Algorithms for Context-Aware Trajectory Analysis

Citation for published version (APA):
Custers, B. A. (2022). *Algorithms for Context-Aware Trajectory Analysis*. [Phd Thesis 1 (Research TU/e / Graduation TU/e), Mathematics and Computer Science]. Eindhoven University of Technology.

Document status and date:
Published: 16/12/2022

Document Version:
Publisher’s PDF, also known as Version of Record (includes final page, issue and volume numbers)

Please check the document version of this publication:

• A submitted manuscript is the version of the article upon submission and before peer-review. There can be important differences between the submitted version and the official published version of record. People interested in the research are advised to contact the author for the final version of the publication, or visit the DOI to the publisher's website.
• The final author version and the galley proof are versions of the publication after peer review.
• The final published version features the final layout of the paper including the volume, issue and page numbers.

Link to publication

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

- Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
- You may not further distribute the material or use it for any profit-making activity or commercial gain
- You may freely distribute the URL identifying the publication in the public portal.

If the publication is distributed under the terms of Article 25fa of the Dutch Copyright Act, indicated by the “Taverne” license above, please follow below link for the End User Agreement:

www.tue.nl/taverne

Take down policy
If you believe that this document breaches copyright please contact us at:

openaccess@tue.nl

providing details and we will investigate your claim.

Download date: 04. Oct. 2023

Algorithms for Context-Aware Trajectory Analysis

PROEFSCHRIFT

ter verkrijging van de graad van doctor aan de
Technische Universiteit Eindhoven, op gezag van de
rector magnificus prof.dr.ir. F.P.T. Baaijens, voor een
commissie aangewezen door het College voor
Promoties, in het openbaar te verdedigen
op vrijdag 16 december 2022 om 16:00 uur

door

Bram Alwin Custers

geboren te Utrecht

Dit proefschrift is goedgekeurd door de promotoren en de samenstelling van de promotiecommissie is als volgt:

Voorzitter:	prof.dr. M. de Berg
Promotoren:	prof.dr. B. Speckmann
	dr. K.A.B. Verbeek
Copromotor:	dr. W. Meulemans
Promotiecommissieleden:	prof.dr. N. Andrienko (City University London) prof.dr. M.E. Buchin (Ruhr-Universität Bochum) prof.dr. A. Vilanova

Het onderzoek of ontwerp dat in dit proefschrift wordt beschreven is uitgevoerd in overeenstemming met de TU/e Gedragscode Wetenschapsbeoefening.

The work in this thesis is supported by the Netherlands' Organization for Scientific Research (NWO) under project no. 628.011.005.

A catalogue record is available from the Eindhoven University of Technology Library.
ISBN: 978-90-386-5604-5.

Printed by Ipskamp Printing, Enschede.

This thesis has been typeset using PDFLaTeX with the fonts Linux Biolinum and Linux Libertine.

© 2022 by Bram Alwin Custers. All rights reserved. Reproduction in whole or in part is prohibited without the written consent of the copyright owner.

Acknowledgements

Looking back at the last four and a bit years, I have learned and experienced a lot during my PhD. It was hard work for me to get to this point, but of course I could not have done this all by myself. I have a lot of people to thank for that. I'll try to thank as many of you as I can, though it is by no means going to be an exhaustive.

First of all, I want to thank my supervisors, Bettina, Kevin, and Wouter. Without you, I would not have made it to the end. Bettina is such a kind and knowledgeable person. It always amazes me how easily she captures intuitions behind the problems we set out to solve. Kevin always has these amazing out-of-the-box ideas that turn out to work, though it always took me quite some time to understand why. Wouter is a really caring person. He was there keeping up morale close to paper deadlines, and I really enjoyed our pair programming/debugging sessions. Thank you all for your support throughout my PhD, teaching me valuable lessons and providing insights in the problems we set out to solve. And also thank you for your patience and support when things did not go so smooth.

Next, a big thanks to the committee, Natalya Andrienko, Maike Buchin, and Anna Vilanova, for your time and effort to read my thesis and provide feedback.

During my PhD, I had the pleasure to be part of three office configurations. First, I got to share the corner office with Jules and Willem, who taught me a lot about how to PhD. Next, I shared an office with Aleks, Daniel, and Huib, a hard working and deeply concentrated office. Finally, I shared an office with Aleks, Max, and Pantea. Here, we mixed hard work with enjoyable political or philosophical discussions. Thanks to all of you for the fun and inspiring times!

I would also like to thank all people of the ALGA cluster for their company and the social events, Bettina, Kevin Verbeek, Wouter, Mark, Antske, Bart, Kevin Buchin, Irina, Morteza, Tim, Marcel, Leonie, Willem, Arthur, Henk, Nathan, Leyla, Huib, Pantea, Thijs, Jari, Max Sondag, Max van Mulken, Dániel, Jules, Astrid, Tom, Alesksandr, Shivesh, Arpan, Martijn, Leonidas, Ruben, and Agnes. Special thanks to Arthur for the very insightful conversations we had. Additionally, I would like to thank all smart and kind people that I had the pleasure of doing research with or otherwise collaborated with, Jeff Ericson, Ivor, Mees, Frank, Maarten, Kevin Buchin, Marcel, Aleks, Martijn, Irina, Craig, Matei, Onur and Aniket. Finally, also a thank you to people I met at conferences, making them great experiences, Thomas, Stefan, Jana.

Outside of academia, I also fortunately had activities to get my mind off of research every so often. Quirijn and Willem, thank you for the many fun game jams and game programming days and nights. Also, a lot of thanks to my bands. In particular, thank you Ben, Loek, Mark, and Simon for your musical enthusiasm, the fun conversations over drinks during rehearsal nights and the nice gigs we got to (eventually) do. And of course a lot of thanks to all my friends for putting up with my complaining. In particular, Thijs, Matthijs, Jan, Laura, Jeroen, thanks a lot to all of you for our conversations regarding life and music.

Last but not least, I would like to thank my family. Where would I be without you all. Mom, thank you for all your love and help. Dad, Greetje, thank you as well for lending me an ear when I needed it. My bros and SOs, Boy & Zoe, Wessel and Rinze & Minke, thank you for the fun times. And thank you, Maddy, for bringing a new sparkle of joy in the family.

Eindhoven, November 2022

Contents

1 Introduction **1**

1.1 On the nature of trajectory data 2

1.2 Analyzing trajectories 5

1.3 Context-aware trajectory analysis 6

1.3.1 Defining context 6

1.3.2 Road network 8

1.3.3 Local traffic situation 10

1.3.4 Physical properties 11

1.4 Contributions 12

1.5 Other results 17

2 Maximum Physically Consistent Trajectories **19**

2.1 Concatenable consistency model 23

2.2 The speed-bounded model in 2D 24

2.2.1 A consistency data structure 24

2.2.2 Supporting insertions 26

2.2.3 Maximum subsequence queries 27

2.2.4 Maximum consistent subtrajectories 29

2.3 The acceleration-bounded model 30

2.3.1 Computing the maximum length of a physically consistent subtrajectory 31

2.3.2 Retrieving the physically consistent subtrajectory 34

2.3.3 Bounding the maximum fragmentation 34

2.3.4 Extending to higher dimensions 37
2.4 Experiments 38
2.4.1 Algorithms 38
2.4.2 Datasets 39
2.4.3 Comparing algorithms and models 40
2.4.4 Sensitivity of model parameters 46
2.5 Discussion 55
2.6 Derivation of the propagation function 57
2.6.1 Proof: achieving minimum velocity without attaining velocity bounds 62
2.6.2 Proof: achieving minimum velocity when attaining velocity bounds 63

3 Physically Consistent Map-Matching **67**
3.1 Consistent map matching 70
3.1.1 Computing a consistent route 70
3.2 Intermezzo: computing $v_{\uparrow,f}$ and $v_{\downarrow,f}$ 79
3.2.1 Other aspects 80
3.3 Experiments 81
3.3.1 Comparison with baseline map matchers 83
3.3.2 Effect of speed slack 86
3.4 Conclusion and discussion 87

4 Route Reconstruction from Traffic Flow via Representative Trajectories **91**
4.1 Modeling 93
4.1.1 Modeling loop-detector data as time-independent complete flows 94
4.1.2 Reconstructed routes 95
4.1.3 Realistic routes 96
4.1.4 Formal problem statement 97
4.2 Computational complexity 97
4.3 Route reconstruction algorithms 101
4.3.1 Fréchet Routes 101
4.3.2 Min-cost flow 105
4.4 Experimental evaluation 106

4.4.1 Fréchet Routes 107
4.4.2 Comparing different methods 109
4.5 Case study 113
4.6 Discussion 115

5 Coordinated Schematization for Visualizing Mobility Patterns on Networks 119
5.1 The pipeline 121
5.2 Step 1: Map-match trajectories to the network 123
5.3 Step 2: Select roads 124
5.4 Step 3: Schematize the network 125
5.5 Step 4: Detect bundles 129
5.6 Step 5: Render schematic network with bundles 133
5.7 Results 134
5.8 Discussion and future work 135

6 Schematizing Orthogonal Polygons 139
6.1 Minimum homotopy area for schematization 145
6.1.1 Computing the minimum homotopy area 145
6.1.2 Simplicity of a schematization under minimum homotopy area 147
6.2 Computing minimum homotopy area schematizations 149
6.2.1 Canonical form 149
6.2.2 Computing the optimal schematization 151
6.3 Greedy approximation algorithm for staircases 156
6.3.1 Properties staircase schematizations under the greedy and optimal algorithm 158
6.3.2 Minimum greedy move size 160
6.3.3 Approximation factor for uniform staircase 166
6.4 Conclusion 170

7 Conclusion and Future Work 173
7.1 Main results 173
7.2 Future work 175
7.2.1 Using the temporal component of trajectories and context . 175
7.2.2 Anonymization, privacy and contextual data 175

7.2.3 Realistic input models 176

Bibliography **177**

8 Summary **193**

Curriculum vitae **197**

Chapter 1

Introduction

Movement is an omnipresent phenomenon in our lives. Arguably, movement is what defines us as human beings and allowed us to connect cultures, trade across the globe, and go even beyond our own planet. How we move and where we move to tells a lot about who we are, whether we travel by car, plane or public transport, what points of interest we visit, such as supermarkets or hospitals, and who we meet and connect to. Similarly, movement patterns tell us a lot about other animals that we share this earth with. Our movement has an effect on our surroundings, we place infrastructure to make movement easier, but also leave pollution depending on the mode of transport that we choose, and we get stuck in traffic jams when we all decide to use the same road. To know about and act upon these effects of our movement, or to better understand the movement of others, it is fundamental to understand how movement works and what drives movement. For this, we need to be able to measure movement.

We can define movement as the mapping of time to a location for a specific entity for some time interval. Since we cannot directly work with continuous data, a convenient and well-established way of representing movement, and thus spatio-temporal data, is the concept of a *trajectory*. A trajectory consists of a sequence of measurements with increasing timestamps that capture the location of an object at the moment of measuring. The measurements of the trajectory may further be enriched with auxiliary data that captures the local context of the measurement, such as weather conditions, acceleration or velocity of the object at that time, or the current mode of transport (airplane, bus, car).

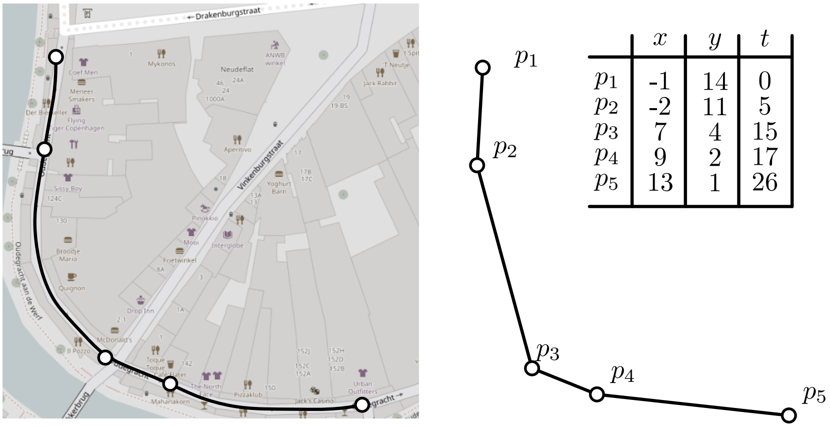
There are myriad ways to acquire trajectories. One common way is to employ the Global Positioning System (GPS) via dedicated hardware in vehicles, or nowadays via smartphones. GPS data are particularly convenient since they allow us to track movement around the globe more easily [102], which makes it useful for human mobility analysis but also for analysis of animal movement patterns. If we want to understand movement, we want to find patterns in a large set of collected trajectories or patterns in trajectories with a large number of measurements. Since these analyses are generally not feasible to do by hand, we need ways to automate our analyses: we need to develop algorithms to analyze the trajectories. These analyses may then directly uncover patterns in the trajectory data set, or they may result in visualizations that make it easier for experts to analyze the trajectory data set.

Using algorithmic trajectory analysis, we may seek to increase our knowledge about movement of individuals or groups for which we have captured the trajectories. Though we can analyze the trajectories by purely looking at their spatial and temporal component, this allows for only a limited extraction of meaningful knowledge from the trajectories. However, movement happens in a *context*: there are internal and external factors for the object that influence how it moves, why it moves, where it moves to. Using some knowledge of the context in which we recorded the trajectories, we can achieve deeper and perhaps more meaningful understanding of the movement represented by the trajectories.

In this thesis, we seek to develop algorithms for analyzing movement that are *context-aware*: we make use of the known contextual data, and combine this in our algorithms with the trajectory data which makes it possible to derive new information or produce visualizations. Since our trajectory data are real-world data, there are challenges to overcome when working with the data. We discuss these challenges in more detail in Section 1.1. Then, we describe some important techniques applicable to trajectories in Section 1.2. We more formally define what we mean by context in Section 1.3. We also look at what are important types of contextual data and what we can achieve when adding context to trajectory analysis algorithms in this section.

▸ 1.1 On the nature of trajectory data

Trajectories are a very convenient and helpful way for analyzing movement and mobility. Formally, we define a trajectory T as a sequence of measurements $T = \langle p_1, \ldots, p_n \rangle$, where each measurement is a tuple $p_i = (x_i, y_i, t_i, \ldots)$ of at least position (x_i, y_i) and timestamp t_i, and potentially additional measured quantities such as velocity, heading or acceleration at the time of measuring (see Fig. 1.1 for an exam-



	x	y	t
p_1	-1	14	0
p_2	-2	11	5
p_3	7	4	15
p_4	9	2	17
p_5	13	1	26

Figure 1.1 (left) Sampling movement of an entity results in a trajectory. (right) A trajetory is then a sequence of tuples with a timestamp and location.

ple). The measurements are ordered in time, that is, for consecutive measurement m_i, m_{i+1}, it holds that $t_i < t_{i+1}$. For convenience, we assume that trajectories are embedded in Euclidean space. Whenever we acquire trajectories that use a non-Euclidean space, such as trajectory data collected via GPS, we can project these to a Euclidean space, introducing a slight error relative to the true measurement locations. When visualizing trajectories, we often draw them as polylines, as shown in Fig. 1.1 on the right side. Note that the actual movement between measurements may have been very different, but this cannot be inferred from the measurements alone.

As with all real world data, there are some challenges that we need to address when working with trajectories. Here, we discuss some major challenges that we seek to overcome in the work presented in this thesis.

Data challenges We discern three types of data challenges when using trajectory data: noise, privacy and completeness (see Fig. 1.2 for examples). Regarding the first challenge, all measurements using physical sensors are subject to noise, or may not be precise in their representation of the location of the object. For GPS data in urban environments, a well-known source of noise is urban canyoning [16], where reflection of the signal on buildings makes the measurements more imprecise. In addition, bad reception also plays a role, in particular when moving through tunnels that may partially or completely block signals of the GPS receiver.

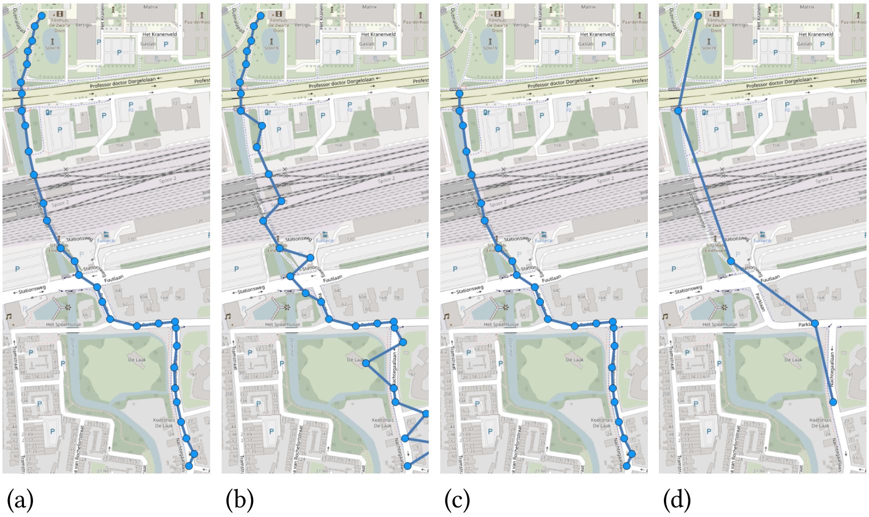

(a) (b) (c) (d)

Figure 1.2 (a) “Ideal” trajectory in an urban environment. (b) Trajectory, subject to noise (c) Incomplete trajectory due to privacy. (d) Sparse, incomplete trajectory (basemaps by OpenStreetMaps)

Secondly, for human mobility, privacy concerns form an important aspect for the data. Knowing the exact location of a person for a long period of time reveals private information of that person, which is deemed unethical or is illegal to reveal in some regions of the world [69]. To counteract these concerns, trajectory data are often preprocessed to remove sensitive information such as home location, warehouse location, etc. This in effect then degrades the quality of the individual trajectories for analysis. Thus, a vast body of research is aimed at developing algorithms to enforce privacy while finding the appropriate trade-off between privacy and usability of the data (see [51] for a survey).

Finally, the collected trajectory data is by definition a subsample of the actual, continuous trajectory. Due to power usage of GPS receiver hardware or privacy concerns, one may consider sampling the movement with a low temporal sampling frequency, in effect making it more uncertain where the object was between measurements. In addition, it can occur that entire sections in a trajectory are missing due to hardware failure or a privacy processing algorithm.

Data set volume Depending on the particular application of interest, the collected trajectory data may be vastly different: one may be interested in the fine-grained movement of a small group of birds, or perhaps the large-scale movement patterns of as many vehicles in a city as possible. The shapes of these data sets give additional challenges, simply due to the sheer size of the number of trajectories, of their individual measurement counts, or both.

Data set completeness Even though the volume of trajectory data sets may be large, it will rarely be the case that we have a complete picture of the phenomenon we are studying. Particularly for human mobility, capturing all traffic at a fine-grained resolution would be intractable from a logistical and technical perspective, and undesirable from a privacy perspective. Hence we can always assume that any trajectory data set relating to human mobility is a sample of the underlying traffic, that is, it is inherently incomplete.

▸ 1.2 Analyzing trajectories

Trajectories themselves can already provide insight into the underlying entities by simply showing their geometry on maps for experts to interpret. However, using algorithms to process the raw data, we are able to extract patterns and improve the visualizations, making interpretation easier for experts. Hence, many processing and analysis techniques exist for handling trajectory data. Below, we discuss some important processing and analyzing techniques.

Similarity measures A fundamental aspect to most trajectory analysis algorithms are similarity measures: measures between trajectories that convey how (dis)similar two trajectories are. In the computational geometry community, well-known and much used measures for trajectories are amongst others the Fréchet distance [4], the Hausdorff distance [60] and dynamic time warping [134]. These measures in general consider a trajectory to be a directed polyline, thus only taking into account the spatial aspect of trajectories. However, actually incorporating the temporal component of trajectories in a meaningful way often depends strongly on the application. The temporal component may just be considered another dimension, in which case the polyline distances directly apply. Alternatively, it can be considered a separate special dimension, for instance in the Skorokhod distance [85], which is very similar to the Fréchet distance, but treats the time dimension differently. For other applications, trajectories may periodically exhibit similar behavior, for instance slow driving during rush hour on week days, thus similarity measures should adapt accordingly. In that respect, trajectories can also be considered to be time series with a spatial component, thus making time series similarity measures potentially applicable. In particular, edit distances are commonly used, such as the longest common subsequence (LCSS) and Edit Distance on Real Sequences [107]. These then explicitly take into account the temporal component, often by applying a time window within which measurements of the compared trajectories are allowed to match.

Simplifying trajectories Depending on the application and the trajectory data set, the amount of data in individual trajectories may be too fine-grained, which causes processing the trajectories to be slow or visualization of the trajectories difficult. In these cases, it may be beneficial to simplify trajectories beforehand, that is, reduce the complexity of the trajectory while maintaining the most important parts for the application. The idea is in general to keep the trajectory similar enough to the original to be useful, commonly measured using similarity measures as described previously. Ideally we reduce the complexity as much as possible. In computational geometry, this is a well-studied topic, especially with regard to the Fréchet, Hausdorff and Dynamic Time Warping distances [111].

▸ 1.3 Context-aware trajectory analysis

Many analysis and processing techniques on trajectories are readily available if we only have spatio-temporal information of the trajectories. However, movement rarely happens in the perfectly flat and empty space: we may move in a road-network, there may be height difference in our space, there may be obstacles on the way, other entities or we are influenced by external factors when choosing how to move, such as traffic jams or bad weather. That is, movement happens in a certain *context*. This context, then, provides more information on the movement of the entity, thus allowing us to improve the analysis and processing of trajectory or trajectories, or deriving new information from the trajectory.

▸ 1.3.1 Defining context

Over the past decades, the GIS community has developed a multitude of definitions for what “context” means when it comes to movement. These definitions capture different aspects of the types of context that movement happens in. Buchin, Dodge and Speckmann define the *geographic context* as “the locational circumstances of a moving agent” [23]. In the context of their work, this is often knowledge about stationary information such as obstacles. Though the context may be localized, its effect does not have to be necessarily, since knowledge of the context by the entities internally may change the global movement behavior, i.e. anticipating that there is an obstacle in the future.

Andrienko, Andrienko and Heurich [6] provide a more general, formal description of context in movement analysis. They define two archetypical elements, *spatial objects* and *events*, from which the major contextual data types derive. Entities (*movers* in their terminology) produce trajectories, which are spatial events, mapping specific times to spatial locations. Contextual data is then either a spatial object (fixed location, does not change in time), an event (present for fixed time interval, no location) or a combination of the two.

Sharif and Alesheikh [105] give a definition of context for movement data by characterizing the context based on whether it is considered internal or external to the entity. Context is internal when the contextual data comes from knowledge of the entity or internal workings of it, while context is external when it describes the situation that the entity is in. They then further subdivide the internal context into motivation, movement and modality context. Motivation captures the intrinsic reasons why an entity moves, while modality captures how the entity moves, e.g. what its capabilities are physically, what its shape is. The movement context is defined by Dodge, Weibel and Lautenschütz [44] as the movement parameters of the entity and captures the state of the entity, such as its velocity and acceleration. Finally, there are relations amongst the context types, as well as interactions with the context, for instance when the entity interacts with other entities.

Broadly speaking then, context is all the circumstances that in any way alter the movement of an entity or group of entities. Context may then be used as extra knowledge regarding the movement, thus increasing the amount of information we can infer from the raw trajectory data and the context. Reversing this idea, we may also try to infer contextual information from raw trajectory data, commonly via segmentation or classification, to determine what type of behavior the entity exhibits for that particular (sub)trajectory.

In the next sections, we introduce several contextual data types that we use in this thesis to develop algorithms. We then provide descriptions of specific analysis and processing techniques for trajectories that benefit from this particular type of contextual data.

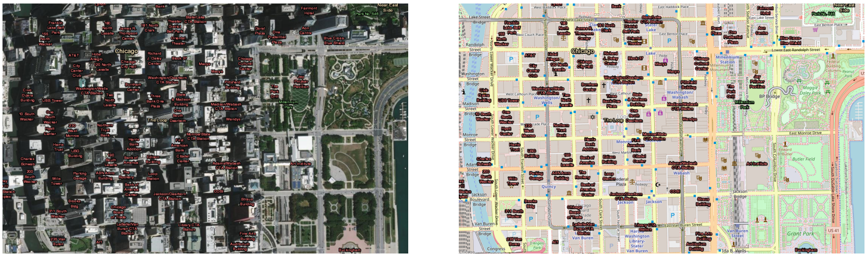


Figure 1.3 (left) Urban environment that may be context of movement (downtown Chicago, satellite image courtesy of the U.S. Geological Survey). (right) Road network representation of the urban environment (basemap by OpenStreetMaps)

1.3.2 Road network

One of the key constraining contexts for human mobility is the *road network*: movement is constrained to a specific subspace due to traffic laws, but also due to the geometry of surrounding obstacles such as buildings and impassable terrain (mountains, rivers, lakes). This acts as geographic context data. In principle, we can view a road network as a non-simple polygon with annotated data such as speed limits, driving directions, lane designations, etc (see Fig. 1.3). In practice, we model a road network as a graph, embedded in $\mathbb{R}^2$ (or $\mathbb{R}^3$). More formally, we define a road network $\mathcal{G} = (V, E)$ to be a simple, directed graph with vertices V and edges E. In addition, we assume all edges in E to be straight-line embedded in $\mathbb{R}^2$, that is, each edge is represented by a straight line, where its start and end vertices have a fixed location in $\mathbb{R}^2$.

Note that graph $\mathcal{G}$ does not need to be planar, since fly-overs may result in crossing edges. We can planarize this graph by inserting vertices at the intersections, while remembering that these vertices are virtual. In this case, we speak of a *planarized road network*. When needed, we can associate to the edges or vertices contextual data such as speed limits, road types (highway, residential road) and other contextual data. Also note that using an embedded graph for constraining the motion of objects need not be restricted to a road network: sea lanes and airways can also be modeled using embedded graphs.

Detecting patterns in sets of trajectories When given a set of trajectories, one can go even further and look for particular behavior of subgroups of entities. Typical approaches here are to find clusters of trajectories that exhibit similar behaviour [136], finding “constructed” trajectories that represent the approximate movement of a group of trajectories [20] and tracking the formation and dissolution of groups in space and time [18]. These approaches result in the discovery of typical behavior in the trajectory data set, which can be used to further analyze the movement patterns of the entities. When we know that the trajectories are actually constrained to a road network, we can use the representation of trajectories on the network: we can describe the measurements of the trajectory by their location on the graph instead of in $\mathbb{R}^2$. This may greatly simplify clustering or make it more efficient and meaningful [59, 125]. In Chapter 5, we use this compact representation of trajectories on road networks to detect major routes in a large trajectory set.

Simplifying trajectories When we know the space that a trajectory moves in, we can alter our simplification algorithms accordingly. In particular, we want to have simplifications that are reasonable in the new space. When we know that there are obstacles in our two dimensional space, it makes sense to keep the topology of the trajectory the same, only allowing trajectories that are homotopic to our input trajectory [1, 28]. When we know our entities move in a road network however, homotopic simplifications are less meaningful or rather trivial. Instead, one can consider simplifying the space, that is, the road network itself, and simplifying the trajectories along with it. We leverage this idea in Chapter 5, where we want to simplify both the network and the trajectories for analysis and visualization. We apply local simplification operators to the network, while also simplifying the matched trajectories on the network.

Visualizing trajectory data sets on road networks When visualizing a trajectory data set, one can simply show all trajectories spatially on a plane or map. However, only for small enough or extremely sparse data sets will this result in a visualization that is easily interpreted. To be able to visualize meaningful data, we need a way to aggregate the trajectory data, resulting in less clutter and summarizing the data sets. This may be done by for instance determining the density of the trajectories [123]. But knowledge of the underlying space that the entities moved in can also make visualization easier. In particular, if we know that the entities adhere to an embedded network, such as a road network or sea lanes, this opens up possibilities to leverage the network for visualization of the trajectories, potentially leading to more concise and uncluttered visualizations [5]. In Chapter 5, we use the knowledge that our traffic moves in a road network to develop a visualization to show major mobility patterns in a trajectory data set.

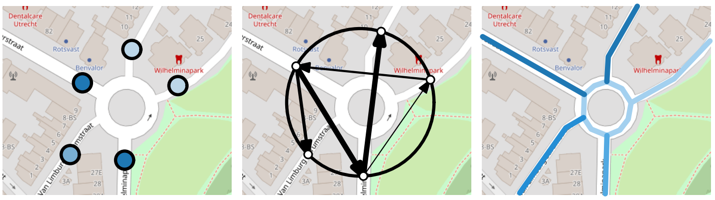
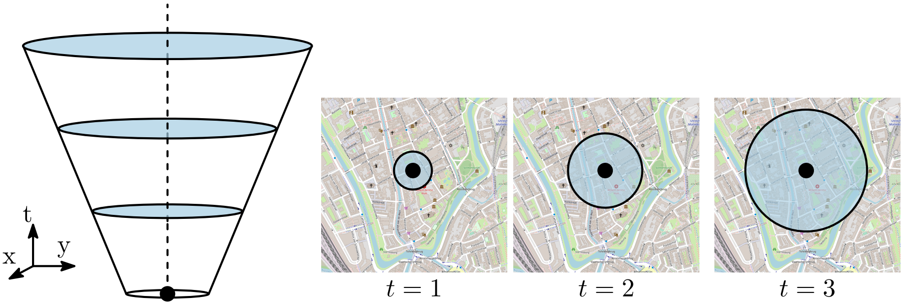


Figure 1.4 We may know the local traffic situation such as number of vehicles passing a point (left), turning ratios (middle) or average speed over parts of the road network (right).

1.3.3 Local traffic situation

In addition to knowing that our trajectories occur in the context of a road network, we may have information about the local traffic situation. This may commonly be represented as functions over the graph edges and vertices, describing the state of local traffic (see Fig. 1.4). Examples of this would be knowledge of the average velocity of vehicles passing fixed points in the road-network, knowledge of the amount of vehicles passing fixed points in the road-network for a specific time period or knowledge of the turning ratio of flows at intersections (the relative amount of traffic exiting via a specific edge of the intersection). This data can be gathered by installing sensors at fixed locations in the road network. Inductive loop detectors and cameras are the most common detector types. The inductive loops can be utilized to count the number of vehicles passing a certain location, their approximate velocity while passing the detector and their approximate length. Using computer vision techniques, the video feeds of the cameras can be used to recognize number plates, which allow for more precise tracking of individuals, but also recording turning ratios of vehicles at intersections. This contextual data can be considered a combination of the spatial objects and events as defined by Andrienko, Andrienko and Heurich [6].

Reconstructing routes from local traffic volume When we have knowledge of the local traffic situation, in particular the number of vehicles that passed a certain location in the network, we can ask the question: is there a reasonable way to infer the routes of the entities that caused these counts? Essentially, we are reconstructing the original route set, of which the counts are a local aggregation. We may then leverage the fact that a set of trajectories and these local counts are complementary data: while the trajectories are a subset of all routes of traffic in the network, the



$t = 1$ $t = 2$ $t = 3$

Figure 1.5 Knowing the maximum speed limits where an entity can be to a disk, given some initial position.

local counts fully count the traffic, only does not capture routes. We use this notion in Chapter 4 to reconstruct realistic routes: routes that remain similar enough to the input trajectory data, that we deem representative. The idea is then to match as many routes with the counts as possible, thus reconstructing the routes.

1.3.4 Physical properties

Another key piece of contextual data is knowing what kind of object or entity was measured with a trajectory. If we know this, we can derive certain physical properties that we can use in processing and analyzing the trajectory. For example, a car in a road network will not travel with the speed of sound, but will reach perhaps 200 to 300 km/h on the German highway at its highest. Similarly, accelerating too fast is in general not a very comfortable or safe experience for living beings, thus acceleration of entities is in principal bounded. Effectively, this information constrains the movement characteristics that an entity can have at any given moment in time, but in addition also limits where an entity can be, given a starting position and some elapsed time (see Fig. 1.5). We can characterize this contextual data as modality context according to the definition by Sharif and Alesheikh [105].

Finding outliers As discussed previously, collected trajectory data may be subject to noise. Hence, it is important to assess and if possible improve the trajectory data before using the data in further analysis. To this end, one uses outlier detection algorithms, sometimes also referred to as "anomaly" detection algorithms [139]. Using these algorithms, we can detect which particular measurements in the trajectory data are not coherent with the other measurements of the trajectory. How to determine coherency is then specified by the particular contextual model that one uses for detecting the outliers, which amongst others can take the physical properties of the entity into account. In Chapter 2, we apply this idea, finding outliers by determining the maximum set of measurements of a trajectory that form a trajectory the entity could reasonably have taken, given physical constraints on its motion.

