*, 46(3):388–404, 2012. doi:10.1287/trsc.1110.0401. 61

A. Ghermandi and M. Sinclair. Passive crowdsourcing of social media in environmental research: A systematic map. *Global Environmental Change*, 55:36–47, 2019. doi:10.1016/j.gloenvcha.2019.02.003. 3

J.-P. Gómez-Barrón, M.-A. Manso-Callejo, R. Alcarria, and T. Iturrioz. Volunteered geographic information system design: Project and participation guidelines. *ISPRS International Journal of Geo-Information*, 5(7), 2016. doi:10.3390/ijgi5070108. 82

J.-P. Gómez-Barrón, M.-Á. Manso-Callejo, and R. Alcarria. Needs, drivers, participants and engagement actions: A framework for motivating contributions to volunteered geographic information systems. *Journal of Geographical Systems*, 21(1):5–41, 2019. doi:10.1007/s10109-018-00289-5. 82

H. Gong, C. Chen, E. Bialostozky, and C. T. Lawson. A GPS/GIS method for travel mode detection in New York City. *Computers, Environment and Urban Systems*, 36(2):131–139, 2012. doi:10.1016/j.compenvurbsys.2011.05.003. Special Issue: Geoinformatics 2010. 1

L. Gong, H. Sato, T. Yamamoto, T. Miwa, and T. Morikawa. Identification of activity stop locations in GPS trajectories by density-based clustering method combined with support vector machines. *Journal of Modern Transportation*, 23:202–213, 2015. doi:10.1007/s40534-015-0079-x. 30

M. F. Goodchild. Citizens as sensors: the world of volunteered geography. *GeoJournal*, 69(4):211–221, 2007. doi:10.1007/s10708-007-9111-y. 1, 59, 88

R. Gotsman and Y. Kanza. Compact representation of GPS trajectories over vectorial road networks. In *Advances in Spatial and Temporal Databases*, Lecture Notes in Computer Science (LNISA), pages 241–258, Berlin, Heidelberg, 2013. Springer. doi:10.1007/978-3-642-40235-7_14. 59

A. G. Greenwald and M. R. Banaji. Implicit social cognition: attitudes, self-esteem, and stereotypes. *Psychological review*, 102(1):4–27, 1995. doi:10.1037/0033-295x.102.1.4. 54

E. Grigore, N. Garrick, R. Fuhrer, and I. K. W. Axhausen. Bikeability in Basel. *Transportation Research Record*, 2673(6):607–617, 2019. doi:10.1177/0361198119839982. 54

P. D. Grünwald, I. J. Myung, and M. A. Pitt. *Advances in minimum description length: Theory and applications*. Neural Information Processing. MIT press, 2005. doi:10.7551/mitpress/1114.001.0001. 59

M. E. Gursoy, L. Liu, S. Truex, and L. Yu. Differentially private and utility preserving publication of trajectory data. *IEEE Transactions on Mobile Computing*, 18(10):2315–2329, 2019. doi:10.1109/TMC.2018.2874008. 31

S. Gustafson, P. Baudisch, C. Gutwin, and P. Irani. Wedge: Clutter-free visualization of off-screen locations. In *Proc. SIGCHI Conference on Human Factors in Computing Systems (CHI'08)*, pages 787–796. ACM, 2008. doi:10.1145/1357054.1357179. 91

A. Guttman. R-trees: a dynamic index structure for spatial searching. *SIGMOD Rec.*, 14(2):47–57, 1984. doi:10.1145/971697.602266. 61

T. Hägerstrand. What about people in regional science? *Papers of the Regional Science Association*, 24 (1):7–24, 1970. doi:10.1007/BF01936872. 90

M. Haklay. How good is volunteered geographical information? A comparative study of OpenStreetMap and ordnance survey datasets. *Environment and Planning B: Planning and Design*, 37(4):682–703, 2010. doi:10.1068/b35097. 3

M. Haklay. Why is participation inequality important? In *European Handbook of Crowdsourced Geographic Information*. Ubiquity Press, 2016. 3

K. Halldórsdóttir, N. Rieser-Schüssler, K. W. Axhausen, O. A. Nielsen, and C. G. Prato. Efficiency of choice set generation methods for bicycle routes. *European Journal of Transport and Infrastructure Research*, 14(4), Sep. 2014. doi:10.18757/ejtir.2014.14.4.3040. 55

M. Hardinghaus and S. Nieland. Assessing cyclists' routing preferences by analyzing extensive user setting data from a bike-routing engine. *European Transport Research Review*, 13(1):1–19, 2021. doi:10.1186/s12544-021-00499-x. 56

M. Hardinghaus and J. Weschke. Attractive infrastructure for everyone? Different preferences for route characteristics among cyclists. *Transportation Research Part D: Transport and Environment*, 111:103465, 2022. doi:10.1016/j.trd.2022.103465. 54, 56

M. Hardinghaus, S. Nieland, M. Lehne, and J. Weschke. More than bike lanes – a multifactorial index of urban bikeability. *Sustainability*, 13(21), 2021. doi:10.3390/su132111584. 54

D. L. Harkey, D. W. Reinfurt, and M. Knuiman. Development of the bicycle compatibility index. *Transportation Research Record*, 1636(1):13–20, 1998. doi:10.3141/1636-03. 54

J.-H. Haunert and B. Budig. An algorithm for map matching given incomplete road data. In *Proc. 20th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems (SIGSPATIAL'12)*, pages 510–513. ACM, 2012. doi:10.1145/2424321.2424402. 37, 38

J.-H. Haunert and A. Wolff. Optimal and topologically safe simplification of building footprints. In *Proc. 18th SIGSPATIAL International Conference on Advances in Geographic Information Systems (GIS'10)*, pages 192–201. ACM, 2010. doi:10.1145/1869790.1869819. 110

S. He, F. Bastani, S. Abbar, M. Alizadeh, H. Balakrishnan, S. Chawla, and S. Madden. RoadRunner: improving the precision of road network inference from GPS trajectories. In *Proc. 26th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems (ACM SIGSPATIAL GIS'18)*, pages 3–12, 2018. doi:10.1145/3274895.3274974. 37

J. Henriksen-Bulmer and S. Jeary. Re-identification attacks – a systematic literature review. *International Journal of Information Management*, 36(6, Part B):1184–1192, 2016. doi:10.1016/j.ijinfomgt.2016.08.002. 30

S.-H. Hong, D. Merrick, and H. A. do Nascimento. Automatic visualisation of metro maps. *Journal of Visual Languages & Computing*, 17(3):203–224, 2006. doi:10.1016/j.jvlc.2005.09.001. 142