Map-matching We can consider the road network as context for trajectories in a given data set. In that case, we assume that all entities were driving on this road-network when the measurements were taken. To be able to reason about the trajectories on the road network, while also removing any potential noise in the trajectories, we can *map-match* the trajectories to the road network. When we map-match a trajectory to the network, we want to find the (spatio-temporal) representation of the trajectory on the road network that is as faithful as possible to the original trajectory. This problem has long been studied in the GIS and computational geometry communities, giving rise to for example map matchers minimizing several variants of the Fréchet distance [3] and map matchers employing Hidden Markov Models to find the route maximizing the probability that the input trajectory and route concur [74].

While the road-network as context already provides a lot of information regarding where a potential similar trajectory could or should be, it may not uniquely define the desired route in the network. Hence, we can infuse even more contextual data into the map matchers to account for the behavioral model for the entity that generates the trajectory. Particularly, the local traffic laws may be incorporated, disallowing counterflow driving and requiring the entity to (approximately) obey the local speed limits. Additionally, one may use the physical constraints of the entity to discard routes that requiring unrealistically high velocities or extreme turns given the velocity. We use these ideas to develop a map-matching algorithm in Chapter 3 that tries to find routes in the road network that are similar to the provided trajectory and makes sense, given the physical constraints for the entity of the trajectory.

▸ 1.4 Contributions

We next describe the contributions of this thesis, seeking to effectively employ contextual data to improve and enrich analysis and visualization techniques for trajectories.

Maximum Physically Consistent Trajectories

In Chapter 2, we propose algorithms for detecting outliers in trajectories by employing *physical consistency* (see Fig. 1.6). This physical consistency models the capabilities that an entity has when it comes to moving around in space: how fast can the entity accelerate, how fast can it decelerate, what is its maximum speed? Using this contextual information for the entities, we propose outlier detection algorithms that find the longest subsequence of the trajectory that is consistent under the known constraints of the object. More precisely, the measurements of the resulting subsequence could be measurements of a traveled path that is possible under the physical constraints.

We present algorithms for *concatenable consistency*, where consistency of the concatenation of two consistent subsequences sharing an end resp. start vertex is guaranteed. In addition, we provide an algorithm for *conditional consistency*, where some state variables influence whether two conditional consistent subsequences sharing an end resp. start vertex are consistent when concatenated. We then perform an extensive experimental evaluation, comparing the proposed methods with established baseline methods.

This chapter is based on
Bram Custers, Mees van de Kerkhof, Wouter Meulemans, Bettina Speckmann, and Frank Staals. Maximum physically consistent trajectories. *ACM Transactions on Spatial Algorithms and Systems*, 7(4):1–33, 2021. DOI: `10.1145/3452378`

This paper won the best paper award at the 27th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems.

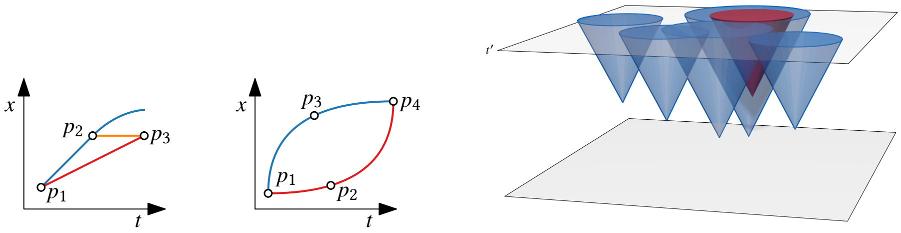


Figure 1.6 We leverage physical consistency for detecting outliers. (left) Consistency between measurements in space and time (right) Consistency for two-dimensional motion.

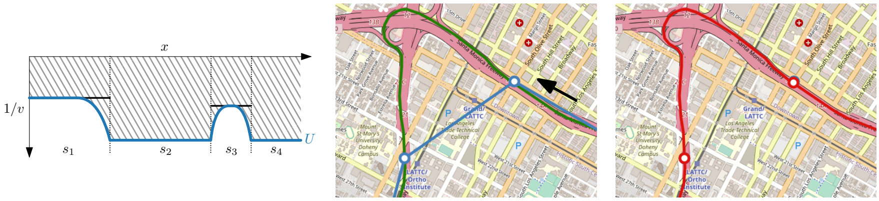


Figure 1.7 Extended physics model (left) to use for map-matching sparse inputs (middle, blue line) to the road network to get a route (right, red line).

Physically Consistent Map-Matching

In Chapter 3, we extend the idea of physical consistency of the previous chapter, and consider the problem of map-matching of trajectories to a road network. We increase the amount of contextual data that is considered, leveraging the local speed limits on the road network to bound the maximum speed an entity may achieve.

We provide an algorithm for verifying that a sequence of measurements along with routes connecting consecutive measurements is physically consistent. We conjecture that the general problem is hard. Hence, we provide a heuristic algorithm for solving the map-matching problem while leveraging physical consistency (Fig. 1.7). In our algorithm, we use a k-fastest paths approach to generate candidate paths for the input trajectory, and leverage our physical consistency check for paths in the network to eliminate inconsistent paths. We compare our approach with two baseline Hidden Markov Model approaches.

This chapter is based on
Bram Custers, Wouter Meulemans, Marcel Roeloffzen, Bettina Speckmann, and Kevin Verbeek. Physically consistent map-matching. In *Proceedings of the 30th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, 2022. To appear.

Route Reconstruction from Traffic Flow via Representative Trajectories

In Chapter 4, we look into the problem of making a set of trajectories more complete. In particular, we consider traffic in a road network, collected as a set of trajectories. In addition, we assume that we are given *checkpoint data*: measurements of the total number of vehicles that traversed specific points in the network for specific time

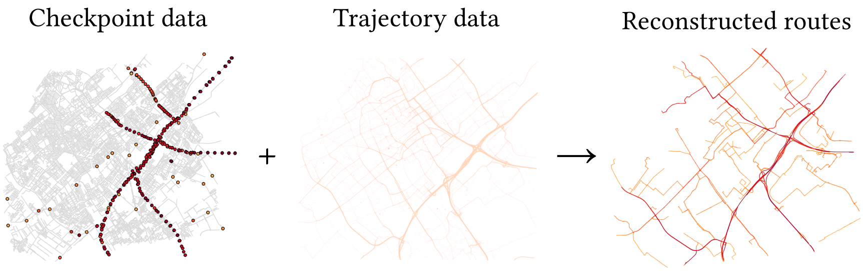


Figure 1.8 Reconstructing routes using checkpoint and trajectory data

intervals. We now investigate whether we can give a more complete set of routes (trajectories on the network), assuming that the checkpoint data is fairly complete and the set of trajectories are representative for traffic in the network (see Fig. 1.8).

We formally define the problem and show that several simplified variants of the problem are NP-hard. Hence, we develop a heuristic approach that reconstructs the routes based on the checkpoint data. We compare our proposed heuristic to baseline approaches that have varying degree of remaining faithful to the representative trajectories. We find that the approach is promising in terms of remaining faithful to the input trajectories and compactly representing major routes in the data set.

This chapter is based on
Bram Custers, Wouter Meulemans, Bettina Speckmann, and Kevin Verbeek. Route reconstruction from traffic flow via representative trajectories. In *Proceedings of the 29th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, pages 41–52, 2021. DOI: `10.1145/3474717.3483650`

Coordinated Schematization for Visualizing Mobility Patterns on Networks

In Chapter 5, we propose a way to visualize the major mobility patterns present in a trajectory data set, where we assume that the entities all drive vehicles subject to a road network. In particular, we want to summarize the trajectory data set, allowing for more intuitive insight into the global structure of the routes present in the data set. To achieve this, we want to maximally reduce visual clutter before visualizing the data set, while in addition visualizing the data set on the road network (Fig. 1.9).

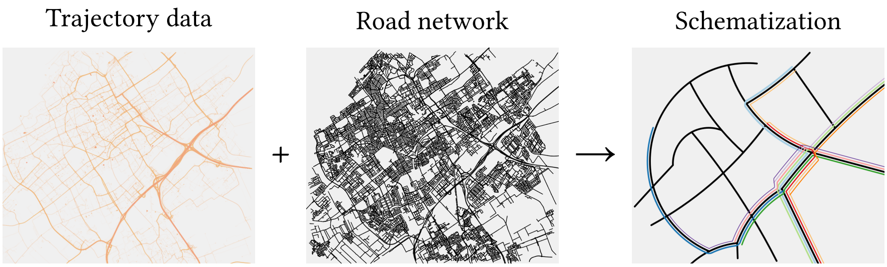


Figure 1.9 Coordinated schematization of trajectory data set and network to produce a schematization that summarizes the trajectory data set.

We propose a coordinated schematization pipeline, where we simplify the road network, while coordinating the changes of the road network with the trajectories, thus maintaining stronger cohesion between the trajectories and the road network. We propose to use a data driven selection of the initial road network, followed by simple local operators that reduce the complexity of the road network, while maintaining a mapping of each trajectory to the simplified road network.

This chapter is based on
Bram Custers, Wouter Meulemans, Bettina Speckmann, and Kevin Verbeek. Coordinated schematization for visualizing mobility patterns on networks. In *Proceedings of the 11th International Conference on Geographic Information Science (GIScience 2021)*, LIPIcs, 2021. DOI: `10.4230/LIPIcs.GIScience.2021.II.7`

This paper was nominated for the best paper award at the 11th International Conference on Geographic Information Science (GIScience 2021).

Schematizing Orthogonal Polygons

In Chapter 6, we explore schematizing orthogonal polygons. We introduce the idea of schematizing orthogonal polygons under the *minimum homotopy area* measure (see Fig. 1.10). This measure is more sensitive to topological changes than established measures, which improves the schematization for certain input polygons. We provide an algorithm that computes the optimal schematization under the minimum homotopy area in $O(n^4(k + \log n))$ time for polygons of size n and a target output schematization of size k.

We analyze the behaviour of simplifications under the minimum homotopy area, conjecturing that it is NP-hard to find an optimal simplification while retaining sim-

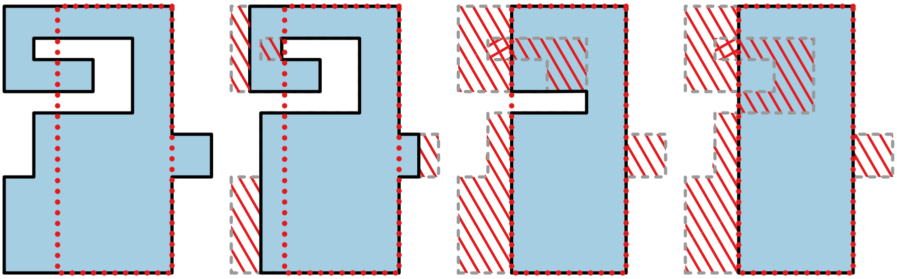


Figure 1.10 The homotopy area between the blue polygon and the polygon with the red dotted boundary, given by the hatched areas.

plicity of the input polygon. Finally, we develop a greedy algorithm for computing the minimum homotopy area, where we can show that on uniform orthogonal staircases, the proposed greedy algorithm is a constant factor approximation for the optimal schematization.

This chapter is partially based on
Bram Custers, Jeff Erickson, Irina Kostitsyna, Wouter Meulemans, Bettina Speckmann, and Kevin Verbeek. Orthogonal schematization with minimum homotopy area. In *Proceedings of the 36th European Workshop on Computational Geometry (EuroCG 2020)*, pages 451–457, 2020. URL: `https://www1.pub.informatik.uni-wuerzburg.de/eurocg2020/data/uploads/papers/eurocg20_paper_64.pdf`

▸ 1.5 Other results

Apart from the results in this thesis, the author of this thesis worked on data structures for visibility queries in polygons [19] and a GIS cup challenge, where a heuristic was developed to maximize the performance of taxis picking up passengers given historical data [21]. The latter result won second place in the challenge.

Chapter 2

Maximum Physically Consistent Trajectories

As we discussed in the introduction, a key challenge when working with trajectory data is appropriately handling the noise that is inherently present in the trajectories. For example, GPS readings notoriously stray far from their real location in urban canyons, resulting in trajectories with multiple significant outliers. These outliers pose problems for many analysis techniques such as clustering or grouping, and they skew the results of statistical methods. Hence, it is common practice to try to eliminate outliers in a preprocessing step.

There are a variety of methods to remove outliers. Some techniques, such as smoothing or averaging the data, have a possibly negative impact on the complete trajectory. Others, such as map matching, are applicable only to trajectories that can be expected to coincide with a road network. In this chapter, we focus on *outlier detection*, that is, we describe algorithms that identify outliers which can subsequently be handled as appropriate for the application.

Specifically, we aim to identify outliers by employing specific contextual information for the trajectory: the physical constraints of the moving (real-world) entity. We define a *physics model* as a model that describes how an entity can move, given its physical constraints. We consider two measurements within a trajectory to be *consistent* for a particular physics model, if the corresponding entity could have traveled between the two measured locations in the time between the two measurements assuming the physics model. In this chapter we present optimal algorithms to compute maximal consistent subtrajectories according to different (simplified) physics models, thereby identifying the omitted measurements as outliers under the chosen model. Before describing our results in more detail, we first introduce the necessary notation and formally state the problem.

Notation Let v^- be the minimum speed, or velocity, that the entity can achieve, and let v^+ be the maximum speed that the entity can achieve. Similarly, let a^- and a^+ be the minimum and maximum possible acceleration. These speed and acceleration bounds represent physical bounds, and thus the entity cannot exceed them at any time, even in between consecutive time stamps t_i and t_{i+1}. The actual continuous motion of an entity is assumed to be a continuous path $\pi : [t, t'] \longrightarrow \mathbb{R}^d$ over time interval $[t, t']$ through d-dimensional space (typically, $d = 2$). We say that a path π *adheres to* the physics model if it never exceeds the bounds. For example, the speed is always in $[\mathrm{v}^-, \mathrm{v}^+]$ and the acceleration is always in $[a^-, a^+]$. A sequence of measurements $T = \langle p_1, \dots, p_n \rangle$ is *consistent* with the physics model, denoted $C(T)$, if and only if there exists at least one *witness*: a path $\pi : [t_1, t_n] \longrightarrow \mathbb{R}^d$ such that (i) for all $i \in \{1, \dots, n\}$, $\pi(t_i)$ coincides with location p_i, and (ii) π adheres to the physics model. We sometimes write $C(p_i, p_j)$ instead of $C(\langle p_i, p_j \rangle)$.

We use *subtrajectory* or *subsequence* of a trajectory T to refer to a subset of the measurements in the same order as in T; note that these measurements do not need to be consecutive in T.

Formal problem statement Given a trajectory T and a physics model, compute a maximum-size subsequence S of T such that S is consistent with the given model. Let the size of S be ℓ. Then, the trajectory T contains $k = n - \ell$ erroneous measurements, or *outliers*, according to the physics model.

Concatenability Regardless of the physics model, if a sequence T is consistent, then so is any subsequence S of T. But we cannot necessarily construct a consistent subsequence from smaller ones: the concatenation $\langle p_1, \dots, p_n = q_1, \dots, q_m \rangle$ of two consistent subsequences $T = \langle p_1, \dots, p_n \rangle$ and $U = \langle q_1, \dots, q_m \rangle$ with $p_n = q_1$ is not necessarily consistent. We call a physics model *concatenable* if this is the case. An example of a concatenable model is one that limits only the speed of the entity. Concatenable models generally allow more efficient algorithms.

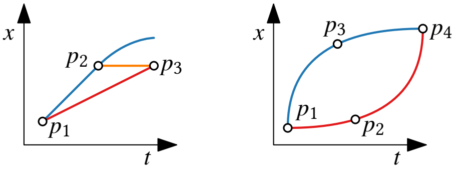


Figure 2.1 (Left) In an acceleration-bounded model $\langle p_1, p_2, p_3 \rangle$ is not consistent, even though $\langle p_1, p_2 \rangle$ and $\langle p_2, p_3 \rangle$ (and even $\langle p_1, p_3 \rangle$) are. (Right) A consistent subtrajectory through p_2 (red) may require a different speed at p_4 than a subtrajectory that includes p_3 (blue).

Not all physics models are concatenable: for example, a model limiting both the speed and the acceleration is not concatenable. See Fig. 2.1 (left): both $T = \langle p_1, p_2 \rangle$ and $U = \langle p_2, p_3 \rangle$ are consistent, but $\langle p_1, p_2, p_3 \rangle$ is not. The main problem is that sequences U and T essentially require the entity to have two different speeds at p_2. The two subtrajectories U and T are concatenable in the acceleration-bounded model, under the condition that they have the same speed at their common measurement. To capture this, we define a notion of *conditional consistency*, denoted $C(T \mid \gamma)$, in which a trajectory T is consistent, provided that it has a witness satisfying condition γ. In case $C(T \mid \gamma)$ and $C(U \mid \gamma')$ imply that the concatenation of T and U is consistent, we say that the physics model is *conditionally concatenable*. Hence, the model with bounded acceleration is conditionally concatenable, using the condition that the speed at the common measurement is the same. The speeds attainable at a certain measurement may depend on the subtrajectory so far (see Fig. 2.1, right).

Results and organization We present three algorithms and the results of computational experiments investigating the efficacy of our methods. Specifically, in Section 2.1 we describe a simple, optimal algorithm that runs in $O(nk)$ time for any concatenable physics model allowing $O(1)$ consistency checks between two measurements. We then describe a more efficient algorithm which runs in $O(n \log n \log^2 k)$ time for the speed-bounded model in Section 2.2. Our final algorithm, described in Section 2.3, uses an acceleration-bounded model, that can optionally also bound the speed. This algorithm runs in $O(nk^2 \log k)$ time under mild assumptions, that are validated by our experiments. We also present a variant of this algorithm that introduces slack in the physics model to obtain an efficient approximate algorithm that achieves the given worst-case running time without assumptions.

In Section 2.4, we discuss the results of a series of computational experiments on real-world data using our open-source implementation[1]. Specifically, we compare the quality of our algorithms to simple greedy approaches and conclude that our algorithms are more reliable, especially for trajectories with more than minor levels of noise. We also observe that the speed-bounded model approximates the acceleration-bounded model, though there is some dependency on the dataset. Finally, we also briefly investigate how sensitive our results are to the model parameters: though speed bound is quite sensitive, the acceleration bounds have comparatively little influence on the number of outliers detected. We conclude with a discussion of our results in Section 2.5.

Related work Outlier detection is necessary to cope with imprecise data. Hence, many different methods have been developed for various contexts. A general survey of outlier detection is given in [64]; see also [57] for a survey focusing on data with a temporal component, including trajectories. For trajectories, outlier detection has mainly focused on finding outlying trajectories in sets of trajectories [54, 78, 81, 138], and not on finding outlying measurements in one trajectory. At a glance, detecting outlying measurements resembles trajectory simplification and trajectory smoothing, both well-studied topics: refer to [139] for a survey. However, simplification generally aims to minimize the number of measurements while still accurately describing the trajectory: this typically retains outliers as these are “salient”.

Physics models are often used in trajectory processing. Kalman filtering [80], for example, is based upon a linear model for physical motion; its extensions handle more complex, nonlinear models. Note however, that Kalman filtering changes the measurement positions rather than selecting a consistent subset. In a similar vein, physics models are used to reconstruct trajectories from data, replacing subtrajectories that cannot be physically realized with ones that can [92, 100]. Here, unrealistic subtrajectories are detected using a local time window, sliding over the trajectory.

Given a trajectory and a physics model, we aim to determine the maximum number of measurements that can be explained through a path adhering to the model. As such, our problem bears some resemblance to two other problems: computing a longest common subsequence (LCSS) and map matching. The former asks to compute the maximum subsequence of two strings [12] and has also been used to compute trajectory similarity [120]. Contrasting our approach, LCSS requires that both trajectories are known. The latter, map matching, is the problem of determining the driven route through a street network, given a noisy trajectory. Myriad

---

[1] Released as part of the MoveTK library, `https://movetk.win.tue.nl/`.

algorithms exist, e.g. [3, 94]; see [139] for an overview. Dealing with noise naturally arises in this application. Though we do not investigate this here, explicit outlier removal before map matching may improve results of simple and faster algorithms; postprocessing map matching results using our methodology may give rise to more realistic results. However, the primary difference is that our method does not rely on knowledge of the street network: the space of potential paths is defined implicitly and as such our methodology is more broadly applicable to movement that does not follow a predefined network (pedestrians, birds, recreational aviation).

▸ 2.1 Concatenable consistency model

We assume an arbitrary concatenable physics model that allows consistency checks between two measurements in $O(f(n))$ time for some function f; typically, $f(n) = O(1)$. We follow the methodology of the Imai-Iri line-simplification algorithm [68]. Let $G = (V, A)$ be a directed acyclic graph with a vertex v_i for each measurement p_i of T and an edge from v_i to v_j if and only if $C(p_i, p_j)$. This graph has $O(n^2)$ edges; each can be tested in $O(f(n))$ time. By concatenability, a path in G describes a consistent subsequence. Since G is directed and acyclic, we compute a longest path in G, and thus a maximum consistent subsequence of T, in $O((|V| + |A|)f(n)) = O(n^2 f(n))$ time.

We now develop an output-sensitive variant of this algorithm. Rather than constructing the full graph, we build a subgraph in which each vertex v has at most one incoming edge (u_v, v). In particular, u_v and v are the last measurements of a longest consistent subsequence T_v ending in v. Let ℓ_v denote the length of T_v.

2.1.1 Theorem. *Consider a concatenable physics model that allows checking the consistency of a pair of measurements in $O(f(n))$ time. A maximum consistent subsequence of a trajectory T with n measurements can be computed in $O(nkf(n))$ time, where k is the number of outliers.*

Proof. We handle the measurements in chronological order, maintaining a linked list $\mathcal{L}$ that stores, for each handled measurement v, the value ℓ_v and the predecessor u_v in T_v. $\mathcal{L}$ is ordered by the lengths ℓ_v in descending order. For a new measurement w, we traverse $\mathcal{L}$ to find the first measurement v consistent with w. Since $\mathcal{L}$ is ordered, we have thus found a longest consistent subsequence of length $\ell_v + 1$ ending in w. We now walk backwards in $\mathcal{L}$ and add w to the appropriate place in the list.

After we have handled all measurements, the maximum consistent subsequence can be retrieved in $O(n - k)$ time, by starting at the head of the list and following the predecessor pointers. For each of the $n - k$ measurements that end up in the longest subsequence, we perform one successful check preceded by at most k failed checks, and for the k outliers we perform at most n checks. This gives a total of $O(nkf(n))$ time performing the checks. The time for inserting a measurement in $\mathcal{L}$ can be charged to the number of checks it performs: this takes $O(nk)$ time in total. Hence, the total running time for the algorithm is $O(nkf(n))$. □

As the speed-bounded model allows to check consistency between two measurements in constant time, we thus obtain the following running time for this model. We can, however, improve upon this algorithm in case the trajectory has many outliers, as discussed in the next section.

2.1.2 **Corollary.** *For the speed-bounded model, a maximum consistent subsequence of a trajectory T with n measurements can be computed in $O(nk)$ time, where k is the number of outliers.*

▸ 2.2 The speed-bounded model in 2D

We now consider the speed-bounded model, with maximum speed v^+, and trajectories in $\mathbb{R}^2$. We present an $O(n \log n \log^2 k)$-time algorithm to find a maximum consistent subtrajectory in this model. To this end we develop an insertion-only data structure that, given a measurement q, can determine the length of a maximum consistent subsequence ending at q in $O(\log^3 n)$ time. Insertions are supported in $O(\log^3 n)$ time. By incrementally building the data structure in chronological order, we can determine the maximum consistent trajectory in $O(n \log^3 n)$ time. With a more careful analysis this can be improved to $O(n \log n \log^2 k)$ time.

▸ 2.2.1 A consistency data structure

Let P be a subset of measurements from T, and let $\hat{t}$ be the time of the last measurement in P. We develop a data structure $\mathcal{D}$ that can efficiently answer *consistency-queries* on P. That is, for a given new query measurement q occurring at time $t \geq \hat{t}$, we can test if there is a measurement in P consistent with q. We view the measurements in P as points in $\mathbb{R}^3$, with the third axis being time, that is, $p_i = (x_i, y_i, t_i)$. Measurements p_j, with $j > i$, that are consistent with p_i must lie inside a cone that starts at p_i and has radius $\mathrm{v}^+(t - t_i)$ at time $t \geq t_i$; see Fig. 2.2. We call this cone the *reachable region* of p_i; testing if p_j is in the reachable region of p_i takes $O(1)$ time.

To determine if a measurement q at time $t_q \geq \hat{t}$ is consistent with any measurement of P we use an *additively weighted Voronoi diagram* (AWVD) [52, 48]. Given a set of

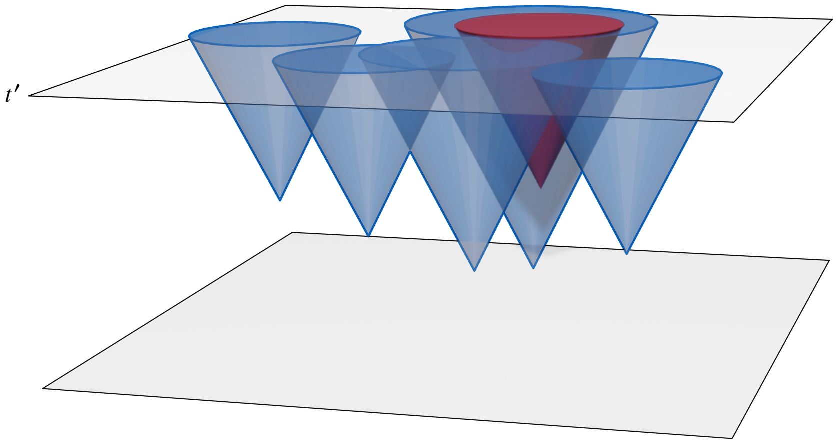


Figure 2.2 Each measurement defines a reachable region (a cone), that intersects the plane at time t' in a disk. These disks define an AWVD. A measurement p_j is consistent with an earlier measurement p_h if (and only if) its cone (shown in red) is contained in the cone of p_h.

disks with centers $\{v_1, \ldots, v_l\}$ and radii $\{r_1, \ldots, r_l\}$, this diagram partitions the plane into cells $\{c_1, \ldots, c_l\}$ associated with the disks, such that for any point $v \in c_i$ it holds that $d(v, v_i) - r_i \leq d(v, v_j) - r_j$, for all $v_j \neq v_i$. Here, d is a distance measure (in our case the Euclidean distance), and equality holds only on boundaries between cells.

We construct an AWVD on the measurements in P by using the locations as the centers and picking $r_i = \mathrm{v}^+(t' - t_i)$ for every measurement for some arbitrary $t' > t_n$. Observe that a measurement p_j is consistent with p_h if the reachable region of p_j at t' is inside the reachable region of p_h at t' (see Fig. 2.2). We preprocess the AWVD for point location queries. Let $\mathcal{D}$ denote the resulting data structure, which we refer to as a *consistency data structure*. We can now query $\mathcal{D}$ with a new measurement $q = p_q$, giving us a previous measurement p_c and a distance s_c between the disks (p_q, r_q) and (p_c, r_c), given by $s_c = d(p_q, p_c) - r_q - r_c$, where d measures the Euclidean distance between the points in the plane, that is, ignoring the temporal component. The following lemma then gives us that $\mathcal{D}$ can be used to answer consistency queries.

2.2.1 Lemma. *Let $\mathcal{D}$ be a consistency data structure on a set P of measurements and let $q = p_q$ be a query measurement, resulting in measurement p_c on $\mathcal{D}$. If $s_c \leq -2r_q$*

for the resulting distance s_c *of* p_c *with* p_q*, then* p_c *is consistent with* q*. Otherwise, no measurement in* P *is consistent with* q*.*

Proof. We denote $w_j = (x_j, y_j)$ for compactness. Let $p_q = (x_q, y_q, t_q)$ be the q-th measurement in a trajectory and let $\mathcal{D}$ be the consistency data structure on a subset $P \subseteq \{p_1, \dots, p_{q-1}\}$. Let p_c be the measurement associated with the found cell in $\mathcal{D}$ containing (x_q, y_q) and let $s_c = d(w_q, w_c) - r_q - r_c$.

Since $s_c = d(p_q, p_c) - r_q - r_c$, we can rewrite the inequality $s_c \leq -2r_q$ to $d(p_q, p_c) \leq r_c - r_q$. Applying the definition of r_c and r_q, the right-hand side can be rewritten to $\mathrm{v}^+(t' - t_c) - \mathrm{v}^+(t' - t_q) = \mathrm{v}^+(t_q - t_c)$. In other words, we have that the distance between the measurements is at most the maximum distance that the object can travel in the given time span. Hence, p_q is consistent with p_c.

Analogously, if the given inequality does not hold, the distance between the two measurements is larger than what the object can cover when traveling at maximum speed: they are not consistent.

To show that no other measurement in P can be consistent, observe that the definition of the AWVD gives us that $d(p_q, p_c) - r_c \leq d(p_q, p_k) - r_k$ for all $p_k \in P \setminus \{p_c\}$. Subtracting r_q from both sides, we get that $d(p_q, p_c) - r_c - r_q = s_c \leq d(p_q, p_k) - r_k - r_q$. Thus, if $s_c > -2r_q$, we must also have that $d(p_k, p_c) > r_k - r_q$. Thus, all measurements p_k are not consistent with measurement p_q. □

We can construct the AWVD for a set of m measurements and preprocess this AWVD for point-location queries in $O(m \log m)$ time [48, 52]. The resulting data structure uses $O(m)$ space, and can answer point-location queries in $O(\log m)$ time. Since a single consistency check takes constant time, we can also answer consistency queries in $O(\log m)$ time.

▸ 2.2.2 Supporting insertions

Next, we describe how to extend our consistency data structure to support insertions. Testing whether a measurement is consistent with any previous measurement of a subsequence of T is a decomposable search problem. Thus, we use the approach by Bentley and Saxe [11] to turn our consistency data structure into an efficient insertion-only data structure.

For a set of m measurements, we maintain $O(\log m)$ instances of our static data structure $\mathcal{D}_1, \dots, \mathcal{D}_{O(\log m)}$ (see Fig. 2.3). Every measurement is in one of these $O(\log m)$ data structures. Data structure $\mathcal{D}_i$ has size 2^i. On insertion, we create a new $\mathcal{D}_1$ with

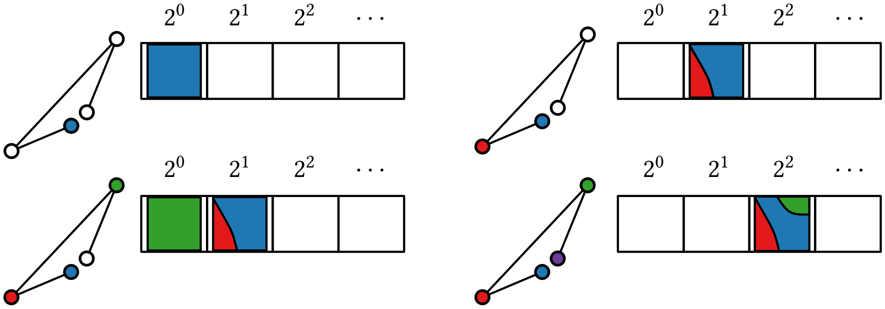


Figure 2.3 Inserting elements using Bentley-Saxe. The colored measurements in the trajectory are the elements in the insertion-only consistency data structure.

the inserted measurement. When we get two data structures of same size 2^i, we remove both and replace them by a single data structure of size 2^{i+1}. We repeat this process until all data structures have a unique size. To answer a query we simply query all $O(\log m)$ data structures.

The above construction together with the consistency query structure gives $O(\log^2 m)$ time for a query and $O(\log^2 m)$ amortized time for an insertion. These bounds can be made worst case as well [99]. We summarize our results in the following lemma.

2.2.2 Lemma. *There is a consistency data structure $\mathcal{D}$ that can store a subset P of m measurements from T and can answer consistency queries for query points q at time $t \geq \hat{t}$, in $O(\log^2 m)$ time, and supports insertions in $O(\log^2 m)$ time. Here, $\hat{t}$ denotes the time of the last measurement currently in P. The data structure uses $O(m)$ space.*

▸ 2.2.3 Maximum subsequence queries

We now use the data structure from Lemma 2.2.2 to build a dynamic data structure that, for a new query measurement $q = p_q$ can determine the length ℓ_q of a longest consistent subsequence $T_q \subseteq P$ ending at q. We store the measurements in $p \in P$ in the leaves of a balanced binary tree $\mathcal{T}$, ordered by the length ℓ_p of the longest consistent subsequence ending in p. Each internal node v with right child r corresponds to a subset $P_v \subseteq P$, and stores the minimum ℓ_p, with $p \in P_r$, occurring in its right subtree, and a consistency data structure $\mathcal{D}_v$ built on the set P_r (see Fig. 2.4).

Given a query measurement q, we find a measurement $u \in P$ consistent with q with maximum length ℓ_u. It then follows that a maximum-length consistent subsequence T_q ending in q has length $\ell_u + 1$, and that u is the predecessor of q in T_q. To find u we start at the root v and query $\mathcal{D}_v$ to test whether any measurement in the right subtree is consistent with q. If so, we repeat the process in the right child. If not, we move to the left child. This way we get the longest-path measurement that is consistent with our query measurement q.