S. Hoogendoorn-Lanser, R. van Nes, and P. Bovy. A rule-based approach to route choice set generation. In *11th International Conference on Travel Behaviour Research, Kyoto, Japan*. Citeseer, 2006. URL https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=639e2b06df40ad0d99ea050af0e40ced4b096903. 54

H. Huang. Anomalous behavior detection in single-trajectory data. *International Journal of Geographical Information Science*, 29(12):2075–2094, 2015. doi:10.1080/13658816.2015.1063640. 1

R. Huang, C. Huang, J. Shan, L. Xiong, and J. Yan. Evaluation of GPS trajectories on VGI and social websites. In *2013 21st International Conference on Geoinformatics*, pages 1–5. IEEE, 2013. doi:10.1109/Geoinformatics.2013.6626121. 88

W. Huang, S.-H. Hong, and P. Eades. Effects of crossing angles. In *2008 IEEE Pacific Visualization Symposium*, pages 41–46. IEEE, 2008. doi:10.1109/PACIFICVIS.2008.4475457. 110, 142

W. Huang, P. Eades, and S.-H. Hong. Larger crossing angles make graphs easier to read. *Journal of Visual Languages & Computing*, 25(4):452–465, 2014. doi:10.1016/j.jvlc.2014.03.001. 142

A. Hull and C. O'Holleran. Bicycle infrastructure: can good design encourage cycling? *Urban, Planning and Transport Research*, 2(1):369–406, 2014. doi:10.1080/21650020.2014.955210. 53

Z. Huo, X. Meng, H. Hu, and Y. Huang. You can walk alone: Trajectory privacy-preserving through significant stays protection. In S.-g. Lee, Z. Peng, X. Zhou, Y.-S. Moon, R. Unland, and J. Yoo, editors, *Database Systems for Advanced Applications*, pages 351–366, Berlin, Heidelberg, 2012. Springer. doi:10.1007/978-3-642-29038-1_26. 31

H. Imai and M. Iri. Polygonal approximations of a curve – formulations and algorithms. In *Computational Morphology*, volume 6 of *Machine Intelligence and Pattern Recognition*, pages 71–86. North-Holland, 1988. doi:10.1016/B978-0-444-70467-2.50011-4. 110

S. Ivanovic, A.-M. Olteanu-Raimond, S. Mustiere, and T. Devogele. Detection of outliers in crowdsourced GPS traces. In *Spatial Accuracy 2016 Symposium*, Montpellier, France, July 2016. URL https://hal.science/hal-01505059. 3, 28

S. Ivanovic, A.-M. Olteanu-Raimond, S. Mustière, and T. Devogele. A filtering-based approach for improving crowdsourced GNSS traces in a data update context. *ISPRS International Journal of Geo-Information*, 8(9), 2019. doi:10.3390/ijgi8090380. 28

G. R. Jagadeesh and T. Srikanthan. Fast computation of clustered many-to-many shortest paths and its application to map matching. *ACM Transactions on Spatial Algorithms and Systems*, 5(3):1–20, 2019. doi:10.1145/3329676. 37

L. Jiang, C. Chen, C. Chen, H. Huang, and B. Guo. From driving trajectories to driving paths: a survey on map-matching algorithms. *CCF Transactions on Pervasive Computing and Interaction*, pages 1–16, 2022. doi:10.1007/s42486-022-00101-w. 62

V. Jokinen, V. Mäkinen, A. Brauer, and J. Oksanen. Would citizens contribute their personal location data to an open database? preliminary results from a survey. In *16th International Conference on Location Based Services*, pages 171–176, 2021. URL https://biblio.ugent.be/publication/8740239/file/8740242.pdf. 4

J. Jun, R. Guensler, and J. H. Ogle. Smoothing methods to minimize impact of global positioning system random error on travel distance, speed, and acceleration profile estimates. *Transportation Research Record*, 1972(1):141–150, 2006. doi:10.1177/0361198106197200117. 28

N. Kami, N. Enomoto, T. Baba, and T. Yoshikawa. Algorithm for detecting significant locations from raw GPS data. In B. Pfahringer, G. Holmes, and A. Hoffmann, editors, *Discovery Science*, pages 221–235, Berlin, Heidelberg, 2010. Springer. doi:10.1007/978-3-642-16184-1_16. 30

A.-S. Karakaya, J. Hasenburg, and D. Bermbach. SimRa: Using crowdsourcing to identify near miss hotspots in bicycle traffic. *Pervasive and Mobile Computing*, 67:101197, 2020. doi:10.1016/j.pmcj.2020.101197. 54

A. Kealy and T. Moore. Land and maritime applications. In *Springer Handbook of Global Navigation Satellite Systems*, pages 841–875. Springer, Cham, 2017. doi:10.1007/978-3-319-42928-1_29. 3, 28

G. Kellaris, N. Pelekis, and Y. Theodoridis. Map-matched trajectory compression. *Journal of Systems and Software*, 86(6):1566–1579, 2013. doi:10.1016/j.jss.2013.01.071. 61

D. K. Kellstedt, J. O. Spengler, M. Foster, C. Lee, and J. E. Maddock. A scoping review of bikeability assessment methods. *Journal of Community Health*, 46(1):211–224, Feb 2021. doi:10.1007/s10900-020-00846-4. 53

M. Kent and A. Karner. Prioritizing low-stress and equitable bicycle networks using neighborhood-based accessibility measures. *International Journal of Sustainable Transportation*, 13(2):100–110, 2019. doi:10.1080/15568318.2018.1443177. 54

L. Knapen, I. B.-A. Hartman, D. Schulz, T. Bellemans, D. Janssens, and G. Wets. Determining structural route components from GPS traces. *Transportation Research Part B: Methodological*, 90:156–171, 2016. doi:10.1016/j.trb.2016.04.019. 55, 60, 67, 78

L. Knapen, I. B.-A. Hartman, and T. Bellemans. Using path decomposition enumeration to enhance route choice models. *Future Generation Computer Systems*, 107:1077–1088, 2020. doi:10.1016/j.future.2017.12.053. 61

M. Knura, F. Kluger, M. Zahtila, J. Schiewe, B. Rosenhahn, and D. Burghardt. Using object detection on social media images for urban bicycle infrastructure planning: A case study of dresden. *ISPRS International Journal of Geo-Information*, 10(11), 2021. doi:10.3390/ijgi10110733. 54

H. Koller, P. Widhalm, M. Dragaschnig, and A. Graser. Fast hidden Markov model map-matching for sparse and noisy trajectories. In *Proc. 18th IEEE International Conference on Intelligent Transportation Systems (ITSC'15)*, pages 2557–2561. IEEE, 2015. doi:10.1109/ITSC.2015.411. 37

S. Korpilo, T. Virtanen, and S. Lehvävirta. Smartphone GPS tracking – inexpensive and efficient data collection on recreational movement. *Landscape and Urban Planning*, 157:608–617, 2017. doi:10.1016/j.landurbplan.2016.08.005. 3

M.-J. Kraak. The space-time cube revisited from a geovisualization perspective. In *Proc. 21st International Cartographic Conference*, pages 1988–1996, 2003. URL https://www.icaci.org/files/documents/ICC_proceedings/ICC2003/Papers/255.pdf. 82, 91

D. Krajzewicz and D. Heinrichs. UrMo accessibility computer – a tool for computing contour accessibility measures. In *System Simulation (SIMUL'16)*, pages 56–60. IARIA, 2016. URL https://elib.dlr.de/106451/. Visited on March 20, 2023. 109, 110

P. J. Krenn, P. Oja, S. Titze, et al. Development of a bikeability index to assess the bicycle-friendliness of urban environments. *Open Journal of Civil Engineering*, 5(04):451–459, 2015. doi:10.4236/ojce.2015.54045. 54

N. Krismer, G. Specht, and J. Gamper. Incremental calculation of isochrones regarding duration. In *Proc. 26th GI-Workshop Grundlagen von Datenbanken*, volume 1313 of *CEUR Workshop Proceedings*, pages 41–45, Bonn, Germany, 2014. Gesellschaft für Informatik. URL https://ceur-ws.org/Vol-1313/paper_8.pdf. Visited on March 20, 2023. 108

N. Krismer, D. Silbernagl, G. Specht, and J. Gamper. Computing isochrones in multimodal spatial networks using tile regions. In *Proc. 29th International Conference on Scientific and Statistical Database Management (SSDBM'17)*, pages 1–6. ACM, 2017. doi:10.1145/3085504.3085538. 106, 108

K. J. Krizek, A. El-Geneidy, and K. Thompson. A detailed analysis of how an urban trail system affects cyclists' travel. *Transportation*, 34(5):611–624, Sep 2007. doi:10.1007/s11116-007-9130-z. 54

B. Krogh, C. S. Jensen, and K. Torp. Efficient in-memory indexing of network-constrained trajectories. In *Proc. 24th ACM SIGSPATIAL international conference on advances in geographic information systems (ACM SIGSPATIAL GIS'16)*, pages 1–10, 2016. doi:10.1145/2996913.2996972. 36, 37, 61

B. W. Landis. Bicycle interaction hazard score: a theoretical model. *Transportation research record*, 1438:3–8, 1994. URL https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=75a7d62a9d3eb36fed9a79cadda78829fa3ad8a2. 54

B. W. Landis, V. R. Vattikuti, and M. T. Brannick. Real-time human perceptions: Toward a bicycle level of service. *Transportation Research Record*, 1578(1):119–126, 1997. doi:10.3141/1578-15. 53

P. Laube. *Computational Movement Analysis.* Springer Briefs in Computer Science. Springer, Cham, 2014. doi:10.1007/978-3-319-10268-9. 1

W.-C. Lee and J. Krumm. Trajectory preprocessing. In *Computing with Spatial Trajectories*, pages 3–33. Springer, New York, NY, USA, 2011. doi:10.1007/978-1-4614-1629-6_1. 28

P. M. Lerin, D. Yamamoto, and N. Takahashi. Encoding network–constrained travel trajectories using routing algorithms. *International Journal of Knowledge and Web Intelligence*, 4(1):34–49, 2013. doi:10.1504/IJKWI.2013.052724. 59

D. Li, J. Li, and J. Li. Road network extraction from low-frequency trajectories based on a road structure-aware filter. *ISPRS International Journal of Geo-Information*, 8(9):374, 2019. doi:10.3390/ijgi8090374. 37

J. Li, Z. Xiao, and J. Kong. A viewpoint based approach to the visual exploration of trajectory. *Journal of Visual Languages & Computing*, 41:41–53, 2017. doi:10.1016/j.jvlc.2017.04.001. 83

L. Li, R. Jiang, Z. He, X. M. Chen, and X. Zhou. Trajectory data-based traffic flow studies: A revisit. *Transportation Research Part C: Emerging Technologies*, 114:225–240, 2020. doi:10.1016/j.trc.2020.02.016. 1

N. Li, T. Li, and S. Venkatasubramanian. t-closeness: Privacy beyond k-anonymity and l-diversity. In *2007 IEEE 23rd International Conference on Data Engineering*, pages 106–115, 2007. doi:10.1109/ICDE.2007.367856. 31

Q. Li, Y. Zheng, X. Xie, Y. Chen, W. Liu, and W.-Y. Ma. Mining user similarity based on location history. In *Proc. 16th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems (GIS'08)*, pages 1–10. ACM, 2008. doi:10.1145/1463434.1463477. 60

R. Li and J. Zhao. Off-screen landmarks on mobile devices: Levels of measurement and the perception of distance on resized icons. *KI - Künstliche Intelligenz*, 31(2):141–149, 2017. doi:10.1007/s13218-016-0471-7. 91

H. Liu, Y. Gao, L. Lu, S. Liu, H. Qu, and L. M. Ni. Visual analysis of route diversity. In *2011 IEEE Conference on Visual Analytics Science and Technology (VAST)*, pages 171–180, 2011. doi:10.1109/VAST.2011.6102455. 84

M. Löffler and W. Meulemans. Discretized approaches to schematization. In J. Gudmundsson and M. H. M. Smid, editors, *Proc. 29th Canadian Conference on Computational Geometry (CCCG)*, pages 220–225, Ottawa, Ontario, Canada, 2017. URL https://research.tue.nl/en/publications/discretized-approaches-to-schematization. 136

Y. Lou, C. Zhang, Y. Zheng, X. Xie, W. Wang, and Y. Huang. Map-matching for low-sampling-rate GPS trajectories. In *Proc. 17th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems (ACM SIGSPATIAL GIS'09)*, pages 352–361. ACM, 2009. doi:10.1145/1653771.1653820. 36

M. B. Lowry, D. Callister, M. Gresham, and B. Moore. Assessment of communitywide bikeability with bicycle level of service. *Transportation Research Record*, 2314(1):41–48, 2012. doi:10.3141/2314-06. 53

M. Lu, C. Lai, T. Ye, J. Liang, and X. Yuan. Visual analysis of multiple route choices based on general GPS trajectories. *IEEE Transactions on Big Data*, 3(2):234–247, 2017. doi:10.1109/TBDATA.2017.2667700. 84

A. M. MacEachren. Compactness of geographic shape: Comparison and evaluation of measures. *Geografiska Annaler: Series B, Human Geography*, 67(1):53–67, 1985. doi:10.1080/04353684.1985.11879515. 142

A. M. MacEachren. Visualization in modern cartography: Setting the agenda. In *Modern Cartography Series*, pages 1–12. Elsevier, 1994. doi:10.1016/b978-0-08-042415-6.50008-9. 81