To insert a new measurement q, we find the leaf corresponding to length ℓ_q and insert q in the appropriate associated data structures of all ancestors along this root to leaf path. To keep the tree $\mathcal{T}$ balanced, we implement it using a BB[α] tree [14, 95]. When a subtree rooted at a node v becomes unbalanced, we rebuild it and its associated data structures from scratch.

The lemma below proves that this data structure can be implemented efficiently.

2.2.3 **Lemma.** *There is a data structure $\mathcal{T}$ that can store a subset P of m measurements from T and that can find, given a query measurement q at time $t_q \geq \hat{t}$, (the length ℓ_q of) a longest consistent subtrajectory T_q ending in q in $O(\log^3 m)$ time. The data structure uses $O(m \log m)$ space and supports insertions in $O(\log^3 m)$ amortized time. Here, $\hat{t}$ denotes the time of the last measurement currently in P.*

Proof. To answer a query we follow a path from the root down to a leaf, and query the associated data structure at each node. Each such query takes $O(\log^2 m)$ time (Lemma 2.2.2), and thus the total query time is $O(\log^3 m)$. Since each measurement is stored in the associated data structure of $O(\log m)$ nodes, the total space use is $O(m \log m)$, and the total direct cost of an insertion is $O(\log^3 m)$. To prove the lemma, we need to bound the costs due to rebalancing operations.

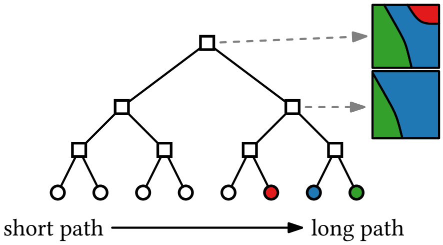


Figure 2.4 Data structure for maximum subsequence queries.

Assume an insertion of a node triggers a rebalance operation for a subtree v of m_v elements, and $C(m_v)$ the total construction time: the time that it takes to acquire all elements in the subtree and construct a perfectly balanced tree (with its associated data structures). By the BB$[\alpha]$ definition, $(1-2\alpha)m_v-2$ insertions must have occurred in v to trigger the rebalance operation. This implies that the amortized rebalance time per insertion is $\frac{C(m_v)}{(1-2\alpha)m_v-2}$.

Consider the tree after rebalancing. At height $h > 0$ we have intermediate nodes, each requiring a data structure $\mathcal{D}$ constructed on $O(2^h)$ elements. There are $O(m_v/2^h)$ nodes at height h. In total, to construct nodes with height h, we require $O((m_v/2^h)2^h \log^2(2^h)) = O(m_v h^2)$ time. Let $H = O(\log m_v)$ be the height of the entire subtree; summing up the construction time of all heights in the subtree gives the total construction time $C(m_v) = \sum_{h=1}^{H} O(m_v h^2) = O(m_v H^3) = O(m_v \log^3 m_v)$. Amortized, this construction cost is $O(\log^3 m)$. Hence the entire insertion time is also $O(\log^3 m)$.[2] □

▸ 2.2.4 Maximum consistent subtrajectories

To compute a maximum-length consistent subtrajectory of T, we process all measurements in chronological order. For each we simply query the data structure from Lemma 2.2.3, and then insert it. This results in an $O(n \log^3 n)$-time algorithm. Next, we show that we can improve this to $O(n \log n \log^2 k)$, with k the number of outliers.

2.2.4 Lemma. *For two consistent measurements p_i and p_j with $i < j$, the reachable region for p_j for all $t > t_j$ is contained in the reachable region of p_i.*

Proof. In the 3-dimensional space (the third being time), the reachable region of each measurement is an upward cone starting at the measurement, with slope v^+. As p_j is consistent with p_i, the former lies inside the latter's cone. As their direction and slope are the same, the cone for p_j is thus contained in the cone for p_i.

We can equally see this in 2-dimensional space. Consider an arbitrary time $t > t_j$. A hypothetical measurement p^* at time t consistent with p_j must be within distance $\mathrm{v}^+(t - t_j)$. Since p_i and p_j are consistent, we know that their distance is at most $\mathrm{v}^+(t_j - t_i)$. Through triangle inequality, we thus know that the distance between p_i and p^* is at most $\mathrm{v}^+(t - t_j) + \mathrm{v}^+(t_j - t_i) = \mathrm{v}^+(t - t_i)$. This readily implies that p^* is consistent with p_i as well. □

---

[2] Note that in our improved bound in Theorem 2.2.5 the total reconstruction time, $C(m_v)$, is simply $O(m_v \log m_v \log^2 k)$, as rebuilding the associated data structure of a node takes $O(m_v \log^2 k)$ time rather than $O(m_v \log^2 m_v)$ time.

From the definition of the AWVD, we know that if a disk c_1 is strictly inside another disk c_2, then c_1 will have an empty associated cell in the diagram. Combining this with Lemma 2.2.4 shows that, any subset of $m \geq 1$ measurements thus produces a diagram with at most $\min(m, k)$ cells. Hence, a static consistency data structure uses only $O(\min(m, k))$ space, and querying it requires $O(\log(\min(m, k)))$ time. When we insert a new measurement p_j into our insertion-only data structure, we first query the data structure to decide whether p_j is consistent with some earlier measure p_i. If so, we simply discard p_j rather than inserting it; even when inserting additional points, the cell of p_i will contain that of p_j. The query and insertion time therefore both become $O(\log^2 \min(m, k))$.

It now follows that the associated data structure $\mathcal{D}_v$ of every node in $v \in \mathcal{T}$ has size only $O(\min(n_v, k))$, thus querying it requires only $O(\log^2 k)$ time, and thus $O(\log n \log^2 k)$ time in total. Similarly, inserting a new measurement takes amortized $O(\log n \log^2 k)$ time.

2.2.5 Theorem. *Given a 2D trajectory T with n measurements, of which k are outliers, we can compute a maximum consistent subsequence of T for the speed-bounded model in $O(n \log n \log^2 k)$ time.*

▸ 2.3 The acceleration-bounded model

We now consider 1D trajectories where each measurement is of the form $p_i = (x_i, t_i)$. We assume a physics model where both velocity and acceleration are restricted. The velocity must lie in the range $[\mathrm{v}^-, \mathrm{v}^+]$ for constants $\mathrm{v}^-, \mathrm{v}^+$. In addition, the acceleration must lie in the range $[a^-, a^+]$ for constants a^-, a^+. For simplicity, we assume $a^- < 0$ and refer to deceleration as acceleration with a negative value.

For this acceleration-bounded model, we can still test in constant time if two points p_i and p_j are consistent: we can check if the distance between the two measurements can be traveled using velocities that lie in the range $[\mathrm{v}^-, \mathrm{v}^+]$. If there exists a velocity at p_i such that the required velocity at p_j can be reached by accelerating, then the pair $\langle p_i, p_j \rangle$ is consistent. Recall, however, that a physics model that limits acceleration is not concatenable: there may be a triplet of measurements $\langle p_1, p_2, p_3 \rangle$ for which the measurements are pairwise consistent, but the entire sequence is not (see Fig. 2.1 (left) for an example). Hence, we cannot use the algorithm described in Section 2.2.

In Section 2.3.1 we describe a dynamic programming algorithm which explicitly computes the velocities achievable at every measurement and, using these velocities, finds the length of a maximum-length consistent subtrajectory. In Section 2.3.2

we show how to retrieve the actual consistent trajectory. The running time of the dynamic program and of the retrieval procedure depends on the maximal fragmentation of the velocity intervals which can arise during the DP. In Section 2.3.3 we first argue that this number can be as large as $\Omega(n)$ for a linear number of sets of velocities. It is easy to see that the maximal fragmentation is at most $O(2^n)$, however, it is unlikely that this bound would ever be reached in practice. In the following we consider an acceleration-bounded model with some slack in the acceleration bounds, modeling real-world imprecision. This slack allows us to prove a linear upper bound for the fragmentation of any set of velocities. Finally, in Section 2.3.4 we explain how to extend the acceleration-bounded model to dimensions greater than one.

▸ 2.3.1 Computing the maximum length of a physically consistent subtrajectory

A subtrajectory T is generally not concatenable with another subtrajectory T' under the acceleration-bounded model, but is conditionally concatenable with T' when the velocities at measurements that they have in common are the same. Intuitively, this follows from the fact that a bound on the acceleration prevents (discontinuous) jumps in velocity. Based on this, we observe the following:

2.3.1 Observation. A (sub)trajectory $S = \langle p_1, \ldots, p_m \rangle$ is consistent in the acceleration-bounded model if and only if there are velocities $\langle v_1, \ldots, v_m \rangle$ such that for all $i \in \{1, \ldots, m-1\}$ we have that $C(p_i, p_{i+1} \mid \mathrm{v}_i = v_i, \mathrm{v}_{i+1} = v_{i+1})$.

Observation 2.3.1 implies that our problem has an optimal substructure. Suppose we have found all subtrajectories of some length ℓ that are consistent. If we now want to know whether a subtrajectory of length $\ell + 1$ exists, we have to determine only whether there is a measurement p' such that the observation holds for one of the subtrajectories when we add p' at the end. That is, there should be witness paths for both the subtrajectory and the trajectory between the last measurement of the subtrajectory and p', that have a common velocity at the last measurement of the subtrajectory. Hence, we can apply the dynamic programming paradigm to find the optimal length for which a subtrajectory is physically consistent.

More formally, for each measurement p_i and each possible length $\ell \in \{1, \ldots, n\}$, we maintain the set of velocities $\mathcal{I}(\ell, i)$ such that for every velocity $v \in \mathcal{I}(\ell, i)$, a subsequence $S = \langle \ldots, p_i \rangle$ ending at p_i of length ℓ exists that is physically consistent and has velocity v at p_i, so that $C(S \mid \mathrm{v}_i = v)$. Let ℓ^* be the maximum value, such that a measurement p_i exists for which some set $\mathcal{I}(\ell^*, i)$ is non-empty. It follows that the maximum consistent subtrajectory of T has length ℓ^*.

Given the set of possible velocities $\mathcal{I}(\ell, h)$ at p_h, we can then determine whether a consistent subsequence of length $\ell + 1$ exists that ends at a later measurement p_i by using the conditional concatenability property: if we find velocities $v_h \in \mathcal{I}(\ell, h)$ and $v \in [\mathrm{v}^-, \mathrm{v}^+]$ such that $C(p_h, p_i \mid \mathrm{v}_h = v_h \wedge \mathrm{v}_i = v)$, then a consistent subsequence $\langle \dots, p_h, p_i \rangle$ of length $\ell + 1$ exists. Hence, we obtain the following recurrence for $\mathcal{I}(\ell, i)$.

$$\mathcal{I}(\ell, i) = \begin{cases} \emptyset, & i < \ell \\ \{v \mid \exists h \,:\, C(p_h, p_i \mid \mathrm{v}_i = v)\}, & \ell = 2 \\ \{v \mid \exists h \,:\, C\left(p_h, p_i \mid \mathrm{v}_h \in \mathcal{I}(\ell - 1, h), \mathrm{v}_i = v\right)\}, & \ell > 2 \end{cases}$$

Moreover, we prove in Lemma 2.3.2 that when the entity directly travels from p_i to p_j, and leaves p_i with velocity v_i, the possible velocities with which it can arrive at p_j form a connected interval. Thus, the sets $\mathcal{I}(\ell, i)$ are actually sets of intervals.

2.3.2 Lemma. *Let p_i and p_j be measurements with $t_i < t_j$, and let $v_1 \le v \le v_2$ be velocities. If $C(p_i, p_j \mid \mathrm{v}_j = v_1)$ and $C(p_i, p_j \mid \mathrm{v}_j = v_2)$, then we also have $C(p_i, p_j \mid \mathrm{v}_j = v)$.*

Proof. $C(p_i, p_j \mid \mathrm{v}_j = v_1)$ and $C(p_i, p_j \mid \mathrm{v}_j = v_2)$ imply that there are two witnesses: paths $\pi_1(t)$ and $\pi_2(t)$ between p_i and p_j that travel $\Delta x = x_j - x_i$ distance, obey the physics model and have velocity v_1 respectively v_2 at p_j. Let $a_1(t)$ and $a_2(t)$ denote the acceleration functions describing these paths.

The traveled distance Δx between t_i and t_j using any acceleration function $a'(t)$ and velocity v' at p_j is given by

$$\Delta x = (t_j - t_i)\left(v' - \int_{t_i}^{t_j} a'(t)\mathrm{d}t\right) + \int_{t_i}^{t_j}\int_{t_i}^{t} a'(t')\mathrm{d}t'\mathrm{d}t \qquad (2.1)$$

Any new path π^* which we create using convex combinations $v = \beta v_1 + (1 - \beta)v_2$ and $a(t) = \beta a_1(t) + (1 - \beta)a_2(t)$ for $\beta \in [0, 1]$, travels exactly the same distance by linearity of the integrals. Since π^* was created via convex combinations, we also know that it satisfies the velocity and acceleration constraints, since its velocity and acceleration always lie between the original velocities and accelerations at any time t in $[t_i, t_j]$. Hence, π^* is a witness that implies $C(p_i, p_j \mid \mathrm{v}_j = v_1 + (1 - \beta)v_2)$ for any $\beta \in [0, 1]$. □

Lemma 2.3.3 below shows how to propagate a single speed interval from p_i to p_j in constant time. The problem is clearly computable, and has $O(1)$ input complexity: two measurements and a single interval of velocities. As such, the lemma readily follows. The complete proof of Lemma 2.3.3, including a derivation of the precise

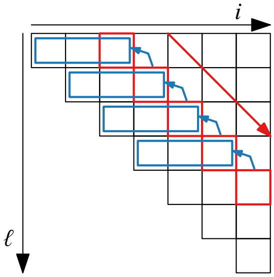


Figure 2.5 The order for computing $\mathcal{I}(\ell, i)$ (red arrow and boxes). Blue boxes and arrows denote dependencies of the cells.

propagation function, requires some lengthy mathematical derivations, which we relegate to Section 2.6 as to not break the flow of the chapter.

2.3.3 Lemma. *Let p_i and p_j be two measurements with $i < j$, and let I be an interval of velocities at p_i. The interval $I' = \{v \mid C(p_i, p_j \mid \mathrm{v}_i \in I \wedge \mathrm{v}_j = v)\}$ of achievable velocities can be computed in $O(1)$ time.*

Let now $\delta(\ell, i)$ denote the number of intervals in $\mathcal{I}(\ell, i)$. We refer to $\delta(\ell, i)$ as the *fragmentation* of $\mathcal{I}(\ell, i)$. Let $\delta_{\max}$ be the maximum fragmentation over all ℓ and i. Using the recurrence for $\mathcal{I}$ defined earlier, we can compute all values $\mathcal{I}(\ell, i)$ using dynamic programming. We compute the $\mathcal{I}(\ell, i)$ values by increasing distance k' from the diagonal, and stop once there are no more reachable speeds. That is, we start by computing all $\mathcal{I}(i, i)$, for increasing i. Observe that these values correspond to having $k' = 0$ outliers. Once we have all sets $\mathcal{I}(i - k', i)$ for some k', we continue with the $\mathcal{I}(i - (k' + 1), i)$ sets (see Fig. 2.5). Let k be the number of outliers in a maximum-length consistent subtrajectory, then all sets of speed intervals $\mathcal{I}(i - (k + 1), i)$ will be empty. Hence, the algorithm finishes after at most $k + 1$ “rounds”. To compute a single entry $\mathcal{I}(i - k', i)$ we have to propagate the speed intervals from at most k other entries (since all sets $\mathcal{I}(\ell, i)$ with $\ell > i$ are also empty). It follows that in total, this procedure takes $O(nk^2 \cdot P)$ time, where P is the time required to propagate all speed intervals in some set $\mathcal{I}(\ell', i)$ to $\mathcal{I}(\ell, j)$. Every set $\mathcal{I}(\ell, i)$ contains at most $\delta_{\max}$ intervals, which we keep in sorted order. Propagating a single interval takes constant time (see Lemma 2.3.3), and merging it with the intervals already in $\mathcal{I}(\ell, i)$ then takes $O(\log \delta_{\max})$ time.

2.3.4 Theorem. *Let T be a 1D trajectory with n measurements. Under the acceleration-bounded model, the maximum length of a physically consistent subtrajectory of T can* *be computed in* $O(nk^2\delta_{\max}\log\delta_{\max})$ *time using* $O(nk\delta_{\max})$ *space, where* k *denotes the number of outliers and* $\delta_{\max}$ *the maximum fragmentation.*

▸ 2.3.2 Retrieving the physically consistent subtrajectory

The dynamic program computes the length ℓ^* of a maximum consistent subsequence. Generally, keeping track of the choices made in a dynamic program allows easy recovery of the actual answer, that is, the actual subsequence. However, we need slightly more, as we join overlapping intervals and thus no longer store which previous measurements led to parts of that interval – generally there may not be only one measurement for an interval.

We could opt for storing a minimum cover of the interval in a cell instead, which we can easily obtain while computing the union. However, this increases memory requirements. Alternatively, we can also use "backpropagation". That is, we extract S itself using the speed intervals in the sets $\mathcal{I}(\ell^*, i)$. We take an interval $I \in \mathcal{I}(\ell^*, i)$ and use an inverse propagation to find a measurement p_h such that $\mathcal{I}(\ell^* - 1, h)$ has a nonempty interval of speeds at which the interval of $\mathcal{I}(\ell^*, i)$ is reachable. We repeat this backpropagation, until the start of the subsequence is reached.

To do this efficiently, we leverage that the intervals in $\mathcal{I}(\ell^* - 1, h)$ are sorted by the dynamic program already. Thus, we use backpropagation in $O(1)$ time by Lemma 2.3.3 to find the velocity interval I' at p_h. We then find whether one of the intervals in $\mathcal{I}(\ell^* - 1, h)$ intersects I' using binary search in $O(\log\delta_{\max})$ time. Thus, computing the subtrajectory after the dynamic program takes $O((n-k)\log\delta_{\max})$ time.

▸ 2.3.3 Bounding the maximum fragmentation

The running time of the dynamic program described in Section 2.3.1 depends on the maximum fragmentation $\delta_{\max}$, that is, the maximum number of intervals in any set of velocities $\mathcal{I}(\ell, i)$. Recall that $\mathcal{I}(\ell, i)$ may contain more than one velocity interval (see Fig. 2.1 (right)). We argue in the following lemma that the fragmentation of a linear number of sets $\mathcal{I}(\ell, i)$ may even be $\Omega(n)$.

2.3.5 **Lemma.** *There is a 1D trajectory* T *with* n *measurements such that* $\Omega(n)$ *sets of velocities* $\mathcal{I}(\ell, i)$ *have fragmentation* $\Omega(n)$.

Proof. We construct a trajectory $T = \langle p_1, \dots, p_{n/2}, q_1, \dots, q_{n/2}\rangle$ such that for parameters $\mathrm{v}^- = 0$ and $a = a^+ = -a^- = 1$, we get $\Omega(n)$ speed intervals at each point p_j, $j > n/2$.

Let $\Delta > 0$ be some real number. We place the points at $p_i = (-4i \cdot \Delta^2, 0)$, for $i \in \{1, \dots, n/2\}$, and $q_j = (0, \Delta)$, for $j \in \{1, \dots, n/2\}$ (see Fig. 2.6). We can ensure that all points have unique time stamps by offsetting them by some arbitrarily small time. This construction ensures that a consistent subtrajectory cannot use two points p_i and p_j simultaneously. We now claim that every point p_i together with point $q = q_1$ generates a consistent subtrajectory $\langle p_i, q \rangle$ for which the possible speeds at q are given by the interval $I_i = [v_i - \Delta, v_i + \Delta]$ with $v_i = 4i\Delta$. Observe that these intervals are all pairwise disjoint. Since the other points q_j are arbitrarily close to q, the same argument shows that we get $\Omega(n)$ speed intervals at those points.

Since $a = 1$ and the time between p_i and q is short, the velocity that the entity has at p_i must be similar to its velocity at q. If the speed at p_i differs too much from the velocity at q, then the entity cannot actually reach q: it will either travel too little or too far. Next, we formalize this argument.

To derive a contradiction, assume that there is a consistent subtrajectory in which an entity travels from p_j, with $j \neq i$, to q and arrives at q with speed $v \in I_i$. Since $\mathrm{v}^- = 0$, the distance that any entity can and has to travel to go from p_j to q is exactly $4j\Delta^2$. The entity covers this in Δ time, and hence its average speed must be $4j\Delta$. Since $a = 1$, it then follows that at any time in the time interval $[0, \Delta]$ its speed lies in the range $[4j\Delta - \Delta, 4j\Delta + \Delta]$.

The entity achieves speed $v \in I_i = [v_i - \Delta, v_i + \Delta]$ at q. So, we have $4j\Delta - \Delta \leq v \leq v_i + \Delta$. Using that $v_i = 4i\Delta$ we get $j \leq i + \frac{1}{2}$. As i and j are natural numbers, we get $j \leq i$. Symmetrically, we have $v_i - \Delta \leq v \leq 4j\Delta + \Delta$, and get $i \leq j$. Combining these results gives $i = j$: a contradiction.

Note that in this construction all consistent subtrajectories have length two. We can easily achieve length $\ell > 2$ by prefixing the construction with a common trajectory of length $\ell - 2$; this prefix provides sufficient time between its last point and the points p_i, to allow the entity to achieve all speeds v_i at p_i. □

It is relatively easy to see that the fragmentation $\delta(\ell, i)$ is at most $O(2^i)$, since any

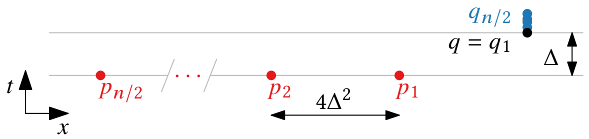


Figure 2.6 Instance with $\Omega(n)$ disjoint speed intervals at c.

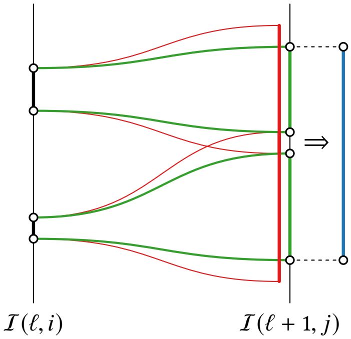


Figure 2.7 Propagation using the slacked model, from p_i to p_j. Green indicates standard propagation and red slacked propagation. The result of merging the intervals is indicated in blue.

fixed subsequence of $\langle \ldots, p_i \rangle$ yields only a single interval (refer to Lemma 2.3.2). To realize such a large number of intervals, they have to be packed ever more closely to the minimum or maximum allowed speed threshold. It seems unlikely that this behavior will appear in realistic settings, and hence we expect that the fragmentation is much smaller in practice. Below, we hence describe an acceleration-bounded model which introduces some slack in the parameters a^- and a^+, which models real-world imprecision.

An acceleration model with slack In a real-world setting we can assume that there is some error in the parameters a^- and a^+ which bound the acceleration. To model this error we introduce a slack parameter $\varepsilon > 0$ for the acceleration bounds. Specifically, let $\Delta a = a^+ - a^-$ denote the difference between minimum and maximum acceleration. For our slacked bounds we add $\frac{\varepsilon}{2}\Delta a$ to a^+ and $-\frac{\varepsilon}{2}\Delta a$ to a^-. During the dynamic program, we first propagate intervals as usual, using the actual bounds a^- and a^+. Then we also propagate using the slacked bounds $a^+ + \frac{\varepsilon}{2}\Delta a$ and $a^- - \frac{\varepsilon}{2}\Delta a$. This is illustrated in Fig. 2.7: the green intervals are the result of standard propagation and the red intervals are the result of slacked propagation. If two slacked intervals intersect, then we merge the corresponding standard intervals and use the merged interval for future propagation (in Fig. 2.7 the blue interval is the result of the merge).

In the following we give an upper bound on the size of any set of intervals $\mathcal{I}(\ell, i)$ as a function of ε. To do so, we estimate the number of disjoint intervals that can occur after propagation. First of all, note that at both the minimum and the maximum velocity, standard intervals can degenerate to a point. Slacked intervals, however, always have non-zero size. Consider now an interval $[\mathrm{v}, \mathrm{w}]$ which slack-propagates to a slacked interval $[\mathrm{v}_s, \mathrm{w}_s]$ of minimum size. This implies that in fact $\mathrm{v} = \mathrm{w}$, that is, the input interval degenerates to a point. We want to determine the minimum separation between v and any other input interval whose slacked propagation touches $[\mathrm{v}_s, \mathrm{w}_s]$. To this end, we compute the largest input velocity v_{hi} which slack-propagates to v_s. The separation is now given by $|\mathrm{v} - \mathrm{v}_{hi}|$. We can now compute a coarse upper bound for the number of intervals by dividing the complete input range $\Delta a \Delta t$ (see Section 2.6) by the separation:

$$\delta_{\text{prop}} = \frac{\Delta a \Delta t}{|\mathrm{v} - \mathrm{v}_{hi}|}$$

Solving for δ_{prop} results in

$$\delta_{\text{prop}} = \left\lfloor \frac{\sqrt{2}}{2\varepsilon\sqrt{\frac{1}{\varepsilon} + 1}\left(\sqrt{2\sqrt{2}\sqrt{\frac{1}{\varepsilon} + 1} - 1} - 1\right)} \right\rfloor = O(\varepsilon^{-1/4}),$$

which proves Theorem 2.3.6 below.

2.3.6 **Theorem.** *Let T be a 1D trajectory with n measurements. Under the slacked acceleration-bounded model, the maximum fragmentation δ_{max} for any set of velocities $\mathcal{I}(\ell, i)$ is $O(n\varepsilon^{-1/4})$.*

▸ 2.3.4 Extending to higher dimensions

The algorithm described above works for one-dimensional data. This may be realistic in some scenarios: for example, if we track contestants in a race along a predefined route, the known route defines an approximately one-dimensional space. However, in most cases, movement is in two or even three dimensions. There are various ways of generalizing the acceleration-bounded model.

There are two standard 2D "interpretations" of our algorithm: either we use the Euclidean distance between the points, or we consider the Euclidean length of the path through all intermediate measurements. In our view, the former is more suitable as we aim to remove outliers which could greatly affect distances in the latter.

Yet, assuming a linear motion between two measurements is unrealistic as well. Thus, we use the Euclidean distance between measurements only as a lower bound;

the upper bound is the Euclidean distance multiplied by a constant λ. Note that an upper bound can also be derived from the current speed and acceleration bounds, but we use our simpler model in the experiments below. To propagate a velocity interval, we use the distance lower bound to determine the minimum velocity at the next measurement, and the upper bound for the maximum velocity.

Of course, the models above assume that the tracked object may turn arbitrarily fast. Effectively, this means that positive or negative velocity becomes meaningless as we can instantaneously rotate from one to the other. We thus set the minimal velocity to zero. However, the direction of movement cannot be changed arbitrarily fast in reality, especially at higher speeds. Though we can easily define various physics models to address this issue, this would require more complex algorithms: we need to know more than just speed for the propagation and thus must generalize from intervals to higher-dimensional regions.

▸ 2.4 Experiments

We introduced various algorithms for computing maximum consistent subsequences of a trajectory, according to different physics models, specifically a speed-bounded and an acceleration-bounded model. The algorithms for the former are simpler and faster than for the latter. However, the acceleration-bounded model is more accurate. Through a series of experiments, we investigate the quality of our algorithms and the trade-off between them.

▸ 2.4.1 Algorithms

We use the following seven algorithms in our experiments. The first two refer to our optimal output-senstive algorithms described above, their running time depending on the number of outliers. Additionally, we use three comparison algorithms to investigate the quality of our methods with respect to simpler algorithms. These algorithms are two variants of an incremental greedy algorithm (under both physics models) and a local greedy method (under the speed-bounded model). We implemented all algorithms in C++; these implementations are open source and available as part of the MoveTK library[3].

[OSB] Optimal Speed-Bounded. This algorithm implements the method of Section 2.1, under the speed-bounded model.

---

[3] `https://movetk.win.tue.nl/`

[OAB] Optimal Acceleration-Bounded. This algorithm implements the method of Section 2.3. We use the 2D generalization, using $\lambda = 1.5$: the upper bound on the traveled distance is 1.5 times the Euclidean distance between two measurements.

[GSB/GAB] Greedy Speed/Acceleration-Bounded. We greedily build a consistent subsequence by testing whether the next considered measurement is consistent with the last measurement in the current subsequence under the speed-bounded model (GSB) or acceleration-bounded model (GAB). For GAB, we use the propagation technique of OAB to maintain an interval of speeds – the next measurement is consistent if the interval after propagation is nonempty. These methods run in $O(n)$ time.

[SGSB/SGAB] Smart Greedy Speed/Acceleration-Bounded. We keep track of multiple subsequences simultaneously. We append the next measurement to each subsequence ending in a consistent measurement; if no such subsequence exists, the measurement starts a new subsequence. The longest subsequence is returned. These methods run in $O(n^2)$ time.

[LGSB] Local Greedy Speed-Bounded. Zheng [139] points to the only other method that uses a speed bound for outlier detection. However, neither survey nor the references therein give a detailed description of this heuristic method. We hence compare against our interpretation of the sketch provided by Zheng [139]. We construct a graph with a vertex per measurement. Two vertices are connected if their measurements are successive in the original trajectory and they are consistent according to the speed bound. A measurement is added to the output, if and only if its vertex is in a connected component of a user-specified size; we set this value to 3 in our experiments. Note that this local heuristic does not guarantee that the complete output is consistent according to the speed bound. This method runs in $O(n)$ time.

▸ 2.4.2 Datasets

We use three real-world datasets in our experiments. They are based on GPS measurements in different modes of transport, at different locations and different times.

[MB] Mountain-bike trips. This dataset consists of 1 214 trajectories of mountain-bike trips of a single cyclist in several European countries from 2012 to 2019.

[HR] The Hague-Rotterdam. This dataset provided by HERE[4] consists of 5 000

---

[4] `https://www.here.com/`

Table 2.1 Summary of the complexities and speeds of trajectories per dataset. The final columns list the default model parameters used throughout the experiment. Speeds are in km/h, accelerations in m/s^2

	trajectories	complexity			speed		model parameters		
		mean	max.	stddev	mean	stddev	v^+	a^-	a^+
MB	1 214	3 377.1	22 426	2 643.4	18.8	10.9	35.0	−3.24	1.62
HR	5 000	424.9	8 925	545.6	62.3	43.0	125.0	−10.00	10.00
LA	78 658	304.4	38 719	1 082.0	55.8	1 557.7	129.6	−10.00	10.00

trajectories of cars and trucks in the region of The Hague-Rotterdam (the Netherlands), on a single day in January 2019.

[LA] Los Angeles. This dataset provided by HERE[4] consists of 78 658 trajectories of cars and trucks in the metropolitan area of Los Angeles, CA (USA), on a single day in September 2018.

All trajectories in the datasets have at least 10 measurements. General statistics of these datasets are provided in Table 2.1, along with our parameter settings per dataset, which are based on the nature and location of the general dataset; note that v^- is always set to 0 to allow the tracked object to remain stationary.

2.4.3 Comparing algorithms and models

In our analysis of the results, we look primarily at relative lengths, that is, the ratio of the number of measurements with respect to the input size. Thus, a result that filters k outliers and keeps $n-k$ measurements has a relative length of $\frac{n-k}{n} \in [0, 1]$. In the remainder, we simply use length to refer to relative length. We start, however, with a brief consideration of efficiency.

Efficiency Table 2.2 provides performance statistics per algorithm and dataset in terms of running time, as performed on a HP Elitedesk 800 g2 TWR (Intel Core i5-6500 CPU at 3.20GHz; 16 GB of RAM; 64-bit Windows 10 Enterprise). Overall, the trend between the algorithms per dataset is roughly the same. We see differences between datasets – specifically MB with respect to LA and HR – which are simply a result of the increased trajectory complexity within the MB dataset. We see that our OSB is competitive with GSB and even considerably faster than SGSB. As we may expect from the theoretical analysis, OAB is very slow in comparison to the other algorithms, yet the greedy alternatives are comparatively fast.

The main questions to investigate are thus two-fold: (1) is the speed-bounded model able to achieve reasonable results, compared to the acceleration-bounded model? (2) how do the faster greedy approaches compare in terms of quality with respect to the optimal algorithms? We first investigate the latter before turning to the former.

Speed-bounded model We have three algorithms that strictly adhere to the speed-bounded model: OSB, GSB and SGSB (see left two columns of Fig. 2.8). As OSB computes optimal results, GSB and SGSB cannot result in longer subsequences. For the MB dataset, we observe that GSB and SGSB perform very similarly in terms of the number of outliers detected. For the HR and LA datasets we see larger differences, especially for GSB. Table 2.3 shows the ratio between OSB and GSB/SGSB according to different brackets of OSB. These numbers indicate that a vast majority of trajectories has less than 10% outliers, and that in such cases the results are on average not much different. The more outliers are present, the more pronounced the difference between our optimal result and the greedy results becomes.