A. M. MacEachren and M.-J. Kraak. Research challenges in geovisualization. *Cartography and Geographic Information Science*, 28(1):3–12, 2001. doi:10.1559/152304001782173970. 81

A. Machanavajjhala, D. Kifer, J. Gehrke, and M. Venkitasubramaniam. L-diversity: Privacy beyond k-anonymity. *ACM Trans. Knowl. Discov. Data*, 1(1):3–es, 2007. doi:10.1145/1217299.1217302. 31

B. B. Majumdar, S. Mitra, and P. Pareekh. Methodological framework to obtain key factors influencing choice of bicycle as a mode. *Transportation research record*, 2512(1):110–121, 2015. doi:10.3141/2512-13. 56

B. Manum, T. Nordström, J. Gil, L. Nilsson, and L. Marcus. Modelling bikeability. In *Proc. 11th International Space Syntax Symposium, Lisbon, Portugal*, pages 3–7, 2017. URL https://www.academia.edu/download/53836190/89_Manum_al_ModellingBikeability.pdf. 54

S. Marciuska and J. Gamper. Determining objects within isochrones in spatial network databases. In *Proc. 14th East European Conference on Advances in Databases and Information Systems (ADBIS'10)*, volume 6295 of *Lecture Notes in Computer Science (LNCS)*, pages 392–405. Springer, 2010. doi:10.1007/978-3-642-15576-5_30. 109, 110, 123

N. Markovic, S. Miller, Z. V. Laan, and Y. Wang. Visual exploration of Utah trajectory data and their applications in transportation. Technical report, TREC at Portland State University, 2020. URL https://rosap.ntl.bts.gov/view/dot/64589. 91

A. D. Marra and F. Corman. A deep learning model for predicting route choice in public transport. In *21st Swiss Transport Research Conference (STRC'21)*. STRC, 2021. doi:10.3929/ethz-b-000504159. 56

J. D. Mazimpaka and S. Timpf. Trajectory data mining: A review of methods and applications. *Journal of Spatial Information Science*, 13(4), Dec. 2016. doi:10.5311/josis.2016.13.263. 1

R. McCall, V. Koenig, R. Martin, and T. Engel. Changing mobility behaviour through recommendations. In *Proc. ACM Recsys CrowdRec Workshop 2015*, 2015. URL https://hdl.handle.net/10993/39663. 84

F. G. Meddah. A novel methodology for geovisualizing epidemiological data. In B. Lejdel, E. Clementini, and L. Alarabi, editors, *Artificial Intelligence and Its Applications*, pages 557–564, Cham, 2022. Springer. doi:10.1007/978-3-030-96311-8_52. 82

K. Mehlhorn and M. Ziegelmann. Resource constrained shortest paths. In M. S. Paterson, editor, *Algorithms - ESA 2000*, pages 326–337. Springer, 2000. 63

Melbourne and Metropolitan Tramways Board. Minimum railway or tramway time zones, 1922. URL http://handle.slv.vic.gov.au/10381/170543. 137

G. Menghini, N. Carrasco, N. Schüssler, and K. Axhausen. Route choice of cyclists in zurich. *Transportation Research Part A: Policy and Practice*, 44(9):754–765, 2010. doi:10.1016/j.tra.2010.07.008. 54

W. Meulemans. *Similarity measures and algorithms for cartographic schematization.* PhD thesis, Technische Universiteit Eindhoven, 2014. URL https://pure.tue.nl/ws/portalfiles/portal/3962300/777493.pdf. 135

D. Michail, J. Kinable, B. Naveh, and J. V. Sichi. JGraphT – a Java library for graph data structures and algorithms. *ACM Transactions on Mathematical Software*, 46(2):1–19, 2020. doi:10.1145/3381449. 71

B. Morris and M. Trivedi. Learning trajectory patterns by clustering: Experimental studies and comparative evaluation. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 312–319, 2009. doi:10.1109/CVPR.2009.5206559. 1

T. Murakami, A. Kanemura, and H. Hino. Group sparsity tensor factorization for re-identification of open mobility traces. *IEEE Transactions on Information Forensics and Security*, 12(3):689–704, 2017. doi:10.1109/TIFS.2016.2631952. 31

V. Mäkinen, A. Brauer, and J. Oksanen. Geoprivacy platform, 2023. URL https://geoprivacy.fi. 4, 30

W. Narzt, G. Pomberger, A. Ferscha, D. Kolb, R. Müller, J. Wieghardt, H. Hörtner, and C. Lindinger. A new visualization concept for navigation systems. In C. Stary and C. Stephanidis, editors, *User-Centered Interaction Paradigms for Universal Access in the Information Society*, pages 440–451. Springer, 2004. doi:10.1007/978-3-540-30111-0_38. 88

National Land Survey of Finland. Geoprivacy open data, 11 2023. National Land Survey of Finland, Finnish Geospatial Research Institute, Department of Geoinformatics and Cartography. 1, 2, 4

M. E. Nergiz, M. Atzori, and Y. Saygin. Towards trajectory anonymization: a generalization-based approach. In *Proc. SIGSPATIAL ACM GIS 2008 International Workshop on Security and Privacy in GIS and LBS (SPRINGL'08)*, pages 52–61. ACM, 2008. doi:10.1145/1503402.1503413. 31

P. Newson and J. Krumm. Hidden markov map matching through noise and sparseness. In *Proc. 17th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems (GIS'09)*, pages 336–343. ACM, 2009. doi:10.1145/1653771.1653818. 37

S. Nickel and M. Nöllenburg. Drawing k-linear metro maps. In *Proc. 2nd Schematic Mapping Workshop*, 2019. URL https://www.ac.tuwien.ac.at/files/pub/smw19-paper-6.pdf. 135

B. Niedermann, M. Nöllenburg, and I. Rutter. Radial contour labeling with straight leaders. In *Pacific Visualization Symposium (PacificVis'17)*. IEEE, 2017. doi:10.1109/PACIFICVIS.2017.8031608. 143

J. Nielsen. The 90-9-1 rule for participation inequality in social media and online communities, 2006. https://www.nngroup.com/articles/participation-inequality/, accessed on 27th Feburary 2024. 3

M. Nöllenburg. A survey on automated metro map layout methods. In *Schematic Mapping Workshop 2014*, 2014. URL https://i11www.iti.kit.edu/extra/publications/n-asamm-14.pdf. Visited on March 20, 2023. 109

M. Nöllenburg and A. Wolff. A mixed-integer program for drawing high-quality metro maps. In P. Healy and N. S. Nikolov, editors, *Graph Drawing*, Lecture Notes in Computer Science (LNTCS), pages 321–333, Berlin, Heidelberg, 2006. Springer. doi:10.1007/11618058_29. 135

M. Nöllenburg and A. Wolff. Drawing and labeling high-quality metro maps by mixed-integer programming. *IEEE Transactions on Visualization and Computer Graphics*, 17(5):626–641, 2011. doi:10.1109/TVCG.2010.81. 135

B. A. Nosek. Implicit–explicit relations. *Current Directions in Psychological Science*, 16(2):65–69, 2007. doi:10.1111/j.1467-8721.2007.00477.x. 54

J. Oehrlein, B. Niedermann, and J.-H. Haunert. Inferring the parametric weight of a bicriteria routing model from trajectories. In *Proc. 25th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems (SIGSPATIAL'17)*, pages 59:1–59:4. ACM, 2017. doi:10.1145/3139958.3140033. 4, 36, 57, 59, 66

J. Oehrlein, A. Förster, D. Schunck, Y. Dehbi, R. Roscher, and J.-H. Haunert. Inferring routing preferences of bicyclists from sparse sets of trajectories. In *Proc. 3rd International Conference on Smart Data and Smart Cities*, volume IV-4/W7 of *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, pages 107–114, 2018. doi:10.5194/isprs-annals-IV-4-W7-107-2018. 53, 59

J. Oksanen, C. Bergman, J. Sainio, and J. Westerholm. Methods for deriving and calibrating privacy-preserving heat maps from mobile sports tracking application data. *Journal of Transport Geography*, 48:135–144, 2015. doi:https://doi.org/10.1016/j.jtrangeo.2015.09.001. 4, 83

OpenStreetMap. OpenStreetMap stats, 2024. https://planet.openstreetmap.org/statistics/data_stats.html, accessed on 13th Feburary 2024. 4

T. Osogami and R. Raymond. Map matching with inverse reinforcement learning. In *Proc. 23rd International Joint Conference on Artificial Intelligence (IJCAI'13)*, pages 2547–2553. AAAI Press, 2013. URL https://www.ijcai.org/Proceedings/13/Papers/375.pdf. 37

R. Osserman. The isoperimetric inequality. *Bulletin of the American Mathematical Society*, 84(6):1182–1238, 1978. 150

D. O'Sullivan, A. Morrison, and J. Shearer. Using desktop GIS for the investigation of accessibility by public transport: an isochrone approach. *International Journal of Geographical Information Science*, 14(1):85–104, 2000. doi:10.1080/136588100240976. 108, 110

A. T. Palma, V. Bogorny, B. Kuijpers, and L. O. Alvares. A clustering-based approach for discovering interesting places in trajectories. In *Proc. 2008 ACM Symposium on Applied Computing (SAC'08)*, pages 863–868. ACM, 2008. doi:10.1145/1363686.1363886. 30

R. G. Pensa, A. Monreale, F. Pinelli, D. Pedreschi, et al. Pattern-preserving k-anonymization of sequences and its application to mobility data mining. In *CEUR Workshop Proceedings*, volume 397, pages 44–60, 2008. URL https://iris.unito.it/handle/2318/68392. 31

D. Pfoser, C. S. Jensen, and Y. Theodoridis. Novel approaches in query processing for moving object trajectories. In *Proc. 26th International Conference on Very Large Data Bases (VLDB'00)*, pages 395–406, San Francisco, CA, USA, 2000. Morgan Kaufmann. URL https://dl.acm.org/doi/10.5555/645926.672019. 37, 61

V. Primault, A. Boutet, S. B. Mokhtar, and L. Brunie. The long road to computational location privacy: A survey. *IEEE Communications Surveys & Tutorials*, 21(3):2772–2793, 2019. doi:10.1109/COMST.2018.2873950. 3, 30

H. C. Purchase. Which aesthetic has the greatest effect on human understanding? In G. DiBattista, editor, *Graph Drawing*, pages 248–261, Berlin, Heidelberg, 1997. Springer. doi:10.1007/3-540-63938-1_67. 142

H. C. Purchase. Effective information visualisation: a study of graph drawing aesthetics and algorithms. *Interacting with Computers*, 13(2):147–162, 12 2000. doi:10.1016/S0953-5438(00)00032-1. 110, 142

E. Pyrga, F. Schulz, D. Wagner, and C. Zaroliagis. Efficient models for timetable information in public transportation systems. *ACM Journal of Experimental Algorithmics*, 12, 2008. doi:10.1145/1227161.1227166. 107, 111, 113, 114, 136

J. Rauscher, R. Buchmüller, D. A. Keim, and M. Miller. SkiVis: Visual exploration and route planning in ski resorts. *IEEE Transactions on Visualization and Computer Graphics*, 30(1):869–879, 2024. doi:10.1109/TVCG.2023.3326940. 84

G. Reggiani, T. Van Oijen, H. Hamedmoghadam, W. Daamen, H. L. Vu, and S. Hoogendoorn. Understanding bikeability: a methodology to assess urban networks. *Transportation*, 49(3):897–925, 2022. doi:10.1007/s11116-021-10198-0. 53, 54

M. Ren and H. A. Karimi. Movement pattern recognition assisted map matching for pedestrian/wheelchair navigation. *The Journal of Navigation*, 65(4):617–633, 2012. doi:10.1017/S0373463312000252. 37

N. Rieser-Schüssler, M. Balmer, and K. W. Axhausen. Route choice sets for very high-resolution data. *Transportmetrica A: Transport Science*, 9(9):825–845, 2013. doi:10.1080/18128602.2012.671383. 54

M. Ringhand and M. Vollrath. Make this detour and be unselfish! Influencing urban route choice by explaining traffic management. *Transportation Research Part F: Traffic Psychology and Behaviour*, 53: 99–116, 2018. doi:10.1016/j.trf.2017.12.010. 84

A. C. Robinson. A design framework for exploratory geovisualization in epidemiology. *Information Visualization*, 6(3):197–214, 2007. doi:10.1057/palgrave.ivs.9500155. 82

T. Rossetti, C. A. Guevara, P. Galilea, and R. Hurtubia. Modeling safety as a perceptual latent variable to assess cycling infrastructure. *Transportation Research Part A: Policy and Practice*, 111:252–265, 2018. doi:10.1016/j.tra.2018.03.019. 54, 56

R. E. Roth. Interactive maps: What we know and what we need to know. *Journal of Spatial Information Science*, 6(3), June 2013. doi:10.5311/josis.2013.6.105. 81

Régie Autonome des Transports Parisiens. Metro map, 2023. https://www.ratp.fr/en/plan-metro, accessed on 13th April 2023. 152

J. Sainio, J. Westerholm, and J. Oksanen. Generating heat maps of popular routes online from massive mobile sports tracking application data in milliseconds while respecting privacy. *ISPRS International Journal of Geo-Information*, 4(4):1813–1826, 2015. doi:10.3390/ijgi4041813. 83