OSB is thus more reliable, as it gives optimal results. When there are few outliers, this algorithm is close to linear and thus we may expect less of a performance loss compared to the simpler methods. Indeed, we see that in terms of running time, OSB (0.48 ms on average per trajectory) performs similarly as the GSB (0.24 ms) and is actually faster than SGSB (5.35 ms). When there are many outliers, the extra time spent may be well worth the effort to obtain the maximum consistent subsequence.

Acceleration-bounded model Referring to Fig. 2.8 and Table 2.3, we observe the same patterns between OAB and GAB/SGAB as above for the speed-bounded variants, but the differences are more pronounced. However, it must be noted that the computation times behave much differently. Although the number of intervals in a single cell never exceeds 2 for almost all trajectories (with a maximum of 4), the computation time of OAB (224.8 ms on average per trajectory) is significantly higher than GAB (0.41 ms) and SGSB (2.86 ms). Thus, OAB seems practical mostly for cases where processing speed is not a primary concern: for example, because much longer offline computations are expected afterwards, or because the trajectory lengths are limited.

Local strategy The LGSB method can also be compared to OSB. However, because this method does not ensure that the entire subsequence adheres to the physics model, it may be the case that LGSB yields a longer sequence than OSB. This is quite structurally the case (see third column in Fig. 2.8), with more pronounced effects for a large number of outliers (see Table 2.3, LGSB row). This indicates that the

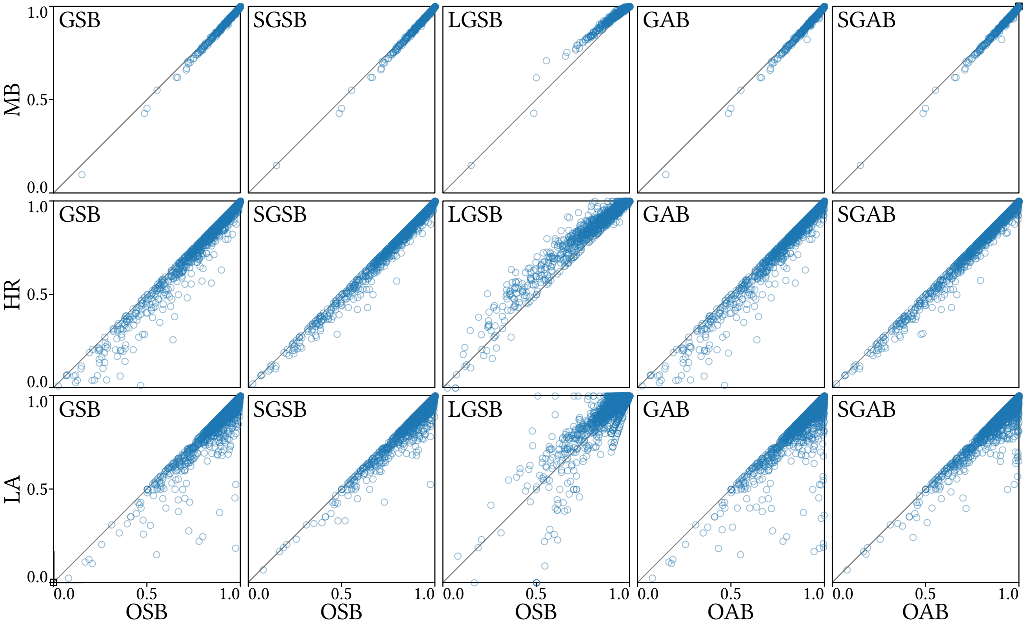


Figure 2.8 Comparing the various algorithms. Each axis represents the (relative) length. Top row: MB data; middle row: HR data; bottom row: LA data. First three columns: comparison of OSB with GSB, SGSB and LGSB; last two columns: comparison of OAB with GAB and SGAB.

Table 2.2 Mean, 99 percentile, and maximum running time in milliseconds (ms), unless indicated otherwise. Running times are shown per dataset, and over all datasets. Note that the imbalance in dataset size skews the mean over all datasets strongly to the mean of the LA dataset.

	MB			HR			LA			All datasets		
	mean	99%	max	mean	99%	max	mean	99%	max	mean	99%	max
OSB	5.32	31.25	203.12	1.01	15.62	62.50	0.37	15.62	359.38	0.48	15.62	359.38
GSB	2.37	15.62	15.62	0.32	15.62	15.62	0.20	15.62	31.25	0.24	15.62	31.25
SGSB	210.26	1 812.50	7 750.00	4.15	78.12	468.75	2.27	15.62	8 593.75	5.35	78.12	8 593.75
LGSB	2.59	15.62	31.25	0.30	15.62	15.62	0.21	15.62	31.25	0.25	15.62	31.25
OAB	7 194.19	89.1**s**	1 074.6**s**	71.03	1 640.62	14.7**s**	127.04	171.88	1 754.6**s**	224.83	1 156.25	1 754.6**s**
GAB	4.22	15.62	31.25	0.45	15.62	15.62	0.35	15.62	46.88	0.41	15.62	46.88
SGAB	95.37	921.88	2 656.25	2.35	31.25	234.38	1.47	15.62	4 843.75	2.86	46.88	4 843.75

Table 2.3 Mean and standard deviation of the ratio between greedy strategies and optimal strategies, split by bins of the optimal length ("length" row). The "size" row indicates the percentage of trajectories in the corresponding length bin. GSB, SGSB and LGSB are compared to OSB; GAB and SGAB to OAB.

	MB				HR				LA			
Length	0.0 - 0.6	0.6 - 0.8	0.8 - 0.9	0.9 - 1.0	0.0 - 0.6	0.6 - 0.8	0.8 - 0.9	0.9 - 1.0	0.0 - 0.6	0.6 - 0.8	0.8 - 0.9	0.9 - 1.0
Size	0.07%	0.20%	0.57%	99.17%	3.14%	4.58%	5.96%	86.32%	0.33%	2.06%	7.00%	90.61%
GSB	0.87 ± 0.14	0.97 ± 0.01	0.98 ± 0.01	1.00 ± 0.01	0.83 ± 0.20	0.95 ± 0.07	0.98 ± 0.03	1.00 ± 0.01	0.87 ± 0.17	0.93 ± 0.11	0.97 ± 0.05	1.00 ± 0.01
SGSB	0.95 ± 0.06	0.97 ± 0.01	0.98 ± 0.01	1.00 ± 0.01	0.93 ± 0.07	0.97 ± 0.04	0.99 ± 0.02	1.00 ± 0.01	0.94 ± 0.08	0.96 ± 0.05	0.98 ± 0.03	1.00 ± 0.01
LGSB	1.10 ± 0.20	1.08 ± 0.02	1.05 ± 0.01	1.01 ± 0.01	1.22 ± 0.28	1.11 ± 0.07	1.05 ± 0.03	1.00 ± 0.01	1.05 ± 0.48	1.04 ± 0.16	1.05 ± 0.05	1.00 ± 0.01
Size	0.07%	0.21%	0.70%	99.02%	3.14%	4.64%	5.94%	86.32%	0.33%	2.06%	7.33%	90.28%
GAB	0.98 ± 0.06	0.97 ± 0.01	0.98 ± 0.01	1.00 ± 0.01	0.83 ± 0.21	0.95 ± 0.07	0.98 ± 0.03	1.00 ± 0.01	0.87 ± 0.17	0.93 ± 0.11	0.97 ± 0.04	1.00 ± 0.01
SGAB	1.10 ± 0.27	0.97 ± 0.01	0.98 ± 0.01	1.00 ± 0.01	0.93 ± 0.08	0.97 ± 0.04	0.98 ± 0.02	1.00 ± 0.01	0.93 ± 0.09	0.96 ± 0.06	0.98 ± 0.03	1.00 ± 0.01

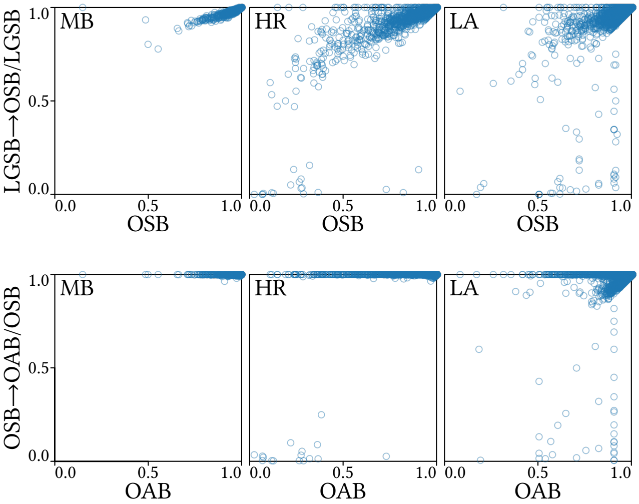


Figure 2.9 Postprocessing to ensure a stricter physics model. Left column: MB data; middle column: HR data; right column: LA data. Top row: comparison of LGSB⟶OSB with OSB; bottom row: comparison of OSB⟶OAB with OAB.

local strategy for determining outliers is not quite suitable for capturing the actual constraints of the physics model.

We further investigate by postprocessing the results of LGSB by OSB (LGSB⟶OSB). That is, we find the longest consistent subsequence of the LGSB result. If LGSB would work perfectly, no outliers are filtered in this postprocessing step. The more outliers are found in the LGSB result, the more violations of the physics model the LGSB result exhibits. The top row of Fig. 2.9 shows the results; note that the vertical axis shows (relative) length of the final result with respect to the length without postprocessing rather than (relative) length with respect to the input. We see again that the results depend on the number of outliers in the trajectory, but overall the difference may be quite pronounced: LGSB⟶OSB on average has 8.75% less measurements than LGSB for cases with OSB length less than 0.9. The dataset also has an effect: MB has less variance than the other two datasets.

Comparing models Since any acceleration-bounded path in our setting is also a speed-bounded path, OSB cannot detect more outliers than OAB. That is, OSB results can be interpreted in the acceleration-bounded model and we can investigate how well the model inherently meets the acceleration bound. We follow the same approach in comparing LGSB to OSB above, postprocessing OSB results by OAB (OSB⟶OAB) to determine how many outliers the OSB result still includes according to the stricter model.

The bottom row of Fig. 2.9 shows the results. We clearly see that that only few measurements are filtered in the OAB postprocessing step for all three datasets. This pattern is strongest in MB (0.09% classified as outliers on average) and HR (0.04% on average), even for more noisy trajectories. For the LA dataset, slightly more measurements are filtered (1.74% on average), but interestingly, this seems mostly the case for the less noisy trajectories. These averages are based on the cases with OAB length less than 0.9.

We may conclude that generally the speed-bounded model is capable of getting quite realistic results even for the acceleration-bounded case, while avoiding the computational complexity. It is interesting that there seems to be slightly different behaviors between the two vehicle datasets: this raises the question whether differences in traffic and driving behavior make acceleration more important in certain environments than in others.

2.4.4 Sensitivity of model parameters

The physics models have a few parameters to capture what is considered feasible movement through space and time. Here, we look at how sensitive the results are to changing the parameter values. Following our observations from the previous section, we focus on the speed-bounded model which effectively has one parameter: the maximum speed v^+, but we also briefly investigate the effect of the detour factor as well as the acceleration bound in OAB.

Procedure Our analysis for each parameter follows the same procedure: we vary the parameter systematically from very restrictive values to very generous values, running the optimal algorithm for the model under consideration on all data. We then consider how the length of the result varies with this parameter.

To allow for summarizing the results, we operationalize the sensitivity σ for a single trajectory as the maximum of the difference between relative length and the difference between two parameter values, over all (consecutive) pairs of parameter values. We refer to the the two parameters that result in the maximum the *sensitive range* ρ of that trajectory; we use the mean value of the range to compute summary statistics. The unit of σ is thus the inverse of the unit of the parameter, but we generally omit this indication. Intuitively, the sensitivity is the "slope" when plotting the relative length as a function of the parameter, which we refer to as a *profile*. We illustrate these functions for each case using a selection of the trajectories for each dataset, consider summary statistics over all trajectories, and investigate the relation between the sensitivity and the sensitivity range.

Note that our choice of step size in varying the parameter inherently limits the maximum sensitivity that can be obtained to the reciprocal of the step size. For example, steps of 2 km/h in varying the speed bound v^+, limits the sensitivity to 0.5, which would indicate jumping from length 0 to 1 between two values of v^+. In degenerate, constructed inputs this can indeed be realized – in fact, any arbitrarily large sensitivity can achieved in theory. Consider a hypothetical trajectory of n measurements along a straight line, sampled every second, with a distance between consecutive measurements a distance c. The length of the optimal result for any $\mathrm{v}^+ < c$ is then $1/n$, as no pair is consistent. However, the length for $\mathrm{v}^+ \geq c$ is 1. Thus, for $\mathrm{v}^+ = c$ and $\mathrm{v}^+ = c - \varepsilon$ we obtain a sensitivity of $(1 - 1/n)/\varepsilon$. For ε approaching zero, this thus tends to infinity. Thus, we focus on the practical slope of these curves, using some reasonable sampling of the domain of the parameter.

Speed bound We run our OSB algorithm, using a speed bound v^+ from 2 km/h to 70 km/h (MB), from 2 km/h to 160 km/h (HR and LA), in steps of 2 km/h. Fig. 2.10 provides a random sample of the resulting profiles. As we can see, many trajectories follow roughly the same pattern of a few steep increases at different speed bounds. We attribute this to different behavior of the moving entity. For MB, this behavior is fairly consistent, with a high sensitivity around 21.5 km/h (average sensitivity range). For the other data sets, this is less clear, likely reflecting different driving behavior due to local speed limits, which varies between trajectories but also within a single trajectory.

Table 2.4 and Fig. 2.11 show summary statistics of the sensitivity for the three

Table 2.4 Sensitivity σ of the speed bound v^+.

dataset	mean	stddev	min	99%	max
MB	0.093	± 0.049	0.013	0.243	0.368
HR	0.059	± 0.042	0	0.244	0.418
LA	0.047	± 0.033	0	0.180	0.458

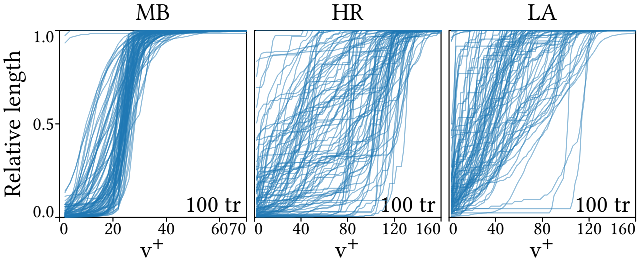


Figure 2.10 Profile of the speed bound: length of OSB as a function of v^+, for 100 random trajectories for each dataset.

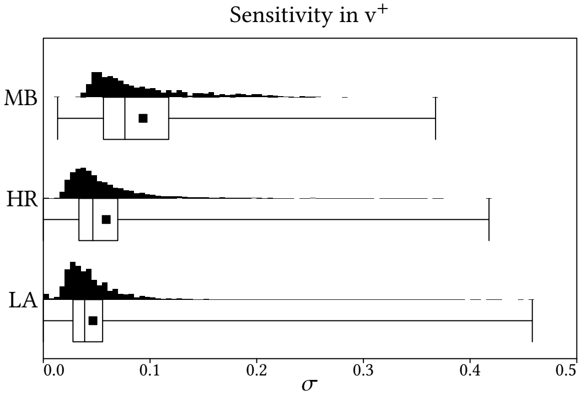


Figure 2.11 Sensitivity σ of the speed bound v^+ per dataset.

datasets. We see that the sensitivity can be quite high in extreme cases: changing the parameter by 1 km/h may change the relative length by almost 0.46. On average, the sensitivity is considerably lower. However, these results still show that careful selection of the model parameters is important: too low values result in measurements being identified as outliers unjustly, but setting them too high might leave too many outliers undetected.

With Fig. 2.12, we look at the relation between the sensitivity and the sensitivity range. We roughly see the same pattern for each of the datasets: a number of trajectories have their relatively large sensitivity at low speeds, followed by another peak at higher speeds. Potentially, this separates the trajectories into different cases of actual behavior: for example, cars drive at different speeds in residential areas, provincial

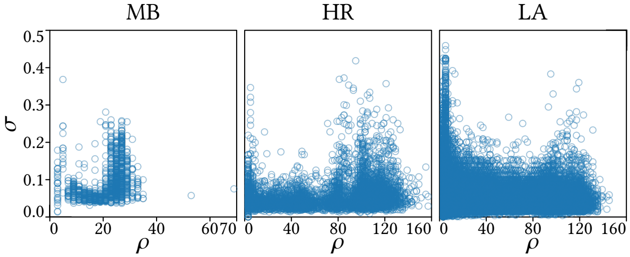


Figure 2.12 Scatterplot relating the sensitivity σ of v^+ to (the mean of) the sensitivity range ρ. Each circle represents a single trajectory.

roads and highways – if a trajectory falls mostly within one of the categories, one may expect the largest sensitivity to occur at that speed. Under this hypothesis, we see that we used quite reasonable bounds on the maximum speed, that is, values slightly higher than the sensitivity range for the majority of the trajectories.

Acceleration bound The acceleration-bounded model has, as the name suggests, parameters controlling the allowed acceleration and deceleration, a^+ and a^- respectively. Due to the high computational cost of OAB, we restrict our attention to only six values combinations of a^+ and a^- per dataset. The parameters we selected for defaults reflect fairly extreme capabilities: limits of racing cars (HR and LA) and estimates of well-trained cyclists (MB). To investigate the effect of these parameters, we thus reduce these parameter values, to reflect settings of "normal" and "slow" behavior in terms of acceleration and deceleration. Specifically, we test the following six combinations for each dataset: $a^+ \in \{2, 4, 10\}$ and $a^- \in \{-2, -10\}$ (HR and LA); $a^+ \in \{0.8, 1.2, 1.62\}$ and $a^- \in \{-2, -3.24\}$ (MB). These values are expressed in $\mathrm{m/s}^2$.

We can now study sensitivity of the one parameter by fixing the other parameter to each of its values. A sample of the resulting profiles are shown in Fig. 2.13 and Fig. 2.14. We immediately see that there is very little effect of the acceleration or deceleration bound. As these are not a random sample, but actually the profiles with highest sensitivity, this tells us that these parameters are of little influence.

The summary statistics over all trajectories further confirm this, as shown in Fig. 2.15. Considering our chosen set of parameters, the sensitivity in a^+ can be at most 0.5 (HR and LA) or 1.67 (MB); for a^- these maxima are 0.125 (HR and LA) and 0.81 (MB). What we observe, however, is that the actual sensitivity is significantly lower – also foregoing the need for further refining the tested parameter values. The strongest

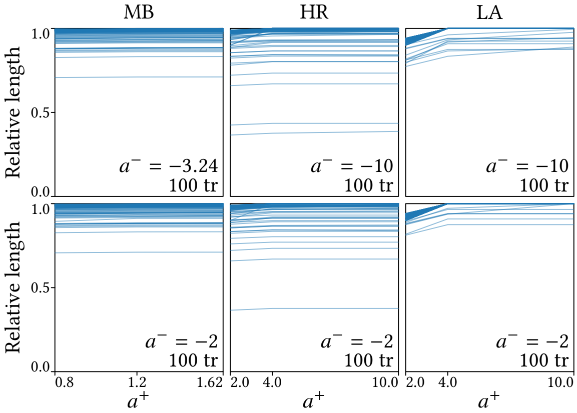


Figure 2.13 Profiles of the acceleration bound: length of OAB as a function of a^+, for the 100 most sensitive trajectories for each dataset.

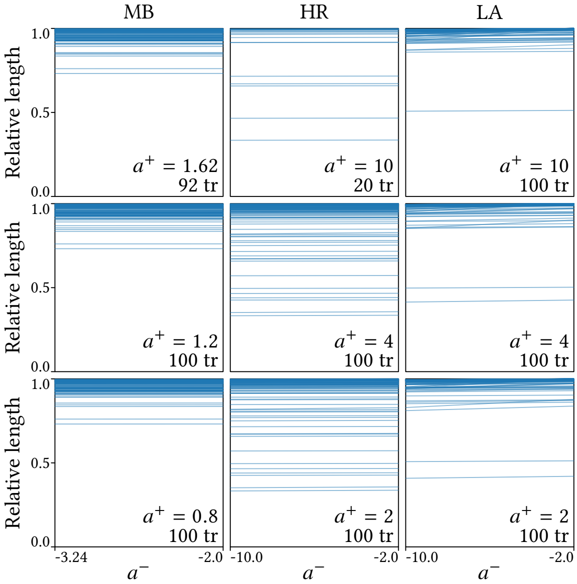


Figure 2.14 Profiles of the deceleration bound: length of OAB as a function of a^-, for the 100 most sensitive trajectories for each dataset.

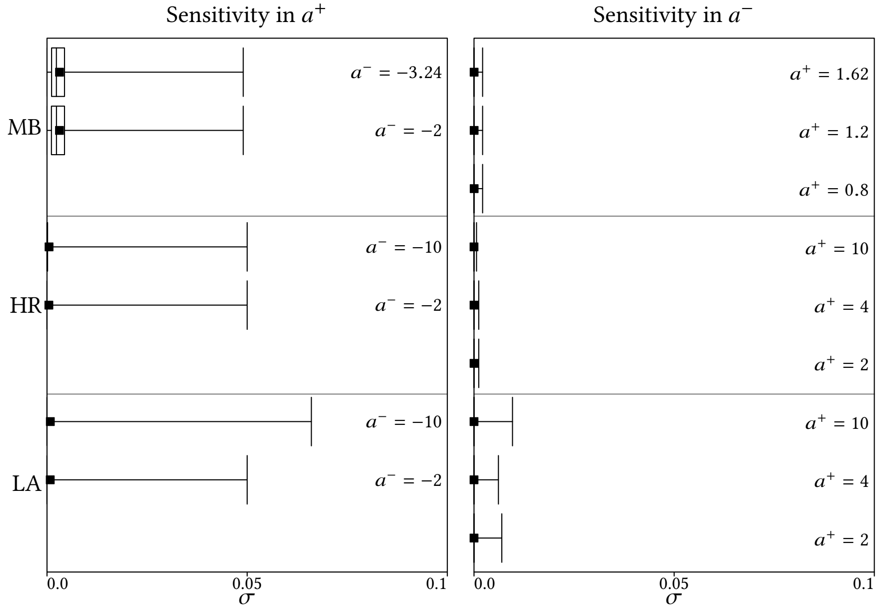


Figure 2.15 Sensitivity of a^+ (left) and a^- (right) per dataset and value of the other parameter. Note that the technical maximum sensitivity is much higher, but this does not occur, hence the horizontal scale.

sensitivity observed is 0.066 for a^+ and 0.01 for a^-. This is, however, an "extreme" with medians and averages laying much closer to zero.

One observation to be made is that the sensitivity for the maximum acceleration in the MB seems to be slightly higher, though still much lower. This is possibly caused by the nature of the data: a mountain biker may accelerate and decelerate more strongly, compared to regular traffic.

In Fig. 2.16, we show histograms for the sensitivity, split by the sensitivity range. In these charts, we omit all trajectories which have sensitivity zero – the number of remaining trajectories is indicated per dataset. Notably, we see that MB has relatively few trajectories with zero sensitivity, whereas for the other datasets this is the majority of trajectories. Again, we attribute this to the different nature of MB.

In light of the little dependence on the acceleration bounds, we assume that the speed bounds used by the acceleration model, in terms of sensitivity, behave similarly as

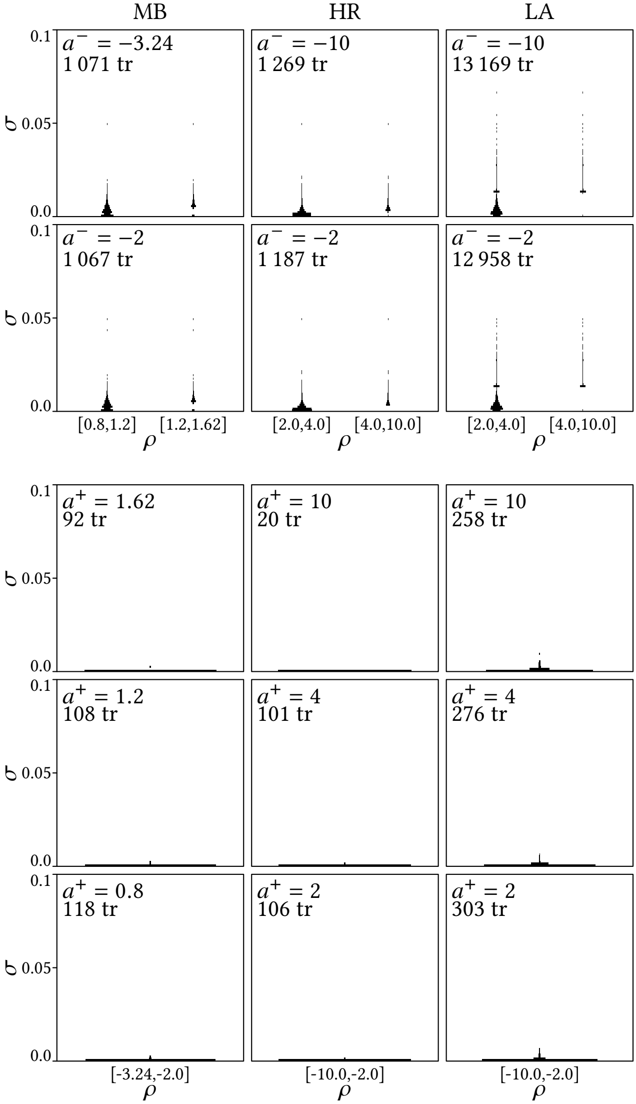


Figure 2.16 Histogram relating sensitivity of a^- (top two rows) and a^+ (bottom three rows) to the sensitivity range. Trajectories with sensitivity 0 have been omitted. Note that the technical maximum sensitivity would be significantly higher, but this does not occur – the horizontal scale has been adjusted.

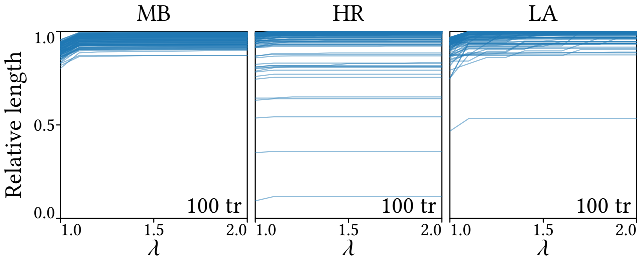


Figure 2.17 Profiles of the detour factor: length of OAB as a function of λ, for the 100 most sensitive trajectories for each dataset.

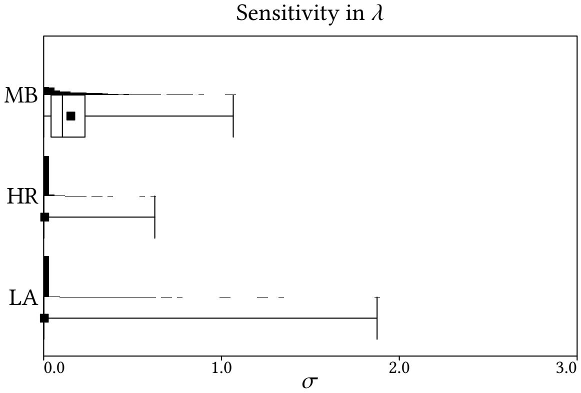


Figure 2.18 Sensitivity of λ per dataset. Note that the technical maximum sensitivity would be 10, but this does not occur – the horizontal scale has been adjusted.

for the speed-bounded model. More importantly, these results further support our conclusion from Section 2.4.3: the speed-bounded model provides realistic results even for the acceleration-bounded model.

Detour factor The OAB algorithm uses a detour factor λ, to determine how much distance can at most be traveled between two measurements, which is λ times the Euclidean distance. In other experiments, this is fixed to 1.5, but here we investigate how much this parameter may influence the results. We run the OAB algorithm using λ from 1 to 2, with increments of 0.1.

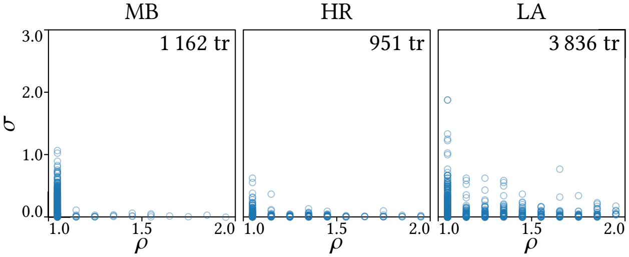


Figure 2.19 Scatterplot relating the sensitivity σ of λ to (the mean of) the sensitivity range ρ. Each circle represents a single trajectory. Trajectories with sensitivity 0 have been omitted. Note that the technical maximum sensitivity would be 10, but this does not occur – the vertical scale has been adjusted.

Refer to Figures 2.17 and 2.18. We observe that detour factor λ has very little influence in general, with most trajectories not being influenced by λ at all: 52 out of 1 214 for MB, 4 049 out of 5 000 for HR and 74 500 out of 78 658 for LA. The detour factor is likely to help in cases where turns are made at relatively high speed: the Euclidean distance might be too short to slow down and reach the next measurement at the right time – but adding some slack gives enough space to travel between two somewhat close points at high speed. Thus, this factor can be expected to be of less influence for trajectories with high sampling frequency or without turns are relatively high speed. This may explain why the mountain-bike dataset exhibits more sensitivity than the other two vehicle datasets.

Fig. 2.19 relates the sensitivity to the sensitivity range. We observe that the highest sensitivity is found in the sensitivity range $[1, 1.1]$, and generally a trend of higher sensitivity at lower values of λ. This supports our suggestion above as to the cause of the low sensitivity.

Most of our trajectories have relatively high sampling frequency and as such the sensitivity is low. The question is how these observations generalize to low sampling frequencies. This will likely depend strongly on the object being tracked. If it travels frequently at nearly maximum speed, sensitivity may be high. However, if the general speed is significantly lower, the admitted variation in the reconstructed speed may already be sufficient to avoid sensitivity in the detour factor.

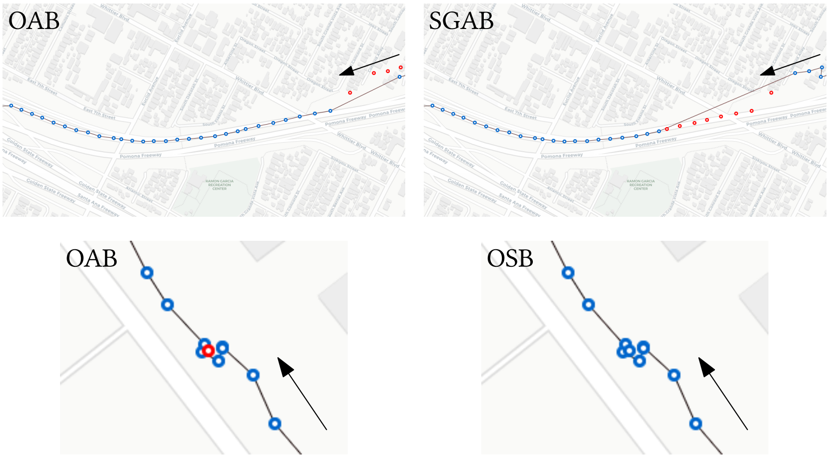


Figure 2.20 Example trajectory where OAB differs from SGAB (top) and OSB (bottom). Arrows indicate the direction of travel, blue markers are measurements that are part of the consistent subsequence computed, and red markers indicate the corresponding outliers. Both trajectories are from the LA dataset. Base maps © OpenStreetMap.

2.5 Discussion

Results Our results indicate that our optimal algorithms outperform simple greedy strategies, either in quality of the results, running time, or both. Noise levels and other characteristics do influence these results, and our methods are particularly effective for dealing with large amounts of noise. The example in Fig. 2.20 (top) illustrates a case where the OAB algorithm computes a longer sequence, compared to SGAB: the cause is that a few erroneous measurements lead this greedy algorithm to make a sequence that prevents it from selecting many measurements later.

Furthermore, the results suggest that the quality difference between speed-bounded models and acceleration-bounded models is small. This must be considered carefully though, as there is an effect of social or geographic environment. Fig. 2.20 (bottom) shows a case where the OSB algorithm detects fewer outliers, though the difference is only minor. Contrasting the previous comparison, this implies that OAB performed better than OSB: OSB fails to capture the outlier that is not physically realizable in the stronger acceleration-bounded model. That is, there is not enough time to realistically decelerate and accelerate to capture all near-stationary measurements.

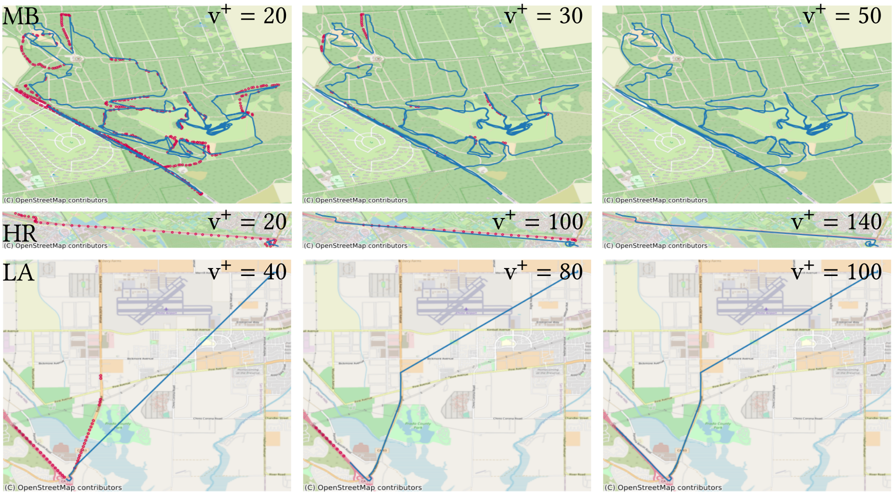


Figure 2.21 Example results, using different speed bounds v^+. One trajectory for each dataset is shown. Blue line represents the resulting trajectory, with red dots marking the outliers. Base maps © OpenStreetMap.

The selection of parameters influences the results, but this is mostly the case for the maximum speed. Acceleration and detour factor for our OAB algorithm tend to have minimal effect on the number of outliers detected, though we observed variation between the types of moving entities. Fig. 2.21 shows a sequence of results for different speed bounds, for a trajectory from each dataset. Increasing the speed bound leads to fewer outliers – but possibly less realistic behavior, if the bound is set too high. The effect of lowering the speed bound is that corners tend to be cut by marking outliers, to lower the traveled distance to one that is achievable within the speed bound.

More context By design, we do not consider the use of other contextual data in this chapter, such as a road network that a vehicle is driving on. As we will discuss in the next chapter, OSB and OAB generalize relatively straightforward when considering consistency on a road-network. For our faster algorithm under the speed-bound model, however, this is not quite the case, as the AWVD is no longer directly applicable, but there may be potential to generalize the approach.

Beyond assessing distances more accurately via road networks, additional data could also be used to define more accurate physics models. Our current models are fairly simple, and use only few parameters to define global thresholds on the maximum speed and acceleration. However, such thresholds may actually depend on the environment. As we show in the next chapter, we can leverage the (expected) maximum speed for driving in a car, which is different on the highway than it is in an urban environment. Similarly, cycling uphill or downhill affects maximum speed. Ideally, physics models and, by extension, outlier-detection algorithms should accommodate for such variations, as this allows for more efficacious processing of heterogeneous trajectories that travel through different environments.

Including additional contextual factors will make the models more accurate and realistic, but a crisp decision boundary (movement is or is not physically possible) may no longer exist. Instead, we may want to define that a car can violate speed limits, but the severity and duration affect how likely the behavior is. Future work could explore “behavioral models” that describe expected movement more closely, including context, and are more robust by allowing deviations from the model, thereby reducing parameter sensitivity.

Enhancing other techniques There are many other forms of trajectory processing and analysis techniques, such as clustering, map matching, and segmentation. Such techniques may be complemented or enhanced by applying physics models to define possible or realistic behavior. For example, a map-matching algorithm could include considerations of whether its result is physically realizable, or clustering may be done based on what physical behavior would be necessary to realize certain trajectories. In the next chapter, we explore how to use physical consistency in a map-matching algorithm, employing local speed limits to derive consistency. We leave exploring such complementarity of other techniques to future work, but our results presented here provide a framework and methods that may be integrated into such enhanced techniques.

▸ 2.6 Derivation of the propagation function

For completeness, we include here the derivation of the propagation functions and the proofs that they indeed follow the shape that we claim.

For our algorithm under the acceleration-bounded model, we need to determine the minimum and maximum speed for which the moving entity can arrive at a measurement p_j, when starting at a measurement p_i with some given velocity. For our asymptotic analysis, we already argued that this is possible in linear time (Lemma 2.3.3).

Here, we show how to precisely compute this interval, providing the exact formula for propagation and thus proving that this is indeed computable. We do so for the minimum velocity; computing the maximum velocity is symmetrical. This minimum velocity $\mathrm{v}_{min} \in \mathbb{R}$ is the smallest value such that $C(p_i, p_j \mid \mathrm{v}_i = \mathrm{v}, \mathrm{v}_j = \mathrm{v}_{min})$ for two measurements p_i, p_j and some initial $\mathrm{v} \in \mathbb{R}$. Note that these velocities may be negative, indicating the direction of movement in the 1D space.

We need particular behaviors of the moving entity to accomplish this minimum velocity. These behaviors are determined by the travel time $\Delta t = t_j - t_i$ and distance $\Delta x = \|p_j - p_i\|$ between p_i and p_j, and the initial velocity v: we want to travel $\Delta x A$ between p_i and p_j within Δt time, while minimizing the velocity at p_j.

Checking consistency First, we determine whether *any* velocity can be obtained, that is, whether it is physically possible to travel distance Δx in Δt time, starting with velocity v. That is, we must test whether $C(p_i, p_j \mid \mathrm{v}_i = \mathrm{v})$ holds. The maximum distance Δx^+ that can be traveled, is obtained by accelerating until reaching the maximum velocity v^+ and then maintaining that speed. Accelerating to maximum speed takes $t^+ = \frac{\mathrm{v}^+ - \mathrm{v}}{a^+}$ time. If $t^+ < \Delta t$, the maximum velocity is achieved and we can express Δx^+ as $(\mathrm{v} + \mathrm{v}^+)t^+/2 + (\Delta t - t^+)v^+$. Otherwise, this behavior accelerates maximally for the entire duration, in which case $\Delta x^+ = (2\mathrm{v} + a^+\Delta t)\Delta t/2$. Analogously, we find an expression for the minimum distance Δx^- that can be traveled. As we can use a convex combination of achieve any traveled distance between these two extremes, we know that $C(p_i, p_j \mid \mathrm{v}_i = \mathrm{v})$ if and only if $\Delta x^- \leq \Delta x \leq \Delta x^+$.

Note that the above test indicates consistency, then we can look for a minimal (and maximal) velocity. If this consistency does not hold, we know that no velocity can be reached at all. For the remainder, we assume that $C(p_i, p_j \mid \mathrm{v}_i = \mathrm{v})$ is indeed true.

Finding the minimum velocity We now strive to find the necessary behavior that results in the minimum velocity, v_{min}, assuming we have already confirmed the basic consistency described above. To visualize this behavior, we look at the velocity-time diagrams for the moving entity; see the examples in Fig. 2.22. The curve in this diagram represents the velocity as a function of time. In this diagram, we need that the total area under the curve is exactly the traveled distance Δx. The speed bounds $\mathrm{v}^-, \mathrm{v}^+$ are now represented as allowed minimum and maximum values for the curve. The bounds on the acceleration, a^-, a^+ translate to the minimum and maximum slope the curve can have at any time.

We can distinguish a number of situations where we need different behavior to get to the minimum velocity, depending on whether we travel at maximal velocity

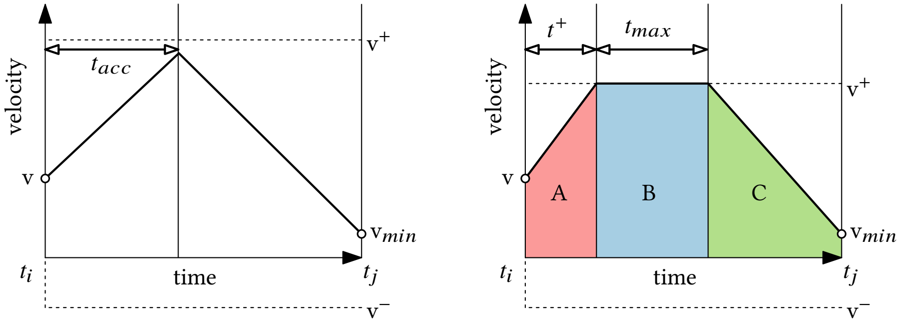


Figure 2.22 (Left) Velocity-time diagram for the behavior where one maximally accelerates for a time t_{acc} and then maximally decelerates to obtain the lowest possible speed v_{min}. (Right) Velocity-time diagram for the behavior of accelerating to the maximum velocity v^+ in time t^+, then maintaining this velocity for t_{max} time, and finally maximally decelerating to get the minimum velocity.

intermediately. If we can reach p_j by first maximally accelerating for some time t_{acc} and then maximally decelerating the rest of the time, this gives us the lowest possible velocity (left diagram in Fig. 2.22). We can, however, encounter the case where the maximum velocity we reach with this behavior exceeds v^+. In this case, we can accomplish the minimum velocity by accelerating to v^+ in time t^+, retaining this speed for some time t_{max} and then maximally decelerating (right diagram in Fig. 2.22) for the remaining time. However, if the result of the appropriate situation above violates the minimum velocity bound v^-, then we can conclude that $v_{min} = v^-$. Detailed proofs that the first two behaviors indeed give the minimum velocity are given in Sections 2.6.1 and 2.6.2.

To compute the minimum velocity for the first and second case, we will use the equation of motion in 1D, given by

$$\Delta x = v\Delta t + \int_{t_i}^{t_j} \int_{t_i}^{t} a(t')\mathrm{d}t'\mathrm{d}t. \tag{2.2}$$

This describes the aforementioned requirement that the area under the curve in the velocity-time diagram is the distance Δx between p_i and p_j. To find the minimum velocity for the first two described situations, we fill in the shape for the acceleration

$a(t)$ and determine the minimum velocity, given by

$$\mathrm{v}_{min} = \mathrm{v} + \int_{t_i}^{t_j} a(t)\mathrm{d}t. \tag{2.3}$$

Maximally accelerate, then maximally decelerate We first consider the situation where we do not reach the velocity bound when maximally accelerating. In this first situation, our acceleration function is equal to

$$a(t) = \begin{cases} a^+, & t_i \le t \le t_i + t_{acc} \\ a^-, & t_i + t_{acc} < t \le t_j \end{cases}. \tag{2.4}$$

What remains is to determine t_{acc}. We do this by solving the 1D equation of motion (Eqn. 2.2) for the distance Δx between the measurements. We fill in the acceleration function and integrate to get

$$\Delta x = \mathrm{v}\Delta t + \frac{1}{2}a^+ t_{acc}^2 + (\mathrm{v} + a^+ t_{acc})(\Delta t - t_{acc}) + \frac{1}{2}a^-(\Delta t - t_{acc})^2. \tag{2.5}$$

We can now solve this quadratic equation for t_{acc}. To simplify notation, we use $\Delta a = a^+ - a^-$ and $\bar{\mathrm{v}} = \Delta x/\Delta t$, that is, the average required velocity to travel the distance in the given amount of time. We pick the root of the solution such that the resulting t_{acc} is in $[t_i, t_j]$ and get

$$t_{acc} = \Delta t - \sqrt{\frac{\Delta t}{\Delta a}}\sqrt{a^+\Delta t + 2(\mathrm{v} - \bar{\mathrm{v}})}. \tag{2.6}$$

With this value, we can now determine the minimum velocity. We fill in Eqn. 2.3 and get

$$\begin{aligned} \mathrm{v}_{min} =& \mathrm{v}(t_i + \Delta t) = \mathrm{v} + t_{acc}a^+ + (\Delta t - t_{acc})a^- \\ =& \mathrm{v} + \Delta t a^+ - \sqrt{\Delta a \Delta t}\sqrt{a^+\Delta t + 2(\mathrm{v} - \bar{\mathrm{v}})} \end{aligned} \tag{2.7}$$

Note that this situation applies only if we do not exceed the velocity bounds when accelerating and decelerating. So we require that

$$\mathrm{v} + a^+ t_{acc} \le \mathrm{v}^+, \qquad \mathrm{v}_{min} = \mathrm{v} + t_{acc}a^+ + (\Delta t - t_{acc})a^- \ge \mathrm{v}^- \tag{2.8}$$

Accelerating to maximum velocity We now consider the situation where we reach the speed bound v^+. We accelerate for some t^+ time, until we are moving with velocity v^+, then we retain that velocity for some time t_{max}, and finally we maximally decelerate to get the minimum velocity. We can describe this behavior with the following acceleration function:

$$a(t) = \begin{cases} a^+, & t_i \le t \le t_i + t^+ \\ 0, & t_i + t^+ < t \le t_i + t^+ + t_{max} \\ a^-, & t_i + t^+ + t_{max} < t \le t_j \end{cases} \tag{2.9}$$

We now need to determine t^+ and t_{max}. As before, $t^+ = \frac{\mathrm{v}^+ - \mathrm{v}}{a^+}$ indicates the time needed to accelerate from v to v^+.

With the equation for t^+, we can now determine t_{max} by again solving the 1D equation of motion in Eqn. 2.2 with our new acceleration function. This gives us the following equation of motion, and solution for t_{max}:

$$\Delta x = \mathrm{v}t^+ + \frac{1}{2}a^+(t^+)^2 + \mathrm{v}^+(\Delta t - t^+) + \frac{1}{2}a^-(\Delta t - t_{max} - t^+)^2 \tag{2.10}$$

$$t_{max} = \Delta t - t^+ - \sqrt{\frac{3a^+}{a^-}(t^+)^2 + \frac{2\Delta t(\bar{\mathrm{v}} - \mathrm{v}^+)}{a^-}} \tag{2.11}$$

Using the above, we now find the minimum velocity, by filling in Eqn 2.3:

$$\mathrm{v}_{min} = \mathrm{v}^+ + (\Delta t - t^+ - t_{max})a^- = \mathrm{v}^+ + \sqrt{\frac{3a^+}{a^-}(t^+)^2 + \frac{2\Delta t(\bar{\mathrm{v}} - \mathrm{v}^+)}{a^-}}\,a^- \tag{2.12}$$

This case is applicable only if $t_{max} > 0$ and the resulting $\mathrm{v}_{min} \ge \mathrm{v}^-$.

Achieving v^- The extreme behavior of the previous two cases achieve the lowest possible speed, without violating v^+, a^+ or a^-. We can readily choose between the two cases, by comparing t_{acc} with t^+: if the former is at most the latter, the first case applies, and otherwise the second. However, the result may still violate the physics model, but only v^-. That is, if the computed v_{min} is below v^-. Our claim is that, in such a case, v_{min} is actually equal to v^-. Intuitively, the previous cases in fact achieve a velocity that is too low: we thus have slack to use less extreme behavior intermittently, such as standing still ($\mathrm{v} = 0$) for a certain time.

Consider the behavior of the previous cases. If we follow the behavior but maintain the minimal velocity bound, we have too much area under the curve: we overshoot our traveled distance. We can compensate for this by accelerating less extremely or maintaining a velocity below v^+. Since some velocity is obtainable, we know that there is sufficient slack and can indeed achieve v^-, if the previous cases would violate the velocity bound of v^-.

2.6.1 Proof: achieving minimum velocity without attaining velocity bounds

We now show that the behavior of maximally accelerating, followed by maximally decelerating indeed gives the minimum velocity, provided that the velocity bounds $\mathrm{v}^-, \mathrm{v}^+$ are never exceeded during this behavior. Without loss of generality, we further assume that $t_i = 0$ to simplify the exposition, and thus $t_j = \Delta t$.

The equation of 1D motion (Eqn. 2.2) must satisfied for $a(t)$, with additional constraints that $a(t) \in [a^-, a^+]$ for any $t \in [0, \Delta t]$. We then want to minimize the velocity at p_j, as given by Eqn. 2.3.

We represent the function $a(t)$ as follows:

$$a(\phi, t) = a^- + (a^+ - a^-)\phi(t) = a^- + \Delta a \psi(t) \tag{2.13}$$

where $\phi : [0, \Delta t] \longrightarrow [0, 1]$. This way, the acceleration bounds are trivially satisfied by the function.

Let $a(\psi, t)$ be the function representing maximal acceleration up to some time $t_{acc} \in [0, \Delta t]$ and then maximum deceleration until t_j. In addition, assume that the traveled distance is satisfied by this function. We can represent $\psi(t)$ by

$$\psi(t) = \begin{cases} 1 & t \le t_{acc} \\ 0 & t > t_{acc} \end{cases}. \tag{2.14}$$

Let $\psi'(t)$ be a function given by $\psi(t) + \delta\psi(t)$ where $\delta\psi(t)$ is a perturbation on the function, such that $a(\psi', t)$ still travels the required distance, but differs in at least one value t from $a(\psi, t)$. We now show that the velocity at p_j for this perturbed function is always greater than the velocity produced by $\psi(t)$.

By assumption, both travel the same Δx distance in Δt time. Thus, filling in Eqn. 2.2 for both gives us the following equality and its simplification:

$$\mathrm{v}\Delta t + \int_0^{\Delta t} \int_0^t (a^- + \Delta a \psi(t'))\mathrm{d}t'\mathrm{d}t = \mathrm{v}\Delta t + \int_0^{\Delta t} \int_0^t (a^- + \Delta a(\psi(t') + \delta\psi(t'))\mathrm{d}t'\mathrm{d}t \tag{2.15}$$

$$\int_0^{\Delta t} \int_0^t \delta\psi(t')\mathrm{d}t'\mathrm{d}t = 0 \tag{2.16}$$

We know that the velocity at p_j is given by Eqn. 2.3. We can now look at the difference

Δv between this velocity for $\psi'(t)$ and for $\psi(t)$. This difference is given by

$$\Delta \mathrm{v} = \Delta a \int_0^{\Delta t} \delta\psi(t) \mathrm{d}t. \tag{2.17}$$

Now, if the minimum velocity at p_j for $a(\psi', t)$ is smaller than the minimum velocity for $a(\psi, t)$, this would imply that Δv is negative for the corresponding $\delta\psi$ function.

By definition, the value of $\delta\psi(t)$ is non-positive for $t < t_{acc}$ and non-negative for $t > t_{acc}$, as otherwise the acceleration bounds would be violated. Eqn. 2.16 implies that $\int_0^{t_{acc}} \delta\psi(t)\mathrm{d}t < 0$ and $\int_{t_{acc}}^{\Delta t} \delta\psi(t)\mathrm{d}t > 0$. Equivalently, the integral $\int_0^{t} \delta\psi(t)\mathrm{d}t$ is non-increasing for the interval $[0, t_{acc}]$ and non-decreasing for the interval $[t_{acc}, \Delta t]$.

Assume for a contradiction that Δv is negative for some $\delta\psi$, that is, $\int_0^{\Delta t} \delta\psi(t)\mathrm{d}t < 0$. This automatically implies that $\int_0^{t} \delta\psi(t')\mathrm{d}t' \leq 0$ for any t in the interval, due to the non-decreasing and non-increasing properties of the integration interval. But then the integral of Eqn. 2.16 is by definition negative, which means that $a(\psi', t)$ does not travel Δx distance. Hence, we must have that $\Delta \mathrm{v} \geq 0$. Observe that $\Delta \mathrm{v} = 0$ only if $\delta\psi(t)$ is zero.

To prove that maximally decelerating and then accelerating results in the maximum velocity follows a similar argumentation.

2.6.2 Proof: achieving minimum velocity when attaining velocity bounds

We now prove that, if we maximally accelerate to the velocity bound v^+ in time t^+, retain this speed for time t_{max}, and then maximally decelerate, this indeed gives us the lowest possible velocity, if the previous situation does not apply – that is, $t^+ \leq t_{acc}$, and we never reach the velocity lower bound v^-. Without loss of generality, we assume that $t_i = 0$ and $t_j = \Delta t$.

We follow the argumentation as described in the previous section. We again describe the acceleration behavior using $a(\psi, t)$ for a to be defined function $\psi(t)$. We assume that $a(\psi, t)$ travels the required distance given the initial velocity. But now, the behavior of this case yields a slightly different function for ψ, do describe $a(\psi, t)$:

$$\psi(t) = \begin{cases} 1, & t \leq t^+ \\ \alpha, & t^+ < t \leq t^+ + t_{max} \\ 0, & t > t^+ + t_{max} \end{cases} \tag{2.18}$$

Here, $\alpha = -\frac{a^-}{\Delta a}$ indicates an acceleration of zero. For brevity, we call the three time regions with the different behaviors A, B and C, see Fig. 2.22.

We again perturb $\psi(t)$ to get a function $\psi'(t) = \psi(t) + \delta\psi(t)$. Here, we again assume that $\delta\psi(t)$ is not the zero function. To obey the acceleration bounds, we observe that $\delta\psi(t)$ is non-positive in region A and non-negative in region C, which implies that $\int_0^t \delta\psi(t)\mathrm{d}t$ is non-increasing in region A and non-decreasing in region C.

For region B, we observe that $\psi(t)$ describes the behavior that results in the highest possible velocity in that region. So, the velocity at a time t in region B that results from $\psi'(t)$ can be at most the velocity obtained via $\psi(t)$. The velocity at any time t is given by

$$\mathrm{v}(\phi, t) = \mathrm{v} + ta^- + \Delta a \int_0^t \phi(t)\mathrm{d}t \tag{2.19}$$

for acceleration function $a(\phi, t)$. Thus, we can now formalize the observation as $\mathrm{v}(\psi', t) \leq \mathrm{v}(\psi, t)$ for all $t \in B$. Using the definition of $\mathrm{v}(\phi, t)$ and simplifying, we obtain that

$$\int_0^t \delta\psi(t) \leq 0 \tag{2.20}$$

should hold for all $t \in B$. Since we want that ψ and ψ' travel the same distance Δx in Δt time, we again get the identity

$$\int_{t_i}^{t_j} \int_{t_i}^{t} \delta\psi(t)\mathrm{d}t'\mathrm{d}t = 0, \tag{2.21}$$

as was shown in Section 2.6.1. Similar to the argumentation in Section 2.6.1, we look at the difference in minimum speed $\Delta\mathrm{v}$ at t_j:

$$\Delta\mathrm{v} = \Delta a \int_{t_i}^{t_j} \delta\psi(t)\mathrm{d}t \tag{2.22}$$

Again, $\psi'(t)$ has a lower minimum velocity if $\Delta\mathrm{v}$ is negative. For this to happen, $\int_{t_i}^{t_j} \delta\psi(t)\mathrm{d}t$ has to be negative.

Now, assume for a contradiction that for some $\delta\psi$, $\Delta\mathrm{v}$ is less than zero, such that the minimum velocity using $a(\psi', t)$ is less than that from $a(\psi, t)$. From Eqn. 2.20, we see that $\int_0^t \delta\psi(t)\mathrm{d}t$ is non-positive for all t in regions A and B. In particular, at the end of region B, the integral is non-positive. We distinguish two cases.

The integral is zero. Suppose the integral $\int_0^t \delta\psi(t)\mathrm{d}t$ is zero at the end of region B. Then, since the integral is non-decreasing in region C as established before, we cannot have that $\int_0^{\Delta t} \delta\psi(t)\mathrm{d}t$ is less than zero: $\Delta \mathrm{v}$ is non-negative, which gives a contradiction.

The integral is negative. Suppose now that $\int_0^{t} \delta\psi(t)\mathrm{d}t$ is negative at the end of region B. If we want $\Delta \mathrm{v}$ to be negative, this requires that the $\int_0^{\Delta t} \delta\psi(t)\mathrm{d}t$ is negative. Since the integral is non-decreasing in region C, we must have that the integral is negative everywhere in region C to accomplish this. But then the traveled distance for $\psi(t)$ and $\psi'(t)$ is not the same, since Eqn. 2.21 is less than zero. This again gives a contradiction.

From the previous argumentation, we can conclude that for any choice of $\psi'(t)$ that satisfies the traveled distance requirement, the minimum velocity at p_j is at least the minimum velocity obtained by using $\psi(t)$.

Chapter 3

Physically Consistent Map-Matching

As we mentioned in the discussion of the previous chapter, we can actually seek to apply physical consistency in spaces that are different from the Euclidean spaces we covered in that chapter. In particular, we can consider the road network as the space that our entity moves in. For human mobility in the form of traffic, the road network defines and constrains vehicle movement, allowing us to infer local information by considering where the vehicle was driving in the network. Combining this with the physical models, we can try to predict what routes an entity could have taken through the network.

Key challenges for using trajectories are that they may contain noise and may even be incomplete or sparsely sampled due to measuring or privacy constraints. Algorithms have been developed in previous work to make the trajectories more complete and less noisy. These try to fill the gaps for such sparse GPS trajectories while mapping them to the road network, such that they are still usable in further analyses. All algorithms essentially try to infer from the context where on the network the measured locations should be and aim to estimate reasonable paths along the road network between these locations. This contextual data is often attached to the road network: estimated travel times or speed limits may be recorded on road segments. Alternatively, contextual data on the trajectory measurements themselves, such as recorded velocity and heading, may be used to guide the algorithms.

Contributions In this chapter, we provide an algorithm to verify that a given route is physically consistent for given speed bounds on the road network and acceleration bounds on the entity. Then, we present a new map-matching algorithm that leverages physical constraints on a vehicle when traversing the road network (Section 3.1). In particular, we assume that the vehicle has bounded acceleration (and deceleration). We also assume that the vehicle is subject to legal constraints: it must adhere to speed limits on the road network. This allows us to leverage the physical consistency checking algorithm for routes in our map-matcher. Our algorithm finds a solution space of routes that are feasible under the given speed limits and acceleration bounds, which acts as a strong quality guarantee. Between the different routes, we select a route according to geometric length and road-type changes.

In Section 3.3 we present the results of a brief experiment into the effect of such physical consistency on the quality of map-matching results, comparing to two baseline hidden Markov models. We observe that consistency helps in ensuring high-quality routes, though it may struggle to find routes when the data violates the assumptions on movement. Yet, this is detectable and points towards discrepancies between the set constraints and the movement of the vehicle. We discuss our results and avenues for further investigation in Section 3.4.

Related work Map matching is a well-studied topic in the GIS community. We refer the reader to recent surveys for a comprehensive overview [31, 66]; here we highlight the results that our most relevant in our context.

The most commonly used map-matching algorithms are based on Hidden Markov Models (HMMs). In these approaches a group of candidate points that lie on the road network are selected and candidate paths are created to connect them. The algorithm itself is incremental and maintains the “best” paths to each of the candidate points of a measured location. At each stage these paths are extended to the next set of candidates points and again the best path to each candidate is stored. Which paths are best is decided through a set of probabilities based on various measures. Differences between approaches occur based on how probabilities are assigned to different candidates. The simplest and most common factor is the distance of a candidate to its measured location. A second common factor is to consider the length of the path to each candidate. One could aim to have this distance be close to the distance between the measured locations [94] or to have it match with the time between measurements based on speed limits [56]. Other options include considering the number of turns [98].

Outside of the incremental HMM method one can also consider speed bounds in a more global manner. In the ST-matching method as proposed by Lou *et al.* [84] a graph is constructed that models paths between candidates. This graph is then weighted based on discrepancies between the time it would take to traverse a path and the time differences between the measurements. One can then find a globally good path through this graph. This approach can also be extended to include measured directions of movement [65, 32].

In our work we aim to ensure that the paths found are physically consistent to provide a quality guarantee on our output. That is, the paths produced are physically realizable within the given acceleration bound and speed limits. Using physics to model driver behavior is a common technique in trajectory processing and GIS analysis in general [89]. In its most simple form, one leverages the velocity of a vehicle to determine whether a certain distance can be traveled in a given time. In a GIS context, such a construction is commonly referred to as the space-time prism. Depending on the type of problem at hand, this can be sufficient. In other problems where the kinetic properties of a vehicle play a larger role, a more complex model can be useful, where acceleration is taken into account [83, 75, 36], as we also investigated in Chapter 2. For graphs Ardizonni *et al.* [9] describe how to find a fastest path under speed and acceleration bounds. This is similar to what we are interested in, however, in our case we are not interested in the fastest path, but in a path that is physically consistent with the measured locations and times. Additionally, the algorithm proposed by Ardizonni *et al.* does not run in polynomial time for general road networks.

Physical consistency Recall from the previous chapter that we consider two measurements of a trajectory T *consistent* if the vehicle could indeed travel in the time span between the measurements from the location of the earlier measurement to that of the later one, considering properties such as maximum speed, acceleration, or turning rate. The actual movement satisfying such physical properties is then a *witness* to the consistency.

In our map-matching application, we assume that our trajectories may contain noise. We then aim to decide whether a witness exists that describes movement over the road network that adheres to restrictions both physical (acceleration) and legal (speed bounds), while arriving at a location close to each measurement at the designated time.

Recall that if we consider only speed limits, the model is concatenable, meaning that we can decide consistency between measurements independently, and combine them without violating consistency. That is, in such a model, it does not matter how one reaches a measurement, for deciding on how to continue to the next. For locations on the road network, the problem then reduces to computing fastest paths between the measurements.

Instead, we consider a more detailed model that includes acceleration (and deceleration) bounds, which is conditionally concatenable: how the witness arrives at a measurement (in our case, at which speed) influences if or how we can reach the next measurement.

▸ 3.1 Consistent map matching

Our overall algorithm for map matching based on consistency is designed similarly to a Hidden Markov Model. Our method maps a trajectory T to a given road network $\mathcal{G}$ as follows:

1. First, we generate c candidate locations on $\mathcal{G}$ for each of the measurements in T. These candidate locations are the nearest point on the c nearest road segments in $\mathcal{G}$.
2. Then, we generate a consistent route for every candidate location of one measurement to each candidate location of the next measurement, if such consistent route indeed exists. Since we have bounded acceleration, it matters at which speed the vehicle is driving at the former candidate location, to decide how it can continue to the next. As such, we must keep track of intervals of speeds at which the vehicle could have reached the candidate location, in order to ensure consistency of the entire eventual route.
3. Finally, we trace back through our generated consistent routes from candidate location to candidate location, to reconstruct the overall route.

In the upcoming sections, we first treat the main question: how do we generate consistent routes, adhering to speed and acceleration bounds, in a road network, from one candidate location to the next? Afterwards, we briefly discuss various other implementation concerns and optimizations of our method.

▸ 3.1.1 Computing a consistent route

Validating a given route We begin our algorithmic consideration with a validation question: how do we check whether a given route, travelling through candidate locations is indeed consistent? That is, does a witness exist that demonstrates that these candidate locations can be visited at exactly the given times, while adhering to the local speed bounds and global acceleration bounds? Overall, we use an approach akin to that presented in the previous chapter: knowing the interval(s) of speeds at which the vehicle could travel at one candidate location, we *propagate* this under consistency to the next candidate location to find new interval(s). However, where we considered a global speed bound in the previous chapter, our speed bound varies along the route. As such, the method for propagating from one location to the next, even if we know the route, needs reconsideration.

To propagate, our input hence is four-fold: (1) a route through the network – a sequence of edge lengths $\langle \ell_1, \dots, \ell_m \rangle$ and speed limits $\langle \mathrm{v}_1, \dots, \mathrm{v}_m \rangle$; (2) an interval of possible initial speeds $[\mathrm{v}_\downarrow, \mathrm{v}_\uparrow]$; (3) a total time Δt between the candidate locations; (4) an interval of realistic acceleration $[a_\downarrow, a_\uparrow]$. We wish to determine the interval $[\mathrm{v}_{\downarrow,\mathrm{f}}, \mathrm{v}_{\uparrow,\mathrm{f}}]$ of final speeds that the vehicle may travel at, when it reaches the end of the route, while taking exactly Δt time and adhering to the local speed limits and global acceleration bounds. Note that we use speed instead of velocity, assuming that the vehicle always travels forwards along the route. Moving backwards is in principle possible, but unlikely to be done in such amounts as to affect consistency between candidate locations – driving back and forth can be mimicked by standing still after all.

As computational model we assume the real RAM model with square roots, such that square root computations are constant time operations. We discuss robustness implications when we discuss our implementation.

SIS diagrams We introduce a *space-inverse-speed* diagram (SIS diagram) as a tool to reason about witnesses. In a SIS diagram, we plot the inverse speed of the vehicle as a function of space. By our assumption of only moving forward, this is indeed a function. In the limit of moving arbitrarily slowly, the vehicle can also stand still. We use the inverse speed, such that the integral of this function is exactly the travel time. To match intuition of the function in the diagram visually going up to increasing speed, we draw the vertical axis downwards, with zero (of inverse speed, so infinite speed) at the top. The travel time is then the area above the curve.

The local speed bounds translate to horizontal segments in the SIS diagram. At the transition, there is a jump in the speed limit, but due to the acceleration bounds, the vehicle cannot discretely change its speed. To strictly adhere to limits, acceleration or deceleration to reach the new speed bound must happen in the segment that has a higher speed bound. Hence, considering also acceleration bounds as well as the maximum initial speed $\mathrm{v}_\uparrow$, we obtain a continuous function $U(x)$ that indicates the

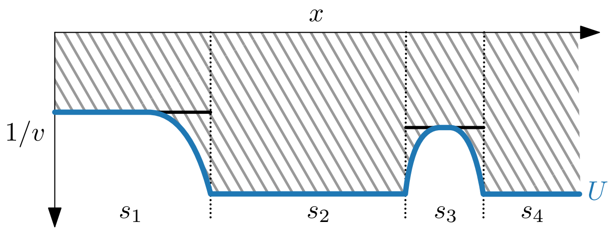


Figure 3.1 Example SIS diagram. Shown are four segments s_1, s_2, s_3, s_4, with the blue line indicating the maximum achievable speed U. The hatched area indicates the travel time to traverse these segments at maximum speed and acceleration. Witnesses must stay on or below U.

maximum achievable speed at distance x along the route, measured from the starting point. Any witness must stay below or on this function in the diagram. An example is given in Fig. 3.1.