J. Salas, D. Megías, and V. Torra. SwapMob: Swapping trajectories for mobility anonymization. In J. Domingo-Ferrer and F. Montes, editors, *Privacy in Statistical Databases*, pages 331–346, Cham, 2018. Springer. doi:10.1007/978-3-319-99771-1_22. 31

J. F. Sallis, L. D. Frank, B. E. Saelens, and M. Kraft. Active transportation and physical activity: opportunities for collaboration on transportation and public health research. *Transportation Research Part A: Policy and Practice*, 38(4):249–268, 2004. doi:10.1016/j.tra.2003.11.003. 53

Y. Sasaki, J. Yu, and Y. Ishikawa. Road segment interpolation for incomplete road data. In *Proc. 2019 IEEE International Conference on Big Data and Smart Computing (BigComp'19)*, pages 1–8, 2019. doi:10.1109/BIGCOMP.2019.8679461. 37

R. Scheepens, N. Willems, H. van de Wetering, and J. J. van Wijk. Interactive visualization of multivariate trajectory data with density maps. In *2011 IEEE Pacific Visualization Symposium*, pages 147–154, 2011. doi:10.1109/PACIFICVIS.2011.5742384. 83

R. Scheepens, H. v. d. Wetering, and J. J. v. Wijk. Non-overlapping aggregated multivariate glyphs for moving objects. In *2014 IEEE Pacific Visualization Symposium*, pages 17–24, 2014. doi:10.1109/PacificVis.2014.13. 90

R. Scheepens, C. Hurter, H. Van De Wetering, and J. J. Van Wijk. Visualization, selection, and analysis of traffic flows. *IEEE Transactions on Visualization and Computer Graphics*, 22(1):379–388, 2016. doi:10.1109/TVCG.2015.2467112. 82

J. Schmid-Querg, A. Keler, and G. Grigoropoulos. The munich bikeability index: A practical approach for measuring urban bikeability. *Sustainability*, 13(1), 2021. doi:10.3390/su13010428. 54

N. Schüssler and K. W. Axhausen. Identifying trips and activities and their characteristics from GPS raw data without further information. *Arbeitsberichte Verkehrs- und Raumplanung*, 502, 2008. URL https://www.research-collection.ethz.ch/bitstream/handle/20.500.11850/9471/eth-30471-01.pdf. 28

D. E. Seidl, P. Jankowski, and M.-H. Tsou. Privacy and spatial pattern preservation in masked GPS trajectory data. *International Journal of Geographical Information Science*, 30(4):785–800, 2016. doi:10.1080/13658816.2015.1101767. 1, 31

H. Senaratne, A. Mobasheri, A. L. Ali, C. Capineri, and M. Haklay. A review of volunteered geographic information quality assessment methods. *International Journal of Geographical Information Science*, 31(1):139–167, 2017. doi:10.1080/13658816.2016.1189556. 3

I. N. Sener, N. Eluru, and C. R. Bhat. An analysis of bicycle route choice preferences in Texas, US. *Transportation*, 36(5):511–539, 2009. doi:10.1007/s11116-009-9201-4. 56

J. Serra. *Image analysis and mathematical morphology*. Academic Press, Inc., 1983. URL https://dl.acm.org/doi/10.5555/1098652. Visited on March 20, 2023. 120

M. Sester and C. Brenner. Continuous generalization for visualization on small mobile devices. In *Developments in Spatial Data Handling*, pages 355–368. Springer, 2005. doi:10.1007/3-540-26772-7_27. 89

D. Shin, J. Jo, B. Kim, H. Song, S.-H. Cho, and J. Seo. RCMVis: A visual analytics system for route choice modeling. *IEEE Transactions on Visualization and Computer Graphics*, 29(3):1799–1817, 2023. doi:10.1109/TVCG.2021.3131824. 84

S. H. Shin, C. G. Park, and S. Choi. New map-matching algorithm using virtual track for pedestrian dead reckoning. *ETRI Journal*, 32(6):891–900, 2010. doi:10.4218/etrij.10.0110.0037. 37

B. W. Silverman. *Density estimation for statistics and data analysis*. Routledge, 2018. doi:10.1201/9781315140919. 83

R. Song, W. Sun, B. Zheng, and Y. Zheng. PRESS: A novel framework of trajectory compression in road networks. *Proc. of the VLDB Endowment*, 7(9), 2014a. doi:10.14778/2732939.2732940. 36, 37, 61

Y. Song, D. Dahlmeier, and S. Bressan. Not so unique in the crowd: a simple and effective algorithm for anonymizing location data. In L. Si and H. Yang, editors, *Proc. 1st International Workshop on Privacy-Preserving*, volume 2014, pages 19–24, 2014b. URL https://ceur-ws.org/Vol-1225/pir2014_submission_11.pdf. 31

S. Spaccapietra, C. Parent, M. L. Damiani, J. A. de Macedo, F. Porto, and C. Vangenot. A conceptual view on trajectories. *Data & Knowledge Engineering*, 65(1):126–146, 2008. doi:10.1016/j.datak.2007.10.008. 29

I. Spassov, M. Bierlaire, and B. Merminod. Map-matching for pedestrians via bayesian inference. In *Proc. European Navigation Conference*, 2006. URL https://infoscience.epfl.ch/record/117113. Visited on March 20, 2023. 37

S. Y. Ssin, J. A. Walsh, R. T. Smith, A. Cunningham, and B. H. Thomas. GeoGate: Correlating geo-temporal datasets using an augmented reality space-time cube and tangible interactions. In *2019 IEEE Conference on Virtual Reality and 3D User Interfaces (VR)*, pages 210–219, 2019. doi:10.1109/VR.2019.8797812. 91

J. Stott, P. Rodgers, J. C. Martínez-Ovando, and S. G. Walker. Automatic metro map layout using multicriteria optimization. *IEEE Transactions on Visualization and Computer Graphics*, 17(1):101–114, 2011. doi:10.1109/TVCG.2010.24. 135

Strava Inc., 2023. https://metro.strava.com/faq, accessed on 25th January 2024. 4

Strava Inc. Year in sport – the trend report, 2024. https://press.strava.com/articles/strava-releases-year-in-sport-trend-report, accessed on 13th Feburary 2024. 4

G. Stylianou. Stay-point identification as curve extrema. *CoRR*, 2017. URL http://arxiv.org/abs/1701.06276. 29

J. Sultan, G. Ben-Haim, and J.-H. Haunert. Extracting spatial patterns in bicycle routes from crowd-sourced data. *Transactions in GIS*, 21(6):1321–1340, 2017. doi:10.1111/tgis.12280. 36, 53