The function U is defined by pieces of movement following extremes; also in our later proofs, we use extreme movement. We distinguish the following types.

(S) The vehicle travels at maximum local speed v as defined by $\mathcal{G}$. This corresponds to a horizontal line at 1/v in a SIS diagram.

(A) The vehicle accelerates maximally, at a rate of $a_\uparrow$. If the initial speed is $\mathrm{v_i}$, then we can express speed as a function of time as $\mathrm{v}(t) = \mathrm{v_i} + a_\uparrow t$. The distance traveled is then expressed as

$$x(t) = \int_{y=0}^{t} \mathrm{v}(y)\mathrm{d}y = \mathrm{v_i}t + \frac{a_\uparrow}{2}t^2. \tag{3.1}$$

However, we need speed as a function of space. Solving the latter for t gives us time as a function of space:

$$t(x) = \frac{-\mathrm{v_i} \pm \sqrt{\mathrm{v_i}^2 + 2a_\uparrow x}}{a_\uparrow}. \tag{3.2}$$

Using this in the former gives $\mathrm{v}(x) = \mathrm{v}(t(x)) = \pm\sqrt{\mathrm{v_i}^2 + 2a_\uparrow x}$. Since we assume positive speeds, this equals $\mathrm{v}(x) = \sqrt{\mathrm{v_i}^2 + 2a_\uparrow x}$. In a SIS diagram, this piece is $1/\mathrm{v}(x)$ and thus a decreasing curve (going up in the drawing).

(D) The vehicle decelerates maximally. That is, it accelerates at a rate of $a_\downarrow$, since $a_\downarrow$ will be negative. Following the same computations, this defines $\mathrm{v}(x) = \sqrt{\mathrm{v}_\mathrm{i}^2 + 2a_\downarrow x}$, an increasing curve in the SIS diagram (going down in the drawing).

Pieces of type (A) and (D) we also sometimes define based on their final speed v_f. Using $L = \sum \ell_i$, these curves are then expressed as $\mathrm{v}(x) = \sqrt{\mathrm{v}_\mathrm{f}^2 - 2a(L - x)}$ where a is $a_\uparrow$ for accelerating towards this final speed, and $a_\downarrow$ for decelerating.

Computing propagation What follows is a sequence of lemmata, to establish eventually that we can propagate an interval of speeds from one candidate location to the next, for a given route, in time linear in the complexity of the route.

3.1.1 Lemma. *For a route of m segments and an initial speed interval $[\mathrm{v}_\downarrow, \mathrm{v}_\uparrow]$, we can compute U in $O(m)$ time.*

Proof. We assume that each segment has a speed bound, different from its direct neighbors in the path; if not, we preprocess the route in linear time to merge these segments. We build U incrementally, storing the pieces of the curve in a stack, keeping track of the current speed v_c of U, which is initially $\mathrm{v}_\uparrow$, the maximum initial speed of the route. For each segment in the route, we now proceed as follows:

We want to add a type-(S) curve to U for the current segment ℓ_i with speed bound v_i. However, we must accommodate for accelerating or decelerating to this new speed bound. So, if $\mathrm{v}_i > \mathrm{v}_\mathrm{c}$, we must accelerate. To this end, we intersect the type-(A) curve starting at v_c with the type-(S) curve. If these intersect before the end of the current segment, we add the appropriate pieces of the accelerating curve and maximum-speed curve to the stack; v_c is set to v_i. Otherwise, we add only the accelerating curve, setting v_c to the speed at the end of the current segment.

If $\mathrm{v}_i < \mathrm{v}_\mathrm{c}$, we must decelerate such that we achieve v_i at the start of the current segment, this defines a type-(D) curve. We need to revise the curves on our stack to accommodate this curve. To this end, we pop curves from the stack, until the type-(D) curve intersects the top curve on the stack. We shorten that curve to this intersection and add the decelerating curve, followed by the maximum-velocity curve of the current segment. We set v_c to v_i. If the stack becomes empty during this process, we check the speed of the curve at $x = 0$. If it is below $\mathrm{v}_\downarrow$, then no witness exist for the given initial interval, and we stop the computation. Otherwise we place the type-(D) curve on the stack.

Each of the above computations can be done in $O(1)$ time. Adding a segment where the speed increases readily takes only $O(1)$ operations. Though adding a segment where the speed decreases may need to pop multiple curves from the stack. each segment adds at most two curves to the stack: the total number of curves to pop from the stack is at most $2m$ in total over all segments. As a result, this procedure runs in $O(m)$ time. □

3.1.2 Corollary. *We can test whether a witness taking at most Δt time exists for a given route in linear time.*

Proof. We compute U in linear time (Lemma 3.1.1), and compute the area above the curve for each of its pieces (if it exists). This area represents the fastest time, and hence we compare it to Δt. □

3.1.3 Lemma. *If a route admits both a witness with total time less than Δt and a witness with total time more than Δt, then a witness exists that takes exactly Δt time.*

Proof. Let $\mathrm{v}'(x)$ and $\mathrm{v}''(x)$ denote the speed functions over space of both witnesses. Consider a convex combination of these functions, $\mathrm{v}(x) = \lambda \mathrm{v}'(x) + (1 - \lambda)\mathrm{v}''(x)$. To prove that $\mathrm{v}(x)$ is a witness, it must adhere to the speed bounds as well as acceleration bounds. Since v' and v'' are witnesses, we know that they adhere to these bounds as well. Since the speed and acceleration of the convex combination must be between the speed and acceleration of these witnesses, we know that v is a witness as well.

The velocity-over-space functions directly imply a travel time (the area over the curve in the SIS diagram). As the convex combination changes continuously as a function of λ, so does the implied travel time. Since v' has a travel time less than Δt and v'' more than Δt, there must exist a λ where the travel time is exactly Δt. □

3.1.4 Lemma. *Given a route, total time Δt and an interval $[\mathrm{v}_\downarrow, \mathrm{v}_\uparrow]$ of initial speeds, we can compute the interval $[\mathrm{v}_{\downarrow,\mathrm{f}}, \mathrm{v}_{\uparrow,\mathrm{f}}]$ of final speeds for all witnesses in linear time.*

Proof. We construct U, the maximal attainable speed over space, using Lemma 3.1.1 in linear time for the given interval. By iterating over the curves, we can compute the total travel time in linear time as well. If this exceeds Δt or if U does not exist, then no consistent witness exists and the final interval is empty. Otherwise, we continue as follows.

Conceptually, the maximum final speed $\mathrm{v}_{\uparrow,\mathrm{f}}$ can be obtained by first decelerating for some distance, possibly fully stopping, followed by a maximal acceleration towards the speed bound. Our goal is then to find a witness with total travel time equal to or exceeding Δt with an as high as possible final speed: Lemma 3.1.3 then implies

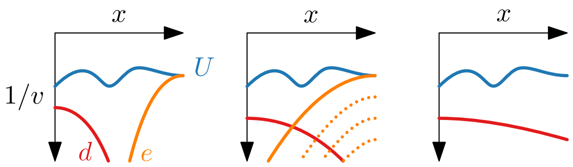


Figure 3.2 Several cases for finding $\mathrm{v}_{\uparrow,\mathrm{f}}$: if d and e do not intersect and reach speed zero, U determines the maximum speed at the end of the route (left); otherwise we search for an ending speed by finding a shift of e where we achieve exactly the right time span Δt (middle); though such may not exist if d does not reach zero (right).

that a witness with the right travel time exists, by combining it with U (the convex combination does not decrease the final speed after all).

Let v_U denote the maximum achievable final speed by U. We construct the following two curves: a type-(D) curve $d(x)$ maximally decelerating starting from speed $\mathrm{v}_\downarrow$; a type-(A) curve $e(x)$ maximally accelerating to end with speed v_U. We consider both curves only until speed zero is reached, or the other endpoint of the route. Note that, by construction, these curves cannot properly intersect U.

We distinguish cases, based on how d and e intersect each other and the limits of the SIS diagram; see also Fig. 3.2.

If d and e intersect, we construct the witness they imply through taking $\mathrm{v}(x) = \max\{d(x), e(x)\}$. If this witness takes at least Δt time, we are done and conclude that $\mathrm{v}_{\uparrow,\mathrm{f}} = \mathrm{v}_U$. If this witness takes less time, we first test whether a slow enough solution must exist: if d reaches zero, then this is always the case. Otherwise, we test the time taken by the witness d. If this slowest witness is still too fast, then no solution exists and the final interval is empty. Otherwise, we search for a minimal shift[1] of e, such that the resulting witness, constructed with the intersection of d takes exactly Δt time. Since we know a solution must exist before reaching speed zero, this equates to solving

$$\int_{x=0}^{L} 1/\max\{\sqrt{\mathrm{v}_\downarrow^2 + 2a_\downarrow x}, \sqrt{\mathrm{v}_{\uparrow,\mathrm{f}}^2 - 2a_\uparrow(L - x)}\}\mathrm{d}x = \Delta t.$$

---

[1] Note that this is not a geometric translation of the curve in the SIS diagram; technically, we shift the final speed, which then determines a different curve.

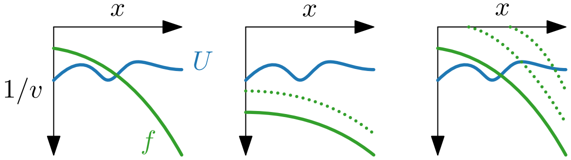


Figure 3.3 Several cases for finding $\mathrm{v}_{\downarrow,\mathrm{f}}$: if f intersects U such that this witness takes at most Δt time, we can reach speed zero (left); if f stays fully below U, we may need to adjust for $\mathrm{v}_\downarrow$ (middle); if the initial witness takes too much time, we shift f up, reducing the time span, to find the right value for $\mathrm{v}_{\downarrow,\mathrm{f}}$ (right).

As we show in Section 3.2, the solution to this equation is

$$\mathrm{v}_{\uparrow,\mathrm{f}} = \sqrt{\mathrm{v}_\downarrow^2 + 2a_\downarrow x' + 2a_\uparrow(L - x')},$$

with

$$x' = -\sqrt{\frac{(\mathrm{v}_\downarrow + a_\uparrow \Delta t)^2(-2L + 2\mathrm{v}_\downarrow \Delta t + a_\uparrow \Delta t^2)}{a_\uparrow - a_\downarrow}} - \frac{2a_\uparrow L + 2a_\downarrow \mathrm{v}_\downarrow \Delta t - 2a_\uparrow^2 \Delta t^2 + a_\uparrow \Delta t(-4\mathrm{v}_\downarrow + a_\downarrow \Delta t)}{2(a_\uparrow - a_\downarrow)}.$$

If d and e do not intersect, we follow a similar procedure. If they both reach zero, then their implied witness must be slow enough (since we can just wait), and we set $\mathrm{v}_{\uparrow,\mathrm{f}} = \mathrm{v}_U$. However, if they do not intersect as e reaches $x = 0$ above d, then we should test whether e as a witness is slow enough. If it is, then we also conclude $\mathrm{v}_{\uparrow,\mathrm{f}} = \mathrm{v}_U$. Otherwise, we follow the same search procedure as above, first testing whether the slowest option constructed via d only is sufficiently slow.

Provided the above computations found $\mathrm{v}_{\uparrow,\mathrm{f}}$ instead of concluding an empty interval, what remains is to compute the minimum final speed $\mathrm{v}_{\downarrow,\mathrm{f}}$. We take a conceptually similar approach: to minimize the final speed, we must traverse as quickly as possible early on, to slow down as much as possible at the end, while achieving the right travel time Δt. If we consider the type-(D) curve f of maximally decelerating curve to end at speed $\mathrm{v}_{\downarrow,\mathrm{f}} = 0$, then we may use this curve in combination with U to construct such a witness (taking the minimum). Again, we must distinguish several cases; see also Fig. 3.3.

We intersect f with U to construct a witness ending at speed zero. If this witness takes less than Δt time, we are done: we can wait at the end until the right time has passed. However, if this witness takes more time, then the witness needs to increase its speed; this can be done only by increasing the final speed. As increasing this final speed will monotonically increase the time of the constructed witness, and intersect U at increasing values of x, we may simply test the type-(A) and type-(S) pieces of U whether they will contain the intersection point that leads to an exact time of Δt, and subsequently solving for the exact intersection point, akin to the above description. For a type-(A), similar to $\mathrm{v}_{\uparrow,\mathrm{f}}$, the solution is given by $\mathrm{v}_{\downarrow,\mathrm{f}} = \sqrt{v^2 + 2a_\uparrow x' + 2a_\downarrow(L - x')}$, where

$$x' = \frac{2La_\downarrow + (a_\uparrow - 2a_\downarrow)\Delta t'(2v + a_\downarrow\Delta t')}{2(a_\uparrow - a_\downarrow)} \pm \sqrt{\frac{(v + a_\downarrow\Delta t')^2(2L - \Delta t(2v + a_\downarrow\Delta t'))}{a_\uparrow - a_\downarrow}}.$$

Here, $\Delta t'$ is the total time Δt minus the travel time (area above of the curve) of U, left of the type-(A) curve and v the initial velocity of the curve. For a type-(S) curve, the solution is

$$\mathrm{v}_{\downarrow,\mathrm{f}} = \sqrt{v^2 + 2a_\downarrow\left(L - v\Delta t' + \sqrt{2}\sqrt{\frac{v^2(c - v\Delta t')}{a_\downarrow}}\right)}$$

with v the speed of the type-(S) curve and $\Delta t'$ similarly defined as for the type-(A) curve. For details on these derivations, refer to Section 3.2.

In the above, we must take care of a boundary case: if f does not intersect U at all, and has a speed at $x = 0$ below $\mathrm{v}_\downarrow$, then we shift it such that f starts at this minimum initial speed. Since we then do not reach speed zero, we cannot wait arbitrarily: if this curve has a time less than Δt, there is no consistent witness spanning exactly Δt time. Note, however, that this last case cannot occur, since we have already concluded that a witness exists via the maximum final speed.

We spend linear time to compute U by Lemma 3.1.1. The computations for the maximum speed require a constant number of cases to be checked. The computations for the minimum require traversing U at most once, and thus also takes linear time. Hence, the entire propagation procedure runs in linear time. □

3.1.5 **Theorem.** *Given a trajectory T whose measurements lie on a route P through a road network, we can compute in linear time whether P is consistent with T given global acceleration bounds and local speed bounds.*

Proof. We begin by partitioning P into the subroutes that reflect the pieces from one measurement to the next in T; this can be done trivially in linear time. Now, we initialize an interval of speeds from zero to the local maximum speed at the first measurement in T. We use Lemma 3.1.4 to repeatedly propagate this interval the an interval of speeds at the next measurement, for the given subroute. This takes linear time in the complexity of the subroute, and thus linear time overall. We conclude that P is consistent with T if this interval is never empty. □

Finding a route The goal of our map matcher is to find a route, rather than just to validate a given one. However, generalizing our method above to a find a route in a network is problematic. Our approach relies heavily on velocity being a function of space (distance along the given route), avoiding a need for explicit consideration of when we arrive at a specific location along the route – this is captured implicitly by the area above the curve after all. For a given route, this simply means that we assume the vehicle did not drive backwards.

In a network we need an explicit consideration of intermediate vertices in order to aggregate information about different routes through $\mathcal{G}$, if we are to avoid enumerating all routes. However, this implies that we must consider not only the speed at a vertex, but also how long it took the vehicle to get to this vertex. That is, the information we need goes from 1-dimensional (interval of speeds, at a fixed time) to 2-dimensional (areas in a "speed-time" diagram). Propagation of such information is cumbersome at best, even if we assume some form of order between the vertices (e.g., the spanned subnetwork is a DAG). But generally, this subnetwork will not have a clear order, and we may visit two vertices in either order in a route; even if we assume we are after a simple route and thereby forbid revisiting a vertex in a route, computationally, we may need to consider vertices multiple times, since the order of traversal matters.

Hence, we opt to use another assumption: the vehicle moves along some route that is "reasonable": a fast route, though not necessarily the fastest. With this assumption in mind, we thus generate k fastest alternative routes between two candidate locations, using purely the speed bounds. Specifically, we use the k-shortest paths algorithm [133] with edge weights corresponding to traversal times of the edges. For each alternative route, we consider consistency and propagate intervals, using the method described above. Though we could also consider computing the fastest path with acceleration bounds [8], the parameter used in the presented algorithm is higher than a trivial constant, leading to impractical running times.

Fragmentation When combining feasible intervals from multiple paths between two measurement, it may be the case that the resulting intervals at the second measurement are disjoint. This phenomenon is referred to as fragmentation of the speed interval [36]. Since we can mimic free movement as considered in [36] through a fully connected road network, fragmentation in our case is also in the worst case at least linear, and at most exponential – though the exponential term now relies on the number of paths through the network, rather than the number of measurements (since we do not skip any measurements). In our map matcher, we combine intervals into a single interval if they overlap for further propagation, though we do store the intervals separately to support reconstructing the route.

3.2 Intermezzo: computing $\mathrm{v}_{\uparrow,\mathrm{f}}$ and $\mathrm{v}_{\downarrow,\mathrm{f}}$

For completeness, we briefly discuss in this section how to derive the analytical functions for the computations of $\mathrm{v}_{\uparrow,\mathrm{f}}$ and $\mathrm{v}_{\downarrow,\mathrm{f}}$. To compute $\mathrm{v}_{\uparrow,\mathrm{f}}$, we need to solve the equation

$$\int_{x=0}^{L} 1/\max\{\sqrt{\mathrm{v}_{\downarrow}^2 + 2a_{\downarrow}x}, \sqrt{\mathrm{v}_{\uparrow,\mathrm{f}}^2 - 2a_{\uparrow}(L-x)}\}\mathrm{d}x = \Delta t.$$

We first reformulate the equation, by expressing e as the function that starts at the intersection point between d and e with the velocity of d at that intersection. We introduce x' to be the x coordinate where d and e intersect. This allows us to eliminate the max term, which is replaced by two separate integrals (0 to x' over curve d, and x' to total length L over curve e). It also eliminates $\mathrm{v}_{\uparrow,\mathrm{f}}$, which is now readily implied from x'. The equation becomes, after integrating the individual terms:

$$\frac{-\sqrt{\mathrm{v}_{\downarrow}^2 + 2a_{\uparrow}x'} + \sqrt{2La_{\downarrow} + \mathrm{v}_{\downarrow}^2 + 2a_{\uparrow}x' - 2a_{\downarrow}x'}}{a_{\downarrow}} + \frac{-\mathrm{v}_{\downarrow} + \sqrt{\mathrm{v}_{\downarrow}^2 + 2a_{\uparrow}x'}}{a_{\uparrow}} = \Delta t$$

To solve this equation analytically, we use Wolfram Mathematica 12. We apply the Reduce function, adding non-negativity or negativity constraints for $a_{\downarrow}, a_{\uparrow}, \Delta t, x'$ and L. This results in the following solution for intersection point x':

$$\begin{aligned} x' \quad = \quad & -\sqrt{\frac{(\mathrm{v}_{\downarrow} + a_{\uparrow}\Delta t)^2(-2L + 2\mathrm{v}_{\downarrow}\Delta t + a_{\uparrow}\Delta t^2)}{a_{\uparrow} - a_{\downarrow}}} \\ & - \frac{2a_{\uparrow}L + 2a_{\downarrow}\mathrm{v}_{\downarrow}\Delta t - 2a_{\uparrow}^2\Delta t^2 + a_{\uparrow}\Delta t(-4\mathrm{v}_{\downarrow} + a_{\downarrow}\Delta t)}{2(a_{\uparrow} - a_{\downarrow})} \end{aligned}$$

With the above expression for x', we can now compute $\mathrm{v}_{\uparrow,\mathrm{f}}$ using

$$\mathrm{v}_{\uparrow,\mathrm{f}} = \sqrt{\mathrm{v}_{\downarrow}^{2} + 2a_{\downarrow}x' + 2a_{\uparrow}(L - x')}$$

Similarly, for $\mathrm{v}_{\downarrow,\mathrm{f}}$, we solve the same equation, with $a_{\downarrow}$ replaced by $a_{\uparrow}$ and vice versa. Note that different non-negativity constraints now apply. We replace $\mathrm{v}_{\downarrow}$ with v, the velocity at the start of the type-(A) curve we want to intersect. Let $\Delta t'$ be the total time Δt, minus the area above curve U, left of the type-(A) curve.

Again solving for the intersection point, we arrive at

$$x' = \frac{2La_{\downarrow} + (a_{\uparrow} - 2a_{\downarrow})\Delta t'(2\mathrm{v} + a_{\downarrow}\Delta t')}{2(a_{\uparrow} - a_{\downarrow})} \pm \sqrt{\frac{(\mathrm{v} + a_{\downarrow}\Delta t')^{2}(2L - \Delta t(2\mathrm{v} + a_{\downarrow}\Delta t'))}{a_{\uparrow} - a_{\downarrow}}}$$

where the sign is positive only if $\Delta t' > -\frac{\mathrm{v}}{a_{\downarrow}}$. In similar fashion to $\mathrm{v}_{\uparrow,\mathrm{f}}$, we can now compute $\mathrm{v}_{\downarrow,\mathrm{f}}$ using

$$\mathrm{v}_{\downarrow,\mathrm{f}} = \sqrt{\mathrm{v}^{2} + 2a_{\uparrow}x' + 2a_{\downarrow}(L - x')}$$

3.2.1 Other aspects

Slack on speed bounds In general, one might expect inaccuracies in locations of the considered candidates, but people may also slightly exceed the provided maximum speed. In other words, we have a “behavioral” model [36] when it comes to speed bounds. Additionally, it may occur that certain speed bounds on the road network are inaccurate or outdated. To be able to cope with these effects, we introduce a slack σ for the speed bounds, such that a vehicle is allowed to travel $1 + \sigma$ times the speed bound on the edges.

We have two approaches to using slack: either, we set a global parameter, effectively changing all speed bounds, or we apply slack when no physically consistent path with the current bounds exists. In the latter case, we gradually increase the slack if none of the candidate locations of a measurement have an interval of speeds that can be reached under the current bounds, to subsequently recompute physical consistency from the previous candidate locations to the current ones, using the increased slack.

Choices in reconstruction The solution space produced by the physically consistent map matcher may be quite large, if the vehicle is moving at a speed below the given bounds. To be able to select reasonable paths within the solution space, we propose to leverage the geometric length of reconstructed paths, as well as the number of changes of road types. If no explicit road types are available, this can also be mimicked using changes in speed bounds.

Specifically, we select routes such that length in meters plus 100 times type changes is minimal. Effectively, this means that routes may be slightly longer than the shortest path, if this saves on changing road types, but not more than 100m for each type change less than the number of changes in the shortest path.

Limiting search network To improve performance for searching shortest and fastest paths, we can limit the network to search in by taking the intersection of the space-time cones of consecutive measurements, assuming some large maximum velocity. This gives us ellipses around the measurements within which we can search for candidates and shortest or fastest paths. We use a maximum velocity of 45 m/s.

Reducing reconstruction solution space In the worst case, where all potential routes are physically consistent, the number of reconstructable physically consistent trajectories can be as large as $O((c^2k)^n)$, with c the number of candidates, k the number of fastest paths considered and n the complexity of the trajectory.

To keep the considered number of reconstructable routes manageable, we use the geometric length and road types as described earlier to weigh the reconstructed candidates. We build up the set of possible consistent routes between measurements, weigh the consistent routes, and retain only a fixed number of consistent subroutes to consider at the next measurement. We set this number to 100 in our experiments. Note that this may imply that we do not find the route that optimally minimizes the geometric length and road-type changes.

▸ 3.3 Experiments

Dataset We use a trajectory dataset provided by HERE Technologies[2] in the metropolitan region of Los Angeles, California, USA. We use a sample of the dataset, containing 30 trajectories. On average, these trajectories are 37 km long, take 38 minutes, and have a measurement every 1.7 seconds. For the road network, we use data from OpenStreetMap[3]. We use the tag annotation to infer the maximum speed

---

[2] `https://www.here.com`

[3] `https://www.openstreetmap.org`

whenever available, otherwise we use predetermined maximum speeds per road type according to local traffic rules.

Ground truth To estimate accuracy of the map-matching results, we construct a ground truth for all trajectories. We select trajectories in the dataset with high measurement frequency. We apply some clustering to remove any local noise and then map-match these trajectories using the map matcher by Newson and Krumm [94]. Afterwards, we manually verify that the result is a logical route for the given trajectory and manually make minor alterations to arrive at a good ground truth.

Sparsification We then create sparse trajectories by subsampling each trajectory as follows: we first fix a target time s between samples. Then, we go through the trajectory from start to end and drop any later measurement that is within time less than s for the currently considered measurement. We always retain the last measurement of the trajectory. We do this for $s \in \{2, 5, 10, 30, 60, 120\}$ seconds.

Performance measures We use the measures that are in line with previous work [66, 31]. Specifically, we use:

CMP The *correct matching percentage* (precision) is defined as the percentage of the correctly identified road segments (edges in the network) with respect to the ground truth. That is, if the result of the map matcher is a route P, and the ground truth is a route GT, then we interpret both as a set of edges and measure $|P \cap GT|/|GT|$.

R The *recall* is defined as the percentage of correctly identified road segments with respect to the complete resulting route. That is, we measure $|P \cap GT|/|P|$.

LCMP The *length correct matching percentage* is similar to CMP, but measures the length of common edges as a percentage of the total length.

C The *circuity* is similar to R, but measures edge lengths.

Algorithms We denote our physically consistent map matcher by PC, when using a fixed slack σ. The variant where the slack is modified dynamically is denoted by PC+. We compare our approach with two baseline map matchers that use an HMM: a pure spatial version, denoted by MS, where no additional data apart from the geometry encoded in $\mathcal{G}$ are considered [94]; and a spatio-temporal version, denoted by MT, that leverages the estimated travel time [74] based on speed bounds.

For all algorithms, we fix the number of considered candidates c to 5. For PC and PC+, we compute $k = 2$ fastest paths for connecting candidates. Thus, we generate a total of 50 paths for every pair of consecutive measurements of the input trajectory. Finally, PC uses a speed slack σ of 0.3, unless indicated otherwise; PC+ uses an initial slack of 0.1, increasing it up to 0.6 in steps of 0.05. We fix the acceleration bounds at $a_{\downarrow} = -5.0\text{m/s}^2$ and $a_{\uparrow} = 6.0\text{m/s}^2$, which corresponds to relatively safe driving [47]. For the baseline map matchers, we set the standard deviation for observation probabilities to 100m and the scale factor for the transition probabilities to 300m.

Implementation We implement the algorithms using Python, using NetworkX, pyproj and (Geo)Pandas, loading the OpenStreetMap data into a SpatiaLite database[4] for efficient querying. We visualize our data on the OpenStreetMap network using ipyleaflet. Since we only have finite precision and we work with complex operators to compute the speed intervals, robustness issues may arise. To be able to cope with this, we slightly scale up intervals when querying whether values lie within the intervals. This approach is especially useful when we propagate intervals backwards for reconstructing the routes when the computed intervals are very small.

3.3.1 Comparison with baseline map matchers

Qualitative comparison We start our evaluation by discussing two specific examples in Fig. 3.1. In the first example the result of PC nicely follows the ground truth, and we know that this is physically consistent (up to the used slack) by construction. The HMM models create less intuitive results and deviate from the ground truth: MS produces an unrealistic result, especially in the south-western corner of the graphic; MT does follow the highway over the interchange, but opted for a different road at the start, causing it to take a detour to get onto the highway. It is not obvious whether these choices are physically consistent.

In the second example both MT and MS opt for taking the shortcut, suggesting that it is a slightly shorter and faster route – though it may not be physically realizable if we factor in acceleration. On the other hand, PC remains on the highway, indeed following the ground truth. By construction, the route is physically consistent, and thus, in combination with fewer road-type changes, is deemed a more likely route. In other words, even though there may have been a theoretically faster route (excluding physical consistency and other factors such as traffic lights), we see that our model can nonetheless capture the right behavior.

---

[4] `https://www.gaia-gis.it/fossil/libspatialite/index`, version 5.0.1

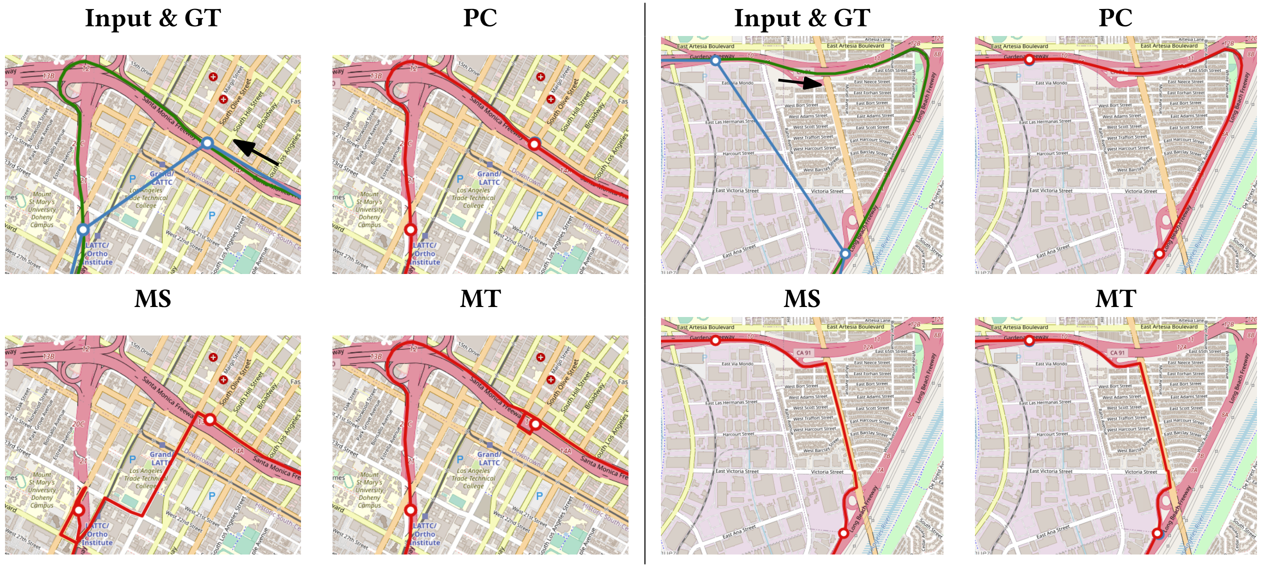


Table 3.1 Two snippets for a result subsampled with s = 120s. Shown are the subsampled trajectory (input, blue), the ground truth (GT, green), and map-matching results (red).

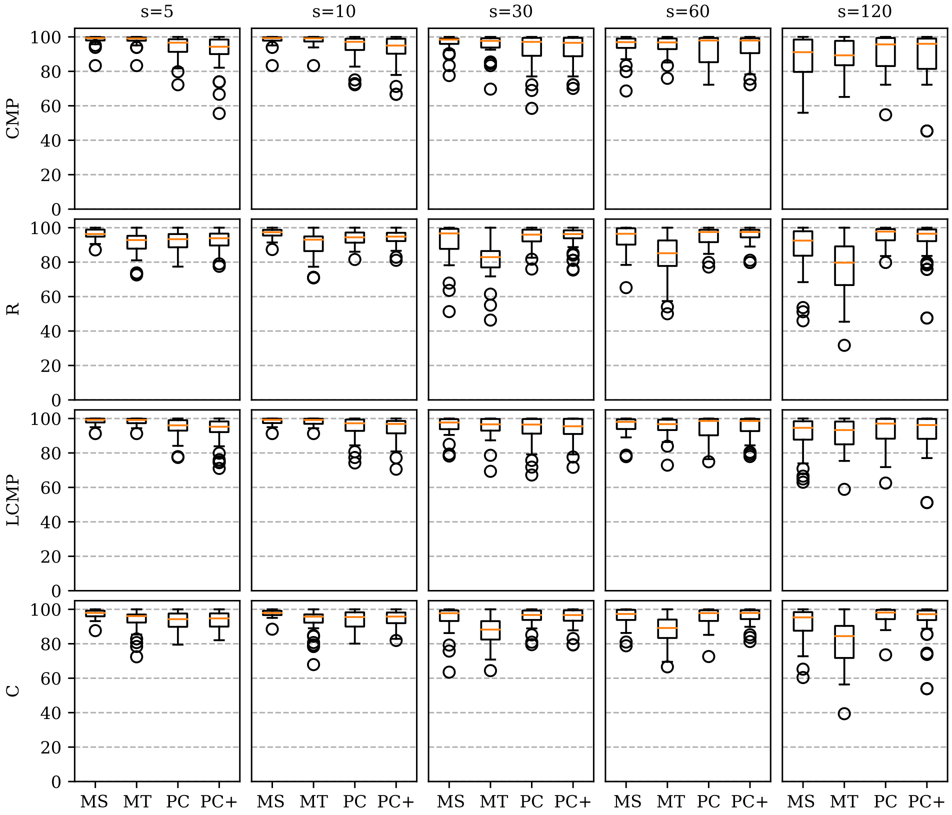


Figure 3.4 Comparison of the different methods on the performance measures, under varying sparsification *s*. For all measures, higher values indicate better performance.