V. Sumantran, C. Fine, and D. Gonsalvez. *Faster, smarter, greener: the future of the car and urban mobility.* MIT Press, 2017. 53

C. C. Sun, J. E. Hurst, and A. K. Fuller. Citizen science data collection for integrated wildlife population analyses. *Frontiers in Ecology and Evolution*, 9, 2021. doi:10.3389/fevo.2021.682124. 82

L. Sweeney. k-anonymity: A model for protecting privacy. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 10(05):557–570, 2002. doi:10.1142/S0218488502001648. 30

S. Taguchi, S. Koide, and T. Yoshimura. Online map matching with route prediction. *IEEE Transactions on Intelligent Transportation Systems*, 20(1):338–347, 2018. doi:10.1109/TITS.2018.2812147. 36

R. Tamassia. On embedding a graph in the grid with the minimum number of bends. *SIAM Journal on Computing*, 16(3):421–444, 1987. doi:10.1137/0216030. 136, 142

Y. Tao, D. Papadias, and J. Sun. The TPR*-Tree: An optimized spatio-temporal access method for predictive queries. In J.-C. Freytag, P. Lockemann, S. Abiteboul, M. Carey, P. Selinger, and A. Heuer, editors, *Proc. 2003 VLDB Conference*, pages 790–801. Morgan Kaufmann, San Francisco, CA, USA, 2003. ISBN 978-0-12-722442-8. doi:10.1016/B978-012722442-8/50075-6. 61

S. M. Tayebeh Saghapour and R. G. Thompson. Measuring cycling accessibility in metropolitan areas. *International Journal of Sustainable Transportation*, 11(5):381–394, 2017. doi:10.1080/15568318.2016.1262927. 53

M. Terrovitis, G. Poulis, N. Mamoulis, and S. Skiadopoulos. Local suppression and splitting techniques for privacy preserving publication of trajectories. *IEEE Transactions on Knowledge and Data Engineering*, 29(7):1466–1479, 2017. doi:10.1109/TKDE.2017.2675420. 31

C. Tominski, H. Schumann, G. Andrienko, and N. Andrienko. Stacking-based visualization of trajectory attribute data. *IEEE Transactions on Visualization and Computer Graphics*, 18(12):2565–2574, 2012. doi:10.1109/TVCG.2012.265. 83

L. H. Tran, Q. V. H. Nguyen, N. H. Do, and Z. Yan. Robust and hierarchical stop discovery in sparse and diverse trajectories. Technical report, EPFL, Lausanne, Switzerland, 2011. URL http://infoscience.epfl.ch/record/175473. 29, 30

Z. Tu, K. Zhao, F. Xu, Y. Li, L. Su, and D. Jin. Protecting trajectory from semantic attack considering k-anonymity, l-diversity, and t-closeness. *IEEE Transactions on Network and Service Management*, 16 (1):264–278, 2019. doi:10.1109/TNSM.2018.2877790. 31

J. W. Tukey. *Exploratory data analysis*, volume 2. Addison-Wesley, 1977. 82, 91

I. Turton. Geo tools. In G. B. Hall and M. G. Leahy, editors, *Open Source Approaches in Spatial Data Handling*, Advances in Geographic Information Science, vol 2 (AGIS), chapter 8, pages 153–169. Springer, Berlin, Heidelberg, 2008. doi:10.1007/978-3-540-74831-1_8. 71

T. C. van Dijk, A. van Goethem, J.-H. Haunert, W. Meulemans, and B. Speckmann. Map schematization with circular arcs. In M. Duckham, E. Pebesma, K. Stewart, and A. U. Frank, editors, *Geographic Information Science*, Lecture Notes in Computer Science (LNISA), pages 1–17, Cham, 2014. Springer International Publishing. doi:10.1007/978-3-319-11593-1_1. 135

A. van Goethem, W. Meulemans, B. Speckmann, and J. Wood. Exploring curved schematization of territorial outlines. *IEEE Transactions on Visualization and Computer Graphics*, 21(8):889–902, 2015. doi:10.1109/TVCG.2015.2401025. 135, 136

R. J. Van Lier. *Simplicity of visual shape: A structural information approach.* Nijmegen Instituut voor Cognitie en Informatie (NICI), 1996. 142

Verkehrbund Rhein-Sieg. Fahrplanauskunft, 2023a. https://www.vrs.de/fahren/fahrplanauskunft, accessed on 27th February 2023. 146

Verkehrbund Rhein-Sieg. Schienennetz 2023, Region Köln, 2023b. https://www.vrs.de/fileadmin/Dateien/Downloadcenter/Netzplaene/Schienennetz_RegionKoeln_2023.pdf, accessed on 27th February 2023. 146

S. Šaltenis, C. S. Jensen, S. T. Leutenegger, and M. A. Lopez. Indexing the positions of continuously moving objects. *SIGMOD Rec.*, 29(2):331–342, 2000. doi:10.1145/335191.335427. 61

H. Wang, K. Zheng, J. Xu, B. Zheng, X. Zhou, and S. Sadiq. SharkDB: An in-memory column-oriented trajectory storage. In *Proc. 23rd ACM International Conference on Conference on Information and Knowledge Management (CIKM'14)*, pages 1409–1418. ACM, 2014. doi:10.1145/2661829.2661878. 61

L. Wang, Y. Zheng, X. Xie, and W.-Y. Ma. A flexible spatio-temporal indexing scheme for large-scale GPS track retrieval. In *Proc. 9th IEEE International Conference on Mobile Data Management (MDM 2008)*, pages 1–8, 2008. doi:10.1109/MDM.2008.24. 37

C. Ware, H. C. Purchase, L. Colpoys, and M. McGill. Cognitive measurements of graph aesthetics. *Information visualization*, 1(2):103–110, 2002. doi:10.1057/palgrave.ivs.9500013. 110, 142

S. Washington, P. Congdon, M. G. Karlaftis, and F. L. Mannering. Bayesian multinomial logit: Theory and route choice example. *Transportation Research Record*, 2136(1):28–36, 2009. doi:10.3141/2136-04. 55

H. Wei, Y. Wang, G. Forman, and Y. Zhu. Map matching by Fréchet distance and global weight optimization. *Technical Paper, Departement of Computer Science and Engineering*, page 19, 2013. URL https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=4c4ea2f8d518ae953ca101a1bbffa0d10cce2a6c. 47

L. L. Y. Wei, A. A. A. Ibrahim, K. Nisar, Z. I. A. Ismail, and I. Welch. Survey on geographic visual display techniques in epidemiology: Taxonomy and characterization. *Journal of Industrial Information Integration*, 18:100139, 2020. doi:10.1016/j.jii.2020.100139. 82