Quantitative comparison Fig. 3.4 shows the performance measures for the different algorithms, separately for each subsampling time s. With very dense data, we see that PC and PC+ perform slightly under MS and MT in terms of precision (CMP, LCMP), though very similarly in recall (R, C). But as the trajectories become sparser, we see that the HMM-based methods take a more considerable performance hit, in both precision and recall. On the other hand, PC and PC+ keep a very similar recall: that is, even for sparse trajectories, the reconstructed routes remain of high quality, following physical consistency and road-type changes.

Precision for PC and PC+ is slightly reduced, but remains higher on the sparsest setting (s = 120s) compared to MS and MT. This reduction is caused by the disconnects that PC(+) may produce: when there are no physically consistent routes to be found. In other words, though it may not be able to reconstruct all parts of the route, the parts that it does find are of high quality. And rather than outputting a route of low quality like MS and MT (which may be hard to detect), such disconnects are trivial to detect to allow manual review or post-processing. Overall, we observe in our experiments that approximately one third of the produced routes of MS and MT are fully physically consistent. Approximately 41% of the routes produced by PC are full routes, where the other results consist of two or more parts.

Interestingly, we see that PC+ sometimes performs worse than PC, even though it has the ability to use higher slack. In cases where it did not disconnect, it uses 2.42 steps on average to increase the speed slack, resulting in an average slack of 0.221, which is below the 0.3 used for PC. However, this may result in more disconnects in certain cases. Starting with very low slack as PC+ can result in very small speed intervals from which the path cannot be continued consistently, even by a large increase in slack. With PC the slack already starts higher and it may be able to avoid this problem resulting in fewer disconnects. In the next section, we dive more into the effect of the slack parameter σ.

▸ 3.3.2 Effect of speed slack

Qualitative comparison. Fig. 3.5 shows a part of a result for PC, run with varying values for the speed slack σ and s = 120s. Since the result disconnects, we know that the vehicle must have been moving faster than the local speed bounds allow – or we may use it as an indication that the local speed bound is incorrect or outdated. As the slack increases, PC is able to recover more of the route. However, too much slack allows for routes that are unrealistic and do not match the ground truth: see for example the south-eastern corner for the σ = 0.5 result. Hence, we should

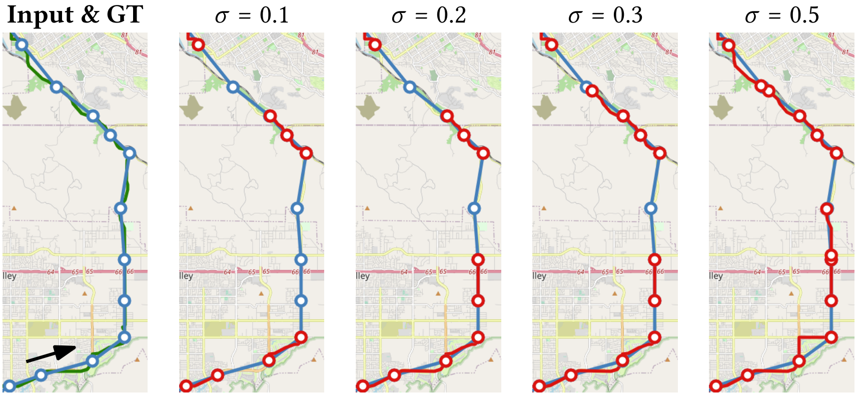


Figure 3.5 Snippet of a trajectory where PC detects that speeding occurs: with more speed slack, more connected segments are found for the input trajectory.

balance slack such that it allows some flexibility and handling of uncertainty, without allowing for unsatisfactory routes to become physically consistent.

Quantitative comparison Fig. 3.6 shows the performance measures for different values of σ. We see that precision (CMP, LCMP) indeed increases with higher slack, though the effect seems to be strongest for more dense trajectories. Recall (R, C) is much less effected, though we see that $\sigma = 0.3$ has fewer outliers than $\sigma = 0.5$. This further supports our conclusion of the qualitative comparison, in needing to balance the slack parameter.

▸ 3.4 Conclusion and discussion

We developed a method that allows us to map-match sparse trajectories, in such a way that the resulting route adheres to constraints on local speed and global acceleration. Though we did not experimentally investigate running time, in our implementation the bottlenecks are computing the k fastest paths between candidates and actually loading all map data; computing with consistency itself is efficient, as demonstrated by Theorem 3.1.5.

Our results suggest that such physical consistency can help finding the correct route, and the reconstructed routes contain few errors. However, we also saw that devia-

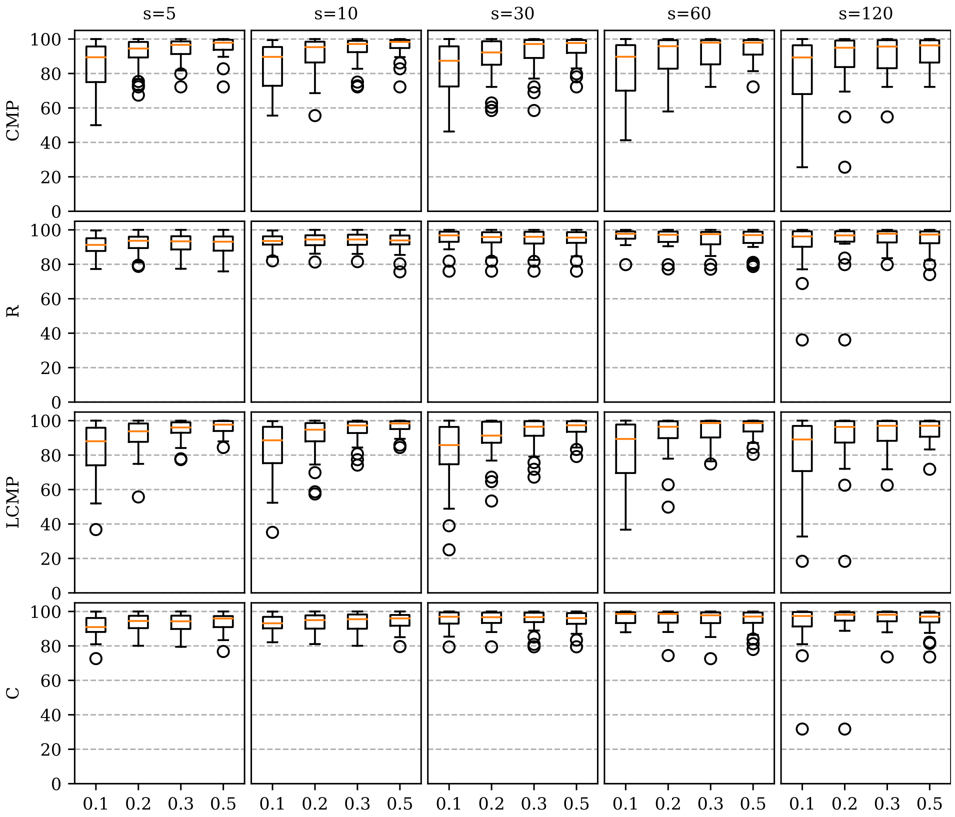


Figure 3.6 Performance of PC, when increasing the slack σ. For all measures, higher values indicate better performance.

tions from the local speed bounds (because the vehicle is violating the bounds, or because the data is inaccurate) can cause our method to not represent all parts of route. Yet, our method guarantees that the parts that are reconstructed adhere to the model of movement, and it helps in identifying problematic areas.

We note that our dataset consists of vehicles that have a clear destination, and follow an efficient route, which we can leverage by applying physical consistency. However, sparse trajectories of vehicles that do not exhibit such behavior are likely more problematic. For example, with a taxi roaming to find customers, there is simply not enough information to reasonably reconstruct the trajectories from sparse samples. This seems unavoidable, unless we can infer such from other context information, such as popular places and common roaming routes.

Below, we briefly review some possible avenues for future investigation.

Improving the map matcher We use only acceleration in our model, disregarding that taking sharp turns requires a lower speed. Though we could model a turning rate explicitly, it seems reasonable to assume that turns in the road network can be taken, if the vehicle travels at a sufficiently low speed. Such an approach can easily be modeled in our method, injecting zero-length edges at turns with a speed limit depending on the angle of the turn.

A common problem in HMM-based map matchers is the need for good candidate selection. Our presented method also suffers from this problem, disconnecting the path when no physically consistent route was found. If we are to follow an approach where we explicitly construct the reachable locations from a previous candidate location (instead of the generate-a-route-and-test approach in our current implementation), this may possibly be circumvented: rather than testing pairs of locations that are decided upfront, we may be able to find the locations that are physically consistent and closest to the next measurement. Alternatives include using continuous candidates (e.g., use an entire road segment as a “location”) or replacing candidates that are not consistent.

Our approach relies on driving at maximum speed, maximally accelerating and maximally decelerating, to decide consistency. Typically, this results in a variety of possible ways to have driven the reconstructed route. If we are to estimate the actual location (and thereby speed and acceleration) of the vehicle at all times, we are looking for the “most reasonable” witness and may consider minimizing e.g. the variation in acceleration, or the deviation from the local speed bounds.

The disconnects caused by our map matcher may arise due to inaccuracies in the data: not just the GPS measurements, but specifically also the speed limits. Integrating estimates of actual driven speed may help here. Yet, such data varies over time, possibly increasing the algorithmic complexity of ensuring consistency.

Using physical consistency We compared our method to comparatively simple HMM-based map matchers. The purpose here is to allow assessing the added value of physical consistency, without the interference of many facets as combined into a typical map matcher. With the potential of physical consistency demonstrated, we may look into how to integrate this into more complex map-matching algorithms. Specifically, we see potential to also integrate consistency into map matchers that provide an explicit matching between the trajectory and the reconstructed route, such as the Fréchet-distance-based map matcher [3].

Furthermore, it may be interesting to consider the “inverse” problem of map matching with physical consistency, in order to augment a road network or assess its quality: if we are given a route and acceleration bounds, what would reasonable speed limits on the network be? Or similarly, how much do the given speed limits need to change to accommodate a consistent route?

Chapter 4

Route Reconstruction from Traffic Flow via Representative Trajectories

It can not only be the case that trajectories themselves are incomplete as we discussed in the last chapter, but the same can apply to trajectory data sets. In general, when acquiring trajectory data of human movement, the data set is not a complete sample of all traffic in the road network. But for analysis of mobility patterns, it may be of interest to have a more complete picture of the trajectories and routes. Hence we can look at contextual data and see if we can enrich the trajectory data by employing this additional data, potentially filling the gaps of the trajectory data set.

Checkpoint data originate from measurements by static devices such as loop detectors or traffic cameras, placed on fixed locations throughout the road network. They provide a comprehensive view of the amount of traffic flow at that particular location, but inherently no information on how people navigate through the network. Tracking data, on the other hand – predominantly captured through GPS in smart phones and navigation systems – provide a detailed view of individual behavior in the form of trajectories. However, trajectory data does not describe the general traffic flow, as not all vehicles are tracked or tracked by the same system. Furthermore, the (often significant) detail in trajectories also raises privacy concerns, so trajectory data are frequently segmented and anonymized before analysis.

Both trajectory and checkpoint data thus give only a partial view on the full dynamics of human mobility. Given their complementarity it is natural to investigate possibilities for data fusion, that is, data enrichment that leverages the strength of both. Given loop-detector measurements as well as a (small) set of representative trajectories, we investigate how one can effectively combine these two partial data sources to create a more complete picture of the underlying mobility patterns. Specifically, we want to reconstruct a concise set of realistic routes from the loop-detector data, using the given trajectories as representatives of typical behavior.

Contributions and organization After a brief review of related work, we formally model our problem in Section 4.1, while also introducing the necessary concepts and notation. We arrive at a formal problem statement which models the loop-detector data as a time-independent network flow that needs to be covered by the reconstructed routes; we capture the realism of the routes via the strong Fréchet distance to the representative trajectories. In Section 4.2 we prove that several forms of the resulting algorithmic problem are NP-hard even in restricted settings. Hence, we explore heuristic approaches which decompose the flow well while following the representative trajectories to varying degrees. In Section 4.3, we propose an iterative *Fréchet Routes* (FR) heuristic which generates candidate routes with bounded Fréchet distance to the representative trajectories. In the same section we also describe a variant of multi-commodity min-cost flow (MCMCF) which is only loosely coupled to the trajectories.

In Section 4.4 we report on an experimental evaluation of these proposed approaches in comparison to a global min-cost flow baseline (GMCF) which is essentially agnostic to the representative trajectories. To make meaningful claims in terms of quality, we derive a ground truth by map matching real-world trajectories. We find that GMCF explains the flow best, but produces a large number of often nonsensical routes (significantly more than the ground truth). MCMCF produces a large number of mostly realistic routes that explain the flow reasonably well. In contrast, FR produces significantly smaller sets of realistic routes that still explain the flow well, albeit at the cost of a higher running time. In Section 4.5 we report on the results of a case study which combines real-world loop-detector data and representative trajectories for the region around The Hague, the Netherlands. Though we predominantly discuss loop-detector data in this chapter, our approaches work for any type of data source that can induce a flow field and are not restricted to loop-detector data. We reflect on our results and discuss avenues for further research in Section 4.6.

Related work Our problem is closely related to flow decomposition, where the goal is to decompose an aggregated flow into paths, optimizing a given objective function. Any flow can be decomposed into at most $O(E)$ paths and cycles, where E is the number of edges in the graph [2]. However, given a set of paths that decompose the flow, it is NP-hard to determine the correct integral coefficients for each path [72]. Minimizing the number of paths in a decomposition is also NP-hard [118], and thus various approximation algorithms have been developed [61]. We require that the reconstructed routes are similar to one of the representative trajectories, but we do not require that the flow is explained completely.

We are aware of only a single geometric approach to reconstruct flow from checkpoint data. Duckham *et al.* [46] use the Earth Mover's distance to estimate the movement of couriers from checkpoint data. Given the limited data, the results are quite accurate, but a full reconstruction of the movement is clearly out of reach.

Reconstructing a route (in a network) given a GPS trajectory is referred to as map matching [3, 62, 70, 122]; see also the survey by Quddus *et al.* [101]. The goal is to find a route in a given network, accounting for potential misalignment between GPS measurements and the network, noise inherent in GPS systems, and inaccuracies in the road network. There are a wide variety of available algorithms; various solutions are based on hidden Markov models, a strategy that was pioneered by Newson and Krumm [94]. The map-matching algorithm by Alt *et al.* [3] is particularly relevant to our work (see Section 4.3), as it decides in quadratic time whether a graph admits a path with Fréchet distance at most ε to an input trajectory. Generally, map-matching techniques are not designed to explain flow data, but rather to correct measurement errors in a single trajectory. Moreover, if we insist on simple routes, map matching is NP-hard for various measures, including the Fréchet distance [82] and Hausdorff distance [15]. If the road network is a perfect grid, routes with bounded (but not necessarily minimal) Fréchet and Hausdorff distance can be found efficiently [15].

The problem we study resembles traffic generation [91, 103]. The main difference is that we prefer a succinct set of routes, which is not the case for traffic generation approaches in general.

4.1 Modeling

Our input has three components: (1) a road network, given as a graph $\mathcal{G}$; (2) a set of representative trajectories $\mathcal{T}$, each encoded by a sequence of measurements; and (3) loop-detector data, which are traffic-volume measurements expressed as the number of cars at a specific time at a specific location. Our goal is to combine these heterogeneous data sources to create a more complete picture of the underlying mobility patterns. Specifically, we want to reconstruct a realistic set of routes from the loop-detector data, using the trajectories as representatives of typical behavior. Here we discuss the modeling decisions we made to finally arrive at a formal problem statement which is amenable to algorithmic treatment.

For ease of notation, we introduce the function $M(P, e)$ that indicates how often the edge $e \in E$ is traversed in a route P. Note that $M(P, e)$ is 0 or 1 if P is simple (necessary but not sufficient), but may take on higher values if P is not simple.

▸ 4.1.1 Modeling loop-detector data as time-independent complete flows

Loop-detector data is gathered by counting the number of vehicles passing an induction loop in the road network, aggregated over a fixed time interval. These time intervals generally range from minutes to hours and thus can give an accurate view on the traffic volume over time at that particular location.

Mathematically speaking, loop-detector data can be interpreted as an (incomplete) **flow** on the road network: we assign the aggregate loop-detector data to the edge in the road network where the detector is located and interpret the data as the volume of traffic through this edge. As not every edge in a road network has a loop detector, the flow data is a priori incomplete. Reconstructing a complete flow is challenging due to the inherent uncertainty surrounding the exact driven routes between detector locations [27]. We focus initially on **complete flow** information, and briefly consider incomplete flows in the case study.

Loop-detector data inherently depend on time and hence a priori imply a time-dependent flow. However, time-dependent flows pose several data, modeling, and complexity problems. First of all, we need additional data to model the time needed to traverse the network. We need to know the travel times for edges to be able to reason about realistic routes in the network for the flow. The realism of the reconstructed routes then heavily depends on the accuracy of these values and on time itself. Furthermore, time-dependent flows naturally require the use of time-dependent representative trajectories, in which case the set of representative trajectories will generally be (too) sparse. In addition, as we show in Section 4.2, time-independent formulations of our problem are already computationally hard, which suggests that the time-dependent problem will be even harder to compute. Hence, we model the loop-detector data via *time-independent* flow, which one can interpret as a “long-time average” of the loop-detector measurements.

We formalize the flow as follows: given a network $\mathcal{G} = (V, E)$, a *flow* (f, S, T) on $\mathcal{G}$ is specified by a flow function $f : E \longrightarrow \mathbb{R}_{\geq 0}$ mapping each edge of $\mathcal{G}$ to its flow value. In addition, there are sources $S \subseteq V$ and sinks $T \subseteq V$ for the flow. Commonly, a flow function must satisfy the flow-conservation property: for each vertex that is neither a source nor a sink, the sum over all incoming flow is equal to the sum over all outgoing flow. However, errors in actual data may cause violations of the flow-conservation property. Furthermore, traffic volume generally does not specify sources or sinks. We hence introduce the notion of a *flow field* to describe the measured traffic volume, which is a function $\phi : E \longrightarrow \mathbb{N}$ (not necessarily satisfying the flow-conservation property).

▸ 4.1.2 Reconstructed routes

Our goal is to compute a multiset $\overline{\mathcal{P}}$ of realistic routes that explains a flow field ϕ well. We represent the multiset $\overline{\mathcal{P}}$ by a base set of routes $\mathcal{P}$ (a *basis* $\mathcal{P}$ for short), along with associated frequency counts. It seems natural to assume that these counts should have integer values. However, with that restriction, even computing the correct counts for a specific basis to explain a given flow is NP-hard [72]. We therefore relax the counts $c : \mathcal{P} \longrightarrow \mathbb{R}_{\geq 0}$ to be fractional *coefficients*. This relaxation allows us to efficiently compute the coefficients for a specific basis and flow. We say that the real-valued multiset of routes $\overline{\mathcal{P}} = (\mathcal{P}, c)$ is a *reconstruction* of the flow field ϕ.

We need to quantify how well a reconstruction $(\mathcal{P}, c)$ explains the input flow field ϕ. To this end, we derive a flow field $\phi_{(\mathcal{P},c)}$ from $(\mathcal{P}, c)$ as follows:

$$\forall e \in E : \qquad \phi_{(\mathcal{P},c)}(e) = \sum_{P \in \mathcal{P}} M(P, e) c(P).$$

The error in the loop-detector measurements can be positive or negative, suggesting a measure based on the absolute difference between $\phi_{(\mathcal{P},c)}$ and ϕ per edge. In line with traffic-analysis literature [25, 26], we compute the *flow deviation* $\Delta(\mathcal{P}, c, \phi)$ as the sum of squared differences between $\phi_{(\mathcal{P},c)}$ and ϕ over all edges.

$$\Delta(\mathcal{P}, c, \phi) = \sum_{e \in E} (\phi(e) - \phi_{(\mathcal{P},c)}(e))^2$$

For a fixed basis $\mathcal{P}$, the coefficients c that minimize the flow deviation can be computed efficiently with standard techniques [17, 71].

▸ 4.1.3 Realistic routes

We are given a set $\mathcal{T}$ of trajectories that represent typical behavior of vehicles in the network. Our aim is to compute a realistic reconstruction of the flow field based on $\mathcal{T}$. We measure the realism of a route in the basis $\mathcal{P}$ via its distance to the closest trajectory in $\mathcal{T}$.

There are many possible similarity measures for polylines such as the Fréchet distance [4], the Hausdorff distance, and Dynamic Time Warping [13]. In our setting we might encounter a large difference (either way) in spatial resolution between the road network and the representative trajectories, since the sampling densities of vehicle trajectories differ greatly between providers and sampling technologies used. In the presence of such large discrepancy in sampling density, discrete measures such as Dynamic Time Warping and the discrete versions of the Fréchet distance and Hausdorff distance are known to perform poorly, since measurements have to be matched to vertices of the road network.

The Hausdorff distance and the weak Fréchet distance do not capture the order of points and edges in trajectories and are hence less suitable for our purpose. Hence, we choose the strong Fréchet distance to measure the realism of our reconstructed routes: it naturally captures the variability in the paths while encouraging that the general direction of reconstructed routes and representative trajectories are similar.

The (strong) *Fréchet distance* $d_F(P, Q)$ between two curves $P, Q : [0, 1] \longrightarrow \mathbb{R}^2$ is defined as

$$d_F(P, Q) = \inf_{\alpha,\beta} \sup_{t\in[0,1]} \|P(\alpha(t)) - Q(\beta(t))\|,$$

where α and β are reparameterizations of P and Q, respectively. The functions α, β must be strictly monotonically increasing, with $\alpha(0) = \beta(0) = 0$ and $\alpha(1) = \beta(1) = 1$. To determine if two curves lie at Fréchet distance at most ε, we use the so-called *free-space diagram* [4]. This diagram represents matching locations on curves P and Q that are within ε Euclidean distance of each other. Two curves have Fréchet distance at most ε if the diagram admits a strictly monotone path. This problem can be solved in $O(n^2)$ time where n is the total complexity of the two curves [4].

We say that a route is *realistic* if it lies within a prespecified Fréchet distance ε of the closest representative trajectory. The parameter ε controls the realism of our reconstruction. A reconstruction $(\mathcal{P}, c)$ is realistic if all routes $P \in \mathcal{P}$ are realistic.

The routes taken by vehicles tend to be simple, since humans generally take shortest paths to their destinations. Hence, we would prefer to reconstruct simple routes only. However, as we show in Section 4.2, even just minimizing the flow deviation is NP-hard for simple paths. Our heuristic approaches hence prefer simple routes but do not exclude non-simple ones.

▸ 4.1.4 Formal problem statement

Our complete input is the road network $\mathcal{G} = (V, E)$, the set of representative trajectories $\mathcal{T}$, the flow field ϕ induced by the loop-detector data, and realism parameter $\varepsilon > 0$. We want to find a realistic reconstruction $(\mathcal{P}, c)$ such that the flow deviation $\Delta(\mathcal{P}, c, \phi)$ is minimized. Following Occam's Razor, we are looking for a concise explanation of the flow field given the representatives, that is, we prefer reconstructions with small cardinality $|\mathcal{P}|$.

▸ 4.2 Computational complexity

In this section we explore the computational complexity of our problem. First of all, we restrict the reconstructed routes to be simple. In this setting, even computing just a single realistic route is NP-hard [82]. By extension, computing a realistic simple reconstruction is hard as well. Hence, we next consider a restricted variant of the problem, where the reconstructed routes do not have to be realistic and even share start and end point. Specifically, we require that all routes in the reconstruction are simple and start at a vertex s and end at a vertex t. We refer to such a reconstruction as an (s, t)-reconstruction. However, we show that even this simplified problem is NP-hard. For this result, we consider two variants of the deviation function: the sum of squared differences as defined in the previous section, but also the sum of absolute differences. In the following two theorems we refer to these as *squared* and *absolute* deviation, respectively.

4.2.1 Theorem. *Given a road network $\mathcal{G}$ with source s, sink t and an associated flow field ϕ, it is NP-hard to compute an (s, t)-reconstruction with only simple paths that minimizes the absolute deviation to ϕ.*

Proof. We show that it is NP-hard to determine whether there exists an (s, t)-reconstruction with simple paths such that the absolute deviation $\Delta_{\text{abs}}(\mathcal{P}, c, \phi) \leq \delta$ for some $\delta \geq 0$.

Our proof uses a reduction from the longest-path problem: given a graph $\mathcal{G}' = (V', E')$, a source s' and sink t' in V', and a threshold $L > 0$, decide whether $\mathcal{G}'$ admits a simple path of at least L edges from s' to t'. We turn this into an instance of our problem as follows. We augment the network to $\mathcal{G}$ by adding a new vertex

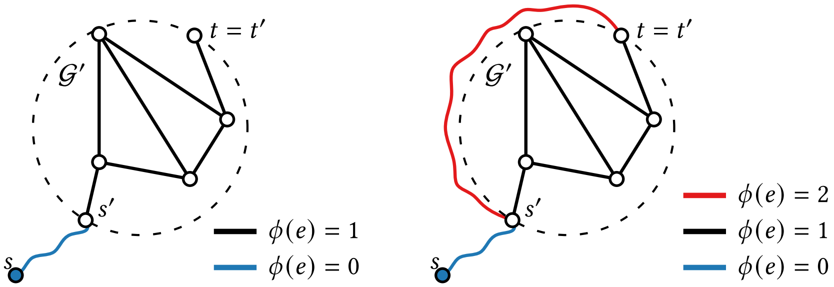


Figure 4.1 Schematic sketch of the constructions to prove NP-hardness of the (s, t)–reconstruction problem while minimizing the absolute deviation (left) and the squared deviation (right). The longest-path graph $\mathcal{G}'$ is in the dashed circle. For the squared deviation construction, E_1 is shown in blue and E_2 is shown in red.

s

s connected by a new path of $L - 1$ edges to s' s(see Figure 4.1, left). For the sink, we use the same vertex, $t = t'$. The flow field ϕ has value 1 for all edges in E', and value 0 otherwise. Finally, we set the deviation threshold at $\delta = |E'| - 1$. We claim that this instance admits an (s, t)-reconstruction with absolute deviation at most δ, if and only if $\mathcal{G}'$ admits a simple path of length at least L from s' to t'.

Assume that $\mathcal{G}'$ admits a simple path of length at least L from s' to t'. Let P' denote this path, and P the route in $\mathcal{G}$ consisting of the path from s to s' concatenated with P'. Consider the reconstruction consisting only of P with coefficient 1. The result is an (s, t)-reconstruction with only simple paths by construction. For every edge e originally from P' we have that $|\phi(e) - \sum_{P \in \mathcal{P}} M(P, e)c(P)| = 0$; for every other edge this value is 1. Thus, the absolute deviation $\Delta_{\text{abs}}(\mathcal{P}, c, \phi)$ is the number of edges along the path from s to s' plus the number number of edges in $\mathcal{G}'$ not covered by P'. Hence, this is at most $(L - 1) + (|E'| - L) = |E'| - 1 = \delta$.

Now, assume that an (s, t)-reconstruction $(\mathcal{P}, c)$ exists with absolute deviation at most δ. Specifically, we assume that $\Delta_{\text{abs}}(\mathcal{P} \setminus \{P\}, c, \phi) > \Delta_{\text{abs}}(\mathcal{P}, c, \phi)$ for all routes $P \in \mathcal{P}$. In other words, removing any route increases the deviation. Observe that a reconstruction without any routes would achieve deviation $|E'| > \delta$, thus the reconstruction must contain at least one route. Consider a route $P \in \mathcal{P}$. As P must start at s and end at t, and the flow field at all edges between s and s' is zero, we know that removing P from the solution locally decreases the deviation by $(L - 1) \cdot c(P)$. Since removing a path must increase deviation and the deviation at one edge can increase by at most $c(P)$, P must contain at least L edges originating from $\mathcal{G}'$. Hence, the subpath of P starting at s' and ending at t' has at least L edges. As a result, we conclude that $\mathcal{G}'$ has a simple path of length at least L.

Finally note that the construction can easily be performed in polynomial time, so the stated problem is NP-hard. □

4.2.2 Theorem. *Given a road network $\mathcal{G}$ with source s, sink t and an associated flow field ϕ, it is NP-hard to compute an (s,t)-reconstruction with only simple paths that minimizes the squared deviation to ϕ.*

Proof. We again use a reduction from the longest path problem. Let the instance of the longest path problem consist of a graph $\mathcal{G}' = (V', E')$ along with source s', sink t', and threshold L. We augment $\mathcal{G}'$ to obtain $\mathcal{G} = (V, E)$ as follows. We first add a new path of $L-1$ edges from the new source vertex s to s', and we refer to this set of new edges as E_1. Furthermore, we add a new path of $L-1$ edges from s' to t', and we refer to this set of edges as E_2 (see Figure 4.1, right). Hence we have $E = E' \cup E_1 \cup E_2$. For the sink, we use the same vertex $t = t'$. For the flow field ϕ we set $\phi(e) = 0$ if $e \in E_1$, $\phi(e) = 1$ if $e \in E'$, and $\phi(e) = 2$ if $e \in E_2$. We now claim the following: (1) the minimum deviation for the instance formed by $\mathcal{G}$ and ϕ is $|E|$ if the longest simple path in $\mathcal{G}'$ has length at most $L-1$, and (2) the minimum deviation is strictly smaller than $|E|$ if there exists a path of length at least L in $\mathcal{G}'$. Note that (1) and (2) taken together directly imply that the stated reconstruction problem is NP-hard.

For (1), first assume that the longest simple path in $\mathcal{G}'$ has length at most $L-1$. Consider the (s,t)-reconstruction $(\mathcal{P}, c)$ consisting of a single path $P^* \in \mathcal{P}$ with $P^* = E_1 \cup E_2$ and $c(P^*) = 1$. Note that the deviation of this reconstruction is exactly $|E|$. Now consider any simple path P between s and t and add it to $\mathcal{P}$ with $c(P) = 0$ (this does not really change the decomposition). By definition, the derivative of the deviation with respect to the coefficient $c(P)$ is given by:

$$\begin{aligned}\frac{\partial \Delta(\mathcal{P}, c, f)}{\partial c(P)} &= \sum_{e \in E} -2M(P,e)\Big(f(e) - \sum_{P_i \in \mathcal{P}} M(P_i, e)c(P_i)\Big) \\ &= 2\sum_{e \in E} M(P,e)(M(P^*, e) - f(e)) \\ &= 2(|P \cap E_1| - |P \cap E_2| - |P \cap E'|)\end{aligned}$$

The last step in the above is due to $M(P^*, e) - f(e)$ being 1 for $e \in E_1$ and -1 otherwise. Note that $|P \cap E_1| = L-1$ by construction. As any simple path P either passes through $\mathcal{G}'$ or is equal to P^*, precisely one of the other terms is zero. If $P = P^*$, then $|P \cap E_2| = L-1$ and thus the derivative is zero. Otherwise, P passes through

$\mathcal{G}'$ and thus $|P \cap E'| \leq L - 1$. Hence, the derivative with respect to $c(P)$ is always non-negative. Since $c(P)$ cannot become smaller than zero, we conclude that $(\mathcal{P}, c)$ is a local minimum for the deviation function. Since both the constraints and the deviation function are convex, this local minimum must also be the global minimum.