L. Wenzheng, L. Junjun, and Y. Shunli. An improved Dijkstra's algorithm for shortest path planning on 2D grid maps. In *2019 IEEE 9th International Conference on Electronics Information and Emergency Communication (ICEIEC)*, pages 438–441, 2019. doi:10.1109/ICEIEC.2019.8784487. 150

C. E. White, D. Bernstein, and A. L. Kornhauser. Some map matching algorithms for personal navigation assistants. *Transportation Research Part C: Emerging Technologies*, 8(1):91–108, 2000. doi:10.1016/S0968-090X(00)00026-7. 37

P. Wilk and J. Karciarz. Optimization of map matching algorithms for indoor navigation in shopping malls. In *Proc. 2014 International Conference on Indoor Positioning and Indoor Navigation (IPIN'14)*, pages 661–669. IEEE, 2014. doi:10.1109/IPIN.2014.7275541. 37

T. D. Wilson, S. Lindsey, and T. Y. Schooler. A model of dual attitudes. *Psychological review*, 107(1): 101–126, 2000. doi:10.1037/0033-295x.107.1.101. 54

S. Winter. Modeling costs of turns in route planning. *Geoinformatica*, 6(4):345–361, dec 2002. doi:10.1023/A:1020853410145. 18, 143, 146

H.-Y. Wu, B. Niedermann, S. Takahashi, and M. Nöllenburg. A survey on computing schematic network maps: The challenge to interactivity. In *The 2nd Schematic Mapping Workshop 2019*, 2019. URL https://www.ac.tuwien.ac.at/files/pub/smw19-position-5.pdf. Visited on March 20, 2023. 109

H.-Y. Wu, B. Niedermann, S. Takahashi, M. J. Roberts, and M. Nöllenburg. A survey on transit map layout – from design, machine, and human perspectives. *Computer Graphics Forum*, 39(3):619–646, 2020. doi:10.1111/cgf.14030. 109, 135

K. Xie, K. Deng, and X. Zhou. From trajectories to activities: a spatio-temporal join approach. In *Proc. 2009 International Workshop on Location Based Social Networks (LBSN'09)*, pages 25–2. ACM, 2009. doi:10.1145/1629890.1629897. 29

Z. Yan, C. Parent, S. Spaccapietra, and D. Chakraborty. A hybrid model and computing platform for spatio-semantic trajectories. In L. Aroyo, G. Antoniou, E. Hyvönen, A. ten Teije, H. Stuckenschmidt, L. Cabral, and T. Tudorache, editors, *The Semantic Web: Research and Applications*, pages 60–75, Berlin, Heidelberg, 2010. Springer. doi:10.1007/978-3-642-13486-9_5. 29

R. Yarovoy, F. Bonchi, L. V. S. Lakshmanan, and W. H. Wang. Anonymizing moving objects: how to hide a mob in a crowd? In *Proc. 12th International Conference on Extending Database Technology: Advances in Database Technology (EDBT'09)*, pages 72–83. ACM, 2009. doi:10.1145/1516360.1516370. 31

Y. Ye, Y. Zheng, Y. Chen, J. Feng, and X. Xie. Mining individual life pattern based on location history. In *2009 Tenth International Conference on Mobile Data Management: Systems, Services and Middleware*, pages 1–10, 2009. doi:10.1109/MDM.2009.11. 21

H. Y. Yoo and S. H. Cheon. Visualization by information type on mobile device. In *Proc. 2006 Asia-Pacific Symposium on Information Visualisation (APVis'06)*, pages 143–146. Australian Computer Society, 2006. URL https://dl.acm.org/doi/10.5555/1151903.1151925. Visited on March 20, 2023. 90

Z. Yu, Y. Guo, and Y. Chen. Learning trajectory routing with graph neural networks. In *Proc. 5th International Conference on Big Data and Computing (ICBDC'20)*, pages 121–126. ACM, 2020. doi:10.1145/3404687.3404701. 56

G. Yuan, P. Sun, J. Zhao, D. Li, and C. Wang. A review of moving object trajectory clustering algorithms. *Artificial Intelligence Review*, 47(1):123–144, Mar. 2016. doi:10.1007/s10462-016-9477-7. 1

P. T. Zellweger, J. D. Mackinlay, L. Good, M. Stefik, and P. Baudisch. City lights: Contextual views in minimal space. In *CHI'03 Extended Abstracts on Human Factors in Computing Systems (CHI EA'03)*, pages 838–839. ACM, 2003. doi:10.1145/765891.766022. 91

E. Zhang and N. Masoud. Increasing GPS localization accuracy with reinforcement learning. *IEEE Transactions on Intelligent Transportation Systems*, 2020. doi:10.1109/TITS.2020.2972409. 37

G. Zhang. Integrating citizen science and GIS for wildlife habitat assessment. In M. Ferretti, editor, *Wildlife Population Monitoring*, chapter 2. IntechOpen, Rijeka, 2019. doi:10.5772/intechopen.83681. 82

G. Zhang and A.-X. Zhu. The representativeness and spatial bias of volunteered geographic information: a review. *Annals of GIS*, 24(3):151–162, 2018. doi:10.1080/19475683.2018.1501607. 3

L. Zhang, S. Dalyot, D. Eggert, and M. Sester. Multi-stage approach to travel-mode segmentation and classification of GPS traces. *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, XXXVIII-4/W25:87–93, 2011. doi:10.5194/isprsarchives-XXXVIII-4-W25-87-2011. 60

L. Zhang, S. Dalyot, and M. Sester. Travel-mode classification for optimizing vehicular travel route planning. In *Progress in Location-Based Services*, pages 277–295. Springer, Berlin, Heidelberg, 2013. doi:10.1007/978-3-642-34203-5_16. 1

L. Zhang, M. Cheng, Z. Xiao, L. Zhou, and J. Zhou. Adaptable map matching using PF-net for pedestrian indoor localization. *IEEE Communications Letters*, 24(7):1437–1440, 2020. doi:10.1109/LCOMM.2020.2984036. 37

Y. Zheng. T-drive trajectory data sample, August 2011. URL https://www.microsoft.com/en-us/research/publication/t-drive-trajectory-data-sample/. 4

Y. Zheng. Trajectory data mining: An overview. *ACM Transactions on Intelligent Systems and Technology*, 6(3):1–41, 2015. doi:10.1145/2743025. 60

Y. Zheng, H. Fu, X. Xie, W.-Y. Ma, and Q. Li. *Geolife GPS trajectory dataset - User Guide*, July 2011. URL https://www.microsoft.com/en-us/research/publication/geolife-gps-trajectory-dataset-user-guide/. 4

Q. Zhou and Z. Li. Empirical determination of geometric parameters for selective omission in a road network. *International Journal of Geographical Information Science*, 30(2):263–299, 2016. doi:10.1080/13658816.2015.1085538. 110