For (2), assume that there exists a simple path P' in $\mathcal{G}'$ with length at least L. As above, let $P^* = E_1 \cup E_2$, and let $P = E_1 \cup P'$. We now show that there always exists a solution with deviation at most $|E|$. In particular, consider the (s, t)-reconstruction $(\mathcal{P}, c)$ with $\mathcal{P} = \{P^*, P\}$, $c(P^*) = 1$, and $c(P) = \epsilon$. Note that the edges in E_2 contribute exactly $|E_2|$ to the deviation. For an edge $e \in E_1$, the deviation is $(1 + \epsilon)^2$, and for an edge $e \in P'$ the deviation is $(1 - \epsilon)^2$. In total, the edges in E' contribute at most $|E'| - L + L(1 - \epsilon)^2$ to the deviation. We obtain the following total deviation:

$$\begin{aligned}
\Delta(\mathcal{P}, c, f) &\leq |E_2| + |E'| - L + L(1 - \epsilon)^2 + (L - 1)(1 + \epsilon)^2 \\
&= |E| - 2L + 1 + L(1 - \epsilon)^2 + (L - 1)(1 + \epsilon)^2 \\
&= |E| - 2L + 1 + 2(L - 1)(1 + \epsilon^2) + (1 - \epsilon)^2 \\
&= |E| - 1 + 2(L - 1)\epsilon^2 + (1 - \epsilon)^2 \\
&= |E| + (2L - 2)\epsilon^2 - 2\epsilon + \epsilon^2 \\
&= |E| + \epsilon((2L - 1)\epsilon - 2)
\end{aligned}$$

Finally, by choosing $\epsilon = 1/(2L - 1)$, we obtain that $\Delta(\mathcal{P}, c, f) = |E| - 1/(2L - 1) < |E|$, which concludes the proof. □

Consequently, we weaken the requirements even further and study relaxed (s, t)-reconstructions which may contain non-simple routes. We sketch a simple algorithm using positive parameter ε which, in the limit of ever smaller ε, approaches a relaxed (s, t)-reconstruction with optimal flow deviation: first, we find the min-cost flow that minimizes the squared difference or absolute difference between the input flow field and the flow [119]. Note that if we can construct routes that cover all flow of this min-cost flow solution, it must be optimal. We decompose the resulting flow of the min-cost flow computation into flows along simple-paths and flows along cycles [2]. The path flows are already valid reconstructed routes. However, we still need to cover that flow of the cycles. We can construct non-simple routes from the cycles by routing a shortest (s, t)-path via each cycle. By sending a very small ε amount of flow along these routes and spinning around the cycles many times, we can construct non-simple route flows that come arbitrarily close to the cycle flows they were constructed from. Hence, they also come arbitrarily close to the optimal deviation. The resulting routes are highly non-simple and generally not what we consider realistic behavior, thus we need the realism measure to enforce this.

We now return to the problem as stated in Section 4.1.4: we reintroduce the requirement that the reconstructed routes must be realistic, while still allowing non-simple routes and dropping the (s, t)-requirement. In this setting, minimizing the flow deviation becomes trivial if ε is large enough: when all individual edges are within ε of a representative trajectory, we can simply use them as routes in our reconstruction, trivially covering the entire flow field. If, on the other hand, ε is smaller than the smallest Fréchet distance from a representative to the road-network, we cannot reconstruct any routes and thus cannot cover any flow.

For values for ε between these extremes, as of yet we cannot establish the computational complexity. We observe that, already for simple paths, the problem under the absolute and squared deviations is always convex; the difficulty stems from the exponential number of candidate paths to consider for the reconstruction. Furthermore, by Carathéodory's Theorem, we know that the optimal reconstruction contains at most $|E|$ paths. Thus, the difficulty of the problem lies in efficiently searching through the solution space for the best routes. Allowing non-simple routes grows the solution space, but may potentially make it possible to search the space more efficiently. However, we see no reason why this would be the case if the routes must also be realistic. We therefore conjecture that the problem is also NP-hard for non-simple realistic reconstructions.

▸ 4.3 Route reconstruction algorithms

Here we present heuristic approaches that decompose the flow well while following the representative trajectories to varying degrees. In Section 4.3.1 we describe our iterative Fréchet Routes (FR) heuristics which creates only realistic routes, i.e., routes that have bounded Fréchet distance to a representative trajectory. We discuss two variants (WFR and EFR) which differ in their approach to constructing routes. In Section 4.3.2 we describe a multi-commodity min-cost flow approach (MCMCF) which is loosely coupled to the representative trajectories, and a global min-cost flow baseline (GMCF) which is essentially agnostic to the representative trajectories.

▸ 4.3.1 Fréchet Routes

Recall that our goal is to find a realistic basis $\mathcal{P}$ and coefficients c such that the flow deviation is minimized. Our Fréchet Routes (FR) heuristic decouples finding the basis and deciding on the coefficients c for a given basis. We grow the basis iteratively, aiming to improve the deviation after each iteration (see Figure 4.2). Since the solution space is infinite, the main challenge is to find a basis that is small

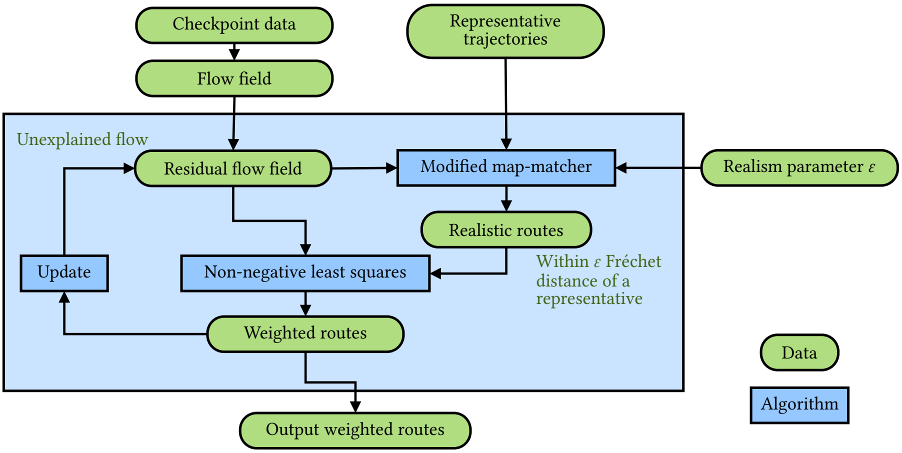


Figure 4.2 High-level overview of the proposed Fréchet Routes approach.

enough for efficient computation but comprehensive enough to result in a small deviation from the flow field.

In one iteration of our heuristic we add new routes to the basis for each representative trajectory independently ("Modified map-matcher") and evaluate the resulting basis. For a given basis we can compute the coefficients c that minimize the flow deviation efficiently with standard techniques [17, 71] ("Non-negative least squares"), resulting in a solution of weighted routes. We prune the basis by eliminating duplicate routes and routes with coefficient zero. Furthermore, we compute the *residual* flow field $\phi_r : E \longrightarrow \mathbb{R}$, defined as $\phi_r(e) = \phi(e) - \phi_{(\mathcal{P},c)}(e)$ for all edges $e \in E$. The residual flow field guides our search for new basis elements.

Generating basis routes Given the road network $\mathcal{G}$, a single representative trajectory T, the residual flow field ϕ_r (or the flow field ϕ in the first iteration), and threshold ε on the Fréchet distance, we generate basis routes for T as follows. The residual flow field stems from our current reconstruction $(\mathcal{P}, c)$. This reconstruction has an associated deviation $\Delta(\mathcal{P}, c, \phi)$ (this deviation is the flow within ϕ initially). If we extend the basis $\mathcal{P}$ with a path P and an associated positive coefficient c_P, the deviation changes by:

$$\begin{aligned}\Delta(\mathcal{P} \oplus P, c \oplus c_P, \phi) - \Delta(\mathcal{P}, c, \phi) &= \textstyle\sum_{e \in P} (\phi_r(e) - M(P, e) c_P)^2 - \phi_r(e)^2 \\ &= -c_P \textstyle\sum_{e \in P} M(P, e)(2\phi_r(e) - M(P, e) c_P)\end{aligned}$$

For routes that visit an edge at most once (e.g., simple routes), the above simplifies to $-c_P \sum_{e \in P}(2\phi_r(e) - c_P)$. Negative values reduce deviation, so we are particularly interested in capturing edges in routes which have high residual flow $\sum_{e \in P} \phi_r(e)$. Below we describe two different approaches to do so: Edge-inclusion Fréchet Routes (EFR), which selects k new routes per trajectory, and Weighted Fréchet Routes (WFR), which select one new route per trajectory. Both are adaptations of the Fréchet map-matching algorithm as described by Alt *et al.* [3], which we briefly describe first.

Fréchet map-matching Alt *et al.* [3] present an algorithm to map-match a trajectory $T = \langle p_1, \ldots, p_\ell \rangle$ to a road network $\mathcal{G}$: the result is a route in $\mathcal{G}$ such that its Fréchet distance to T is at most ε. This decision version can subsequently be used to find the route that minimizes the Fréchet distance to T, but we need only this decision algorithm.

Let the allowed Fréchet distance be fixed to ε, and let T be parameterized on $[1, \ell]$ such that for $\tau \in [i, i+1]$, $T(\tau) = p_i + (\tau - i)(p_{i+1} - p_i)$. Furthermore, we assume for simplicity here that $\mathcal{G}$ is the subset of the overall network that is fully covered by the Minkowski sum of a disk of radius ε with the trajectory T.

Trajectory T, network $\mathcal{G}$ and ε define a free-space manifold $\mathcal{F}$. This manifold is defined as the locations in the direct product space $(\tau, r) \in T \times \mathcal{G}$ such that $d(T(\tau), r) \leq \varepsilon$. A monotone path through the free space on this free-space manifold from some point with $\tau = 1$ to some point with $\tau = \ell$ then matches to a route in $\mathcal{G}$ with Fréchet distance at most ε. Note that the monotonicity avoids moving backwards along an edge, but an edge may be visited multiple times.

First, for each vertex v of $\mathcal{G}$, the algorithm computes a 1-dimensional free-space diagram $FD(v) : [1, \ell] \longrightarrow \{0, 1\}$. In this diagram, intervals that map to value 1 are called *white intervals*. The white intervals mark the parameter values of τ on T for which v is within Euclidean distance ε and thus a potential match.

Then, for each edge (u, v) in $\mathcal{G}$, the algorithm computes the left-right pointers for all white intervals I of $FD(u)$. These left-right pointers mark a range of $FD(v)$ that can be reached from I by a monotone path in the free space of the connecting 2-dimensional free-space strip defined by (u, v) and T. Due to convexity, any point in the free space of $FD(v)$ between these extremal left-right pointers can be reached, but note that this range may encompass multiple white intervals at v.

To determine whether a route exists within Fréchet distance ε, the algorithm now applies a sweep-line algorithm over the parameter space defined by τ, starting at 1. For every vertex v, it maintains a list $C(v)$ of ranges in $FD(v)$ that can be reached by a monotone path in the free-space manifold defined by $\mathcal{G}$ and T. That is, the intersection of $C(v)$ with the white intervals of $FD(v)$ define reachable intervals. In particular, this list contains the ranges such that the last part of the matching path ends at or goes through the sweep-line value in the dimension of τ. If $C(v)$ includes an interval that contains $t = \ell$, there must be a route within ε Fréchet distance.

The events that the sweep line handles are when the start of a reachable white interval I in some $C(u)$ is reached that was previously not discovered yet. This event is handled by updating all $C(v)$ for all neighboring v such that (u, v) in E, taking into account the new intervals that can be reached from I. These can be efficiently retrieved via the precomputed left-right pointers.

The sweep line stops as soon as either a white interval is added to some $C(v)$ that contains parameter value ℓ or no white intervals are available anymore. The former implies the existence of a route within Fréchet distance ε, whereas the latter implies that such a route does not exist. To reconstruct the route, the algorithm keeps track of predecessor vertices whenever the white intervals in $C(v)$ are updated when handling a white interval I. In total, the algorithm runs in $O(l(V + E))$ time, assuming efficient bookkeeping.

Edge-inclusion Fréchet Routes (EFR) We modify the map-matching algorithm to find a route that must include a specific edge $e = (u, v)$ with high residual flow. In fact, we are attempting to find k routes which include the k edges with the highest residual flow and that each have positive residual flow. To find a route within Fréchet distance ε of the representative trajectory T which contains edge e, we need to find two path in the free-space manifold $\mathcal{F}$: a path from the start to a white interval at u and a path from a white interval at v to the end. Moreover, the concatenation of these two paths with e needs to be monotone in $\mathcal{F}$. To do so, we consider each white interval at u, decide if it is reachable from the start, and if so, continue from all possible white intervals at v.

EFR stops its search in the free-space manifold as soon as the begin/endpoint of the reconstructed route lies within ε Euclidean distance of the start/end of T. This might ignore flow on edges in the ε-vicinity of the start and end of T. Hence, we greedily add suitable edges to the ends of the route, taking care not to introduce cycles and to not decrease the average amount of residual flow per edge in the route.

Weighted Fréchet Routes (WFR) EFR uses the residual flow only to indicate the top k routes. We further modify the map-matching algorithm to find routes that generally include edges with high residual flow, that is, high weight. To do so, we maintain a sorted list of weights with each white interval. Let τ be the parameter of trajectory T (recall that these are the height values of the free-space manifold $\mathcal{F}$) and consider an interval I for vertex v. Each entry in the list for I is a tuple (τ, ψ) such that there is a monotone path through $\mathcal{F}$ with weight at least ψ ending in v with height at least τ. Note that the value of ψ depends on the execution order of Dijkstra's algorithm and is hence only a lower bound. We prune tuples (τ, ψ) for I whenever there is another tuple (τ', ψ') in the list where $\tau' < \tau$ and $\psi' \geq \psi$. We maintain the weights as we execute the map-matching algorithm. In principle, we can construct a high-weight route at the end of the algorithm. However, note that the road network $\mathcal{G}$ can contain cycles which result in cycles between white intervals. Hence we construct a high-weight route explicitly via back-tracking, using each tuple at most once. Note that the resulting route may still contain cycles in the road-network; we only break cycles in the dependency of white intervals.

▸ 4.3.2 Min-cost flow

We now describe two heuristics that are based on min-cost flow and relax the realism constraint of Fréchet Routes. On a high level, both heuristics follow the same approach: we first solve the min-cost flow problem for the flow field (guided by the representatives to a certain degree) using our flow deviation as cost function, and then we heuristically compute a reconstruction from the resulting flow.

Multi-commodity min-cost flow (MCMCF) For each representative trajectory T we construct a subgraph $G(T)$ of $\mathcal{G}$ with all vertices and edges within distance ε of T. Vertices within distance ε from the start or end of T can act as sources or sinks of a flow in $G(T)$. Each representative trajectory hence induces a single (min-cost) flow problem. By overlapping the graphs $G(T)$ for all $T \in \mathcal{T}$, we construct a *multi-commodity min-cost flow* problem on $\mathcal{G}$, where each trajectory T has an associated commodity. We can solve the resulting MCMCF using standard software packages (see Section 4.4).

Global min-cost flow (GMCF) We retain only the sources and sinks of MCMCF and otherwise impose no restriction on the flow. This results in a min-cost flow problem over the entire road network $\mathcal{G}$, which is essentially agnostic to the representative trajectories.

Heuristic path reconstruction The result of either min-cost flow approach is an edge flow per commodity or over the complete road network. Our goal is to approximate these flows via a reconstruction that may use non-simple paths. For each commodity (or the complete network) we first compute a “path flows-cycle flows” decomposition [2], which is equivalent to the edge flows. We can directly add the resulting path flows to our basis. The cycle flows, however, generally are not correct source-sink paths. We observe that cycle flows which are disjoint from all path flows cannot be close to any of the representative trajectories; we hence exclude them from the basis. For each other cycle flow, we greedily merge it with one of the path-flows that overlap it at one or more vertices. If the path flow is higher than the cycle flow, then we reduce it to be equal to the cycle flow. If the path flow is lower than the cycle flow, then we traverse the cycle multiple times, using the path flow, to create a consistent (non-simple) route, rounding where necessary.

▸ 4.4 Experimental evaluation

We evaluate and compare the various heuristics of the previous section using real-world trajectories. Particularly, we investigate the extensions and parameters of our Fréchet-Routes methods and compare Fréchet Routes to the min-cost-flow-based methods.

We implemented all algorithms[1] in C++ using Boost and MoveTK[2]. For flow problems and determining coefficients c, we use IBM ILOG CPLEX 12.9. We ran all experiments single-threaded on Ubuntu 18.04, on an Intel(R) Xeon(R) Gold 5118 CPU @ 2.30GHz.

Data To evaluate how well representative trajectories assist in route reconstruction beyond the given sample, we require a ground truth: all driven routes $\mathcal{P}^*$ that together define the flow field ϕ. We use as network $\mathcal{G}$ the roads surrounding The Hague (the Netherlands) extracted from OpenStreetMap [97]; see Fig. 4.3, the left figure. As the complete trajectory set $\mathcal{T}^*$, we use 11 445 real-world trajectories provided by HERE Technologies[3] in the same area. We map-match $\mathcal{T}^*$ to $\mathcal{G}$ to obtain $\mathcal{P}^*$. To avoid bias, we use the approach by Yang and Gidofalvi [131] instead of the approach by Alt*et al.* [3], as the latter relies on the Fréchet distance and is the basis for our Fréchet Routes. We derive the flow field ϕ for $\mathcal{G}$ by counting the number of occurrences of each edge in $\mathcal{P}^*$. The representative trajectories $\mathcal{T}$ are sampled from $\mathcal{T}^*$, using $\alpha > 0$ such that $|\mathcal{T}| = \lceil \alpha |\mathcal{T}^*| \rceil$.

---

[1] Source available at https://github.com/tue-alga/RouteReconstruction

[2] https://movetk.win.tue.nl

[3] https://www.here.com

Measures To evaluate our heuristics, we use the measures listed below that are generally based on Section 4.1. Whereas some measures can be used to assess any result, the two measures marked with "GT" rely on having a ground truth. That is, the latter type can measure performance beyond the provided representatives.

flow deviation: how well do the reconstructed routes represent the input flow? We measure the sum of squared differences between input and reconstructed flow, over all edges.

realism: how realistic is the reconstruction? We average the Fréchet distance from each reconstructed route to its closest representative trajectory in $\mathcal{T}$, weighted by the coefficients.

realism (GT): how realistic are the reconstructed routes in relation to the ground truth? This is identical to realism, but we use the closest trajectory in the $\mathcal{T}^*$ data set.

completeness (GT): how well is the ground truth captured by the result? We average the Fréchet distance from each route in the ground truth $\mathcal{P}^*$ to the closest reconstructed route.

cardinality: the number of reconstructed routes.

running time: total computation time (wall clock).

For all measures, lower values indicate better performance. We note that finding a subset of $\mathcal{T}$ with optimal completeness is NP-hard, via a reduction from dominating set for unit-disk graphs [86]. Generally, the data does not admit a solution with perfect completeness or realism, as the trajectories are not aligned to the road network. To indicate the distortion inherent in the data due to map matching, we visualize the Fréchet distance between $\mathcal{T}^*$ and $\mathcal{P}^*$ in Fig. 4.3 on the right side.

▸ 4.4.1 Fréchet Routes

Here we investigate our Fréchet-Routes algorithm, in terms of candidate-generation methods and its parameters. Throughout this section, we keep ε fixed at $100m$ and run each trial with seven random samples and average the results. Throughout this investigation, we do not consider our realism measures, since FR and its extensions guarantee realism by construction.

Candidate generation We aim to investigate EFR and WEFR to generate basis routes. Specifically, we run four variants: with weighted routes (WFR), with

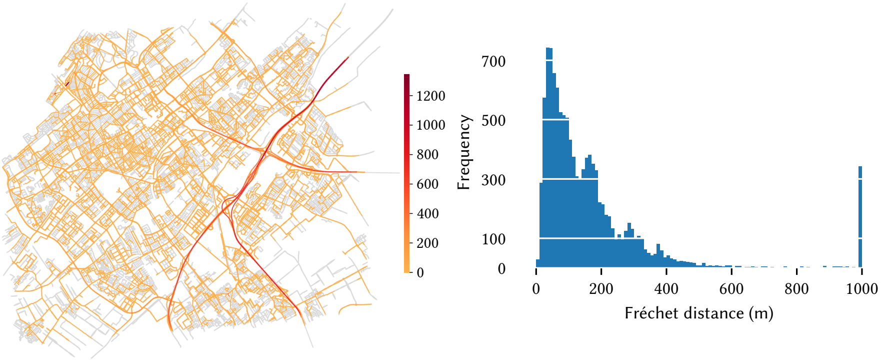


Figure 4.3 (left) Road network of The Hague used in our experiments, 60 277 vertices and 100 654 edges, with color indicating the flow field induced by $\mathcal{P}^*$; gray edges do not contain flow. (right) Histogram of Fréchet distances between all $T \in \mathcal{T}^*$ and their routes in $\mathcal{P}^*$. The rightmost bar aggregates higher values, which is approximately 3% of the trajectories.

edge-inclusion (EFR), with both (WEFR) and without either extension (FR). We use $\alpha = 0.05$, $i_{\max} = 8$ and $k = 2$. Fig. 4.4 summarizes the results. Compared to FR, the extensions have a mild positive effect on completeness and a strong positive effect on deviation, but increase cardinality. Whereas EFR is slightly worse in deviation, completeness and running time than WFR, it performs better in cardinality. Interestingly, WEFR seems to effectively combine these, resulting in deviation and complexity between WFR and EFR and completeness is actually slightly better than either variant in isolation, suggesting a complementary nature here in obtaining large variation (WFR) and trying to address specific edges with deviation (EFR). The drawback is the increase in running time, since we effectively run both methods together.

Iterations and edge inclusion For WEFR we investigate the effect of $i_{\max}$, the number of iterations, and k, the number of paths generated by edge inclusion. We run our algorithm with $\alpha = 0.05$ and $k \in \{2, 10\}$ for $i_{\max} = 8$ iterations, recording the result after each iteration. Fig. 4.5 illustrates the results. The time spent per iteration remains roughly similar, even though cardinality tends to increase. Iterating has a very mild positive effect on completeness and flow deviation. Cardinality increases quickly in the first iterations, but the later ones do not necessarily increase

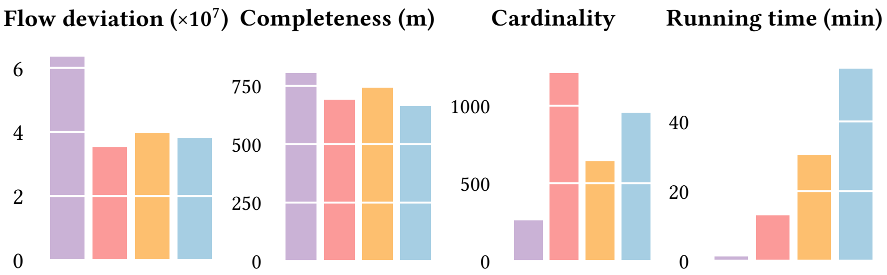


Figure 4.4 Results using variants of Fréchet Routes, from left to right: FR (purple), WFR (red), EFR (orange), WEFR (blue).

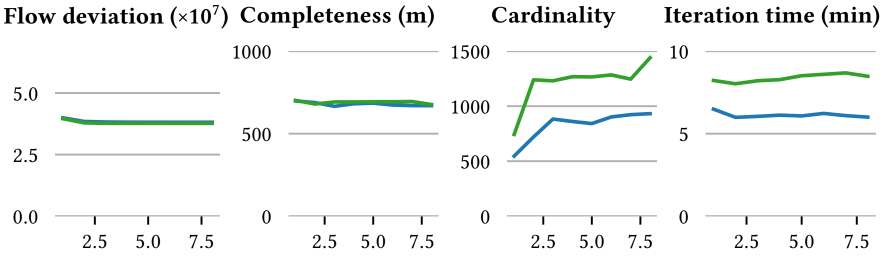


Figure 4.5 Results for WEFR per iteration, with $k = 2$ (blue) and $k = 10$ (green).

cardinality, as alternative routes are generated that can replace earlier candidates. Compared to $k = 2$, $k = 10$ yields higher cardinality and running time, but improves deviation. The somewhat unstable completeness can be attributed to the incomplete knowledge about the ground truth: routes can improve deviation while being further away from unknown ground truth routes.

▸ 4.4.2 Comparing different methods

We now compare WEFR to the MCF-based (min-cost-flow) methods: GMCF and MCMCF. We look into the effect of varying the sampling rate for the approaches, and then provide an in-depth analysis of the structure of the solutions provided by the approaches.

Sampling rate We vary α using values in $\{0.05, 0.1, 0.15, 0.2, 0.25\}$. This mimics different degrees of coverage of the representative trajectories relative to the overall

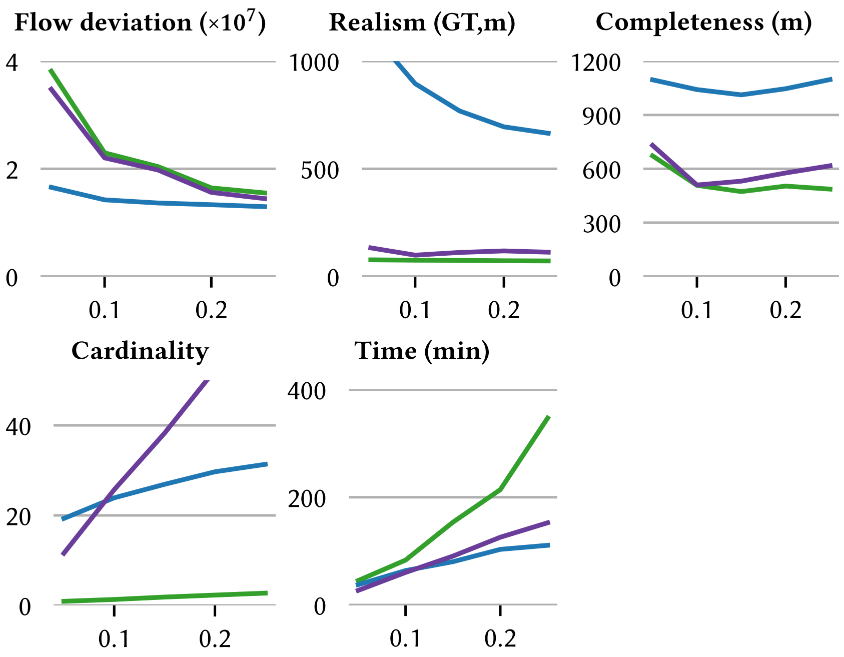


Figure 4.6 Results of WEFR (green), GMCF (blue) and MCMCF (purple) for varying sampling rate α.

traffic. Generally, we may expect that solution quality increases as we have more representative trajectories. Based on the analysis of the FR approaches, we use $k = 2$ but reduce $i_{\max}$ to 5 for WEFR, since a larger sample implies that more candidates are generated per iteration. We show the results for WEFR, GMCF and MCMCF in Fig. 4.6. As expected, increasing sampling rate yields better deviation and completeness at the expense of longer computation times and higher cardinality for all approaches. We observe that completeness seems to not vary much for α between 0.1 and 0.25 for WEFR: there may be outlying behavior in the full dataset which is not readily captured by other traffic, but this may also be in part due to the inherent error in the ground truth (see Fig 4.3, right). Interestingly, at $\alpha = 0.25$, the deviation for both WEFR and MCMCF is already close to the deviation of GMCF, the least restrictive approach. Both approaches dominate GMCF in terms of realism and completeness, showing that they are able to balance realism, completeness and deviation better for higher sampling rates.

Table 4.1 Deviation and cardinality for the three different methods on the full solution, as well as on subsets thereof.

		WEFR	MCMCF	GMCF
Full	Deviation ($\times 10^7$)	2.13	1.95	1.07
	Cardinality	926	11553	19336
	Running time (min)	46.6	31.1	43.5
Restricted	Deviation ($\times 10^7$)	2.13	3.35	15.7
	Cardinality	465	638	28
Adjusted	Deviation ($\times 10^7$)	2.13	2.12	14.1
	Cardinality	465	638	28
Reduced	Deviation ($\times 10^7$)	2.13	2.16	
	Cardinality	465	465	

Analyzing the structure of the solutions We now analyze the structure of the different solutions that are produced by our approaches. We use $\epsilon = 100$m and $\alpha = 0.05$ to get representative trajectories. We repeated the analysis below for seven random samples, but the outcome was similar every time; we hence focus our exposition on one such random sample here.

Table 4.1 lists measures of the (*full*) solution for each of the three methods. MCMCF and WEFR are similar in deviation, with MCMCF performing about 10% better than WEFR. Considering cardinality, WEFR achieves this deviation with but a fraction of the routes that MCMCF uses. Interestingly, we see that MCMCF uses roughly the same (but slightly more) routes than in the ground truth. The question is though, how realistic all such routes are. GMCF achieves the lowest deviation, but is also least constrained in terms of realism. Running times are in the same order of magnitude, with WEFR being slowest and MCMCF being fastest.

In Fig. 4.7 we relate the coefficient of each reconstructed route to its realism (GT); we use the ground-truth variant which gives a slight advantage to the MCF-based methods, as WEFR constructs realistic routes by definition. We observe that the MCF-based methods include many unrealistic routes, even when measuring with respect to $\mathcal{T}^*$. Also, there are one or two routes with considerable coefficient, and many routes with low coefficients for this dataset.

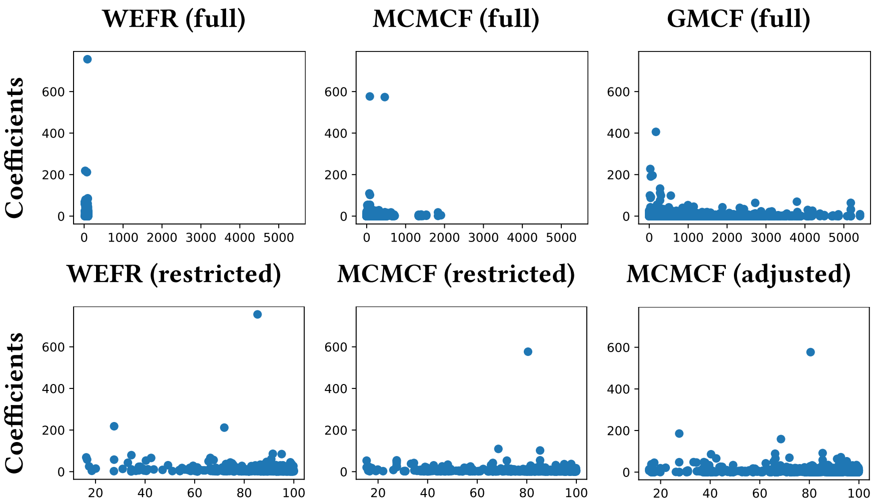


Figure 4.7 Relation between realism (GT) in meters on the horizontal axes and the coefficient of each reconstructed route in the solutions of WEFR, MCMCF and GMCF. Here $\epsilon = 100\text{m}$ and $\alpha = 0.05$. (Top) Routes in the full solution. (Bottom) Routes for WEFR and MCMCF after restricting the routes and for MCMCF also after adjusting coefficients.

We thus restrict the solutions. First, we filter to include only reconstructed routes that are deemed realistic with respect to the ground truth, using a threshold of 100m. Motivated by the very high cardinality of the MCF solutions, we also filter to include only routes that have a coefficient of at least 1: that is, we focus on those routes that actually contribute to the flow deviation minimization.

Table 4.1 lists statistics for these *restricted* solutions. WEFR retains roughly half of its routes (only based on the coefficient). Yet, the deviation is hardly affected. For MCMCF, the cardinality is reduced to roughly 5%, though this increases deviation to be over 50% worse than WEFR. This suggests that MCMCF produces fairly realistic routes, among many unrealistic ones, yet they are not immediately selected to represent the flow field. GMCF reduces to few routes, with excessive deviation: it does not explain traffic flow well with realistic routes and we do not consider it further.

Fig. 4.7 illustrates also the coefficient-realism relation for WEFR and MCMCF for the restricted solutions. Here, it is evident that WEFR and MCMCF have somewhat similar structure, whereas nearly all routes in the GMCF disappear. Of note is that one of the two high-coefficient routes was filtered out for MCMCF.

By removing routes from the solution, the computed coefficients are no longer optimal for flow deviation. For better understanding the solution, we adjust the coefficients by recomputing them for the restricted set. For these *adjusted* (and restricted) solutions, MCMCF recovers its loss in deviation to be comparable with WEFR; GMCF barely recovers (see Table 4.1). In Fig. 4.7 we see that the solution structure resembles the WEFR situation even more.

To assess to what degree the higher cardinality of MCMCF influences the above, we reduce its solution further: we select the 465 highest-coefficient routes, and again adjust the coefficients. For this *reduced* set, we see that the deviation is increased again slightly, to be only slightly worse than WEFR (Table 4.1). This reinforces the conclusion that MCMCF is able to find realistic routes, but that these are hidden between the unrealistic ones. Though filtering as above is possible, it provides computational overhead compared to using WEFR which directly guarantees realism.

▸ 4.5 Case study

We present a case study using real-world loop-detector data, obtained from the Dutch National Traffic Dataportal[4]. Using this loop-detector data and the full trajectory set $\mathcal{T}^*$ described in the previous section, we apply the different approaches to reconstruct routes from these loop detectors. Fig. 4.8 illustrates the locations of the loop detectors and their amount of flow for a single hour during the day the trajectory dataset was recorded. To acquire a (partial) flow field out of the detector data, we assign their flow to the nearest edge in the road network, selecting the maximum flow when multiple detectors match to the same edge.

Incomplete flow As Fig. 4.8 illustrates, the loop detectors do not cover all edges in the road-network and it hence gives an incomplete flow field. Hence, we adapt the deviation measure to include only edges that have a loop detector and we adapt the non-negative least squares accordingly. This does not fundamentally alter the described approaches. One consequence of this approach is that there are (sub)routes in the network that do not affect flow deviation, and thus may lead to unnatural routes which include small cycles and detours. EFR does not suffer from this problem as the underlying algorithm [3] tends to create short paths. For WFR we discourage the use of edges without a loop detector by giving them a small negative residual

---

[4] `https://ndw.nu/en/`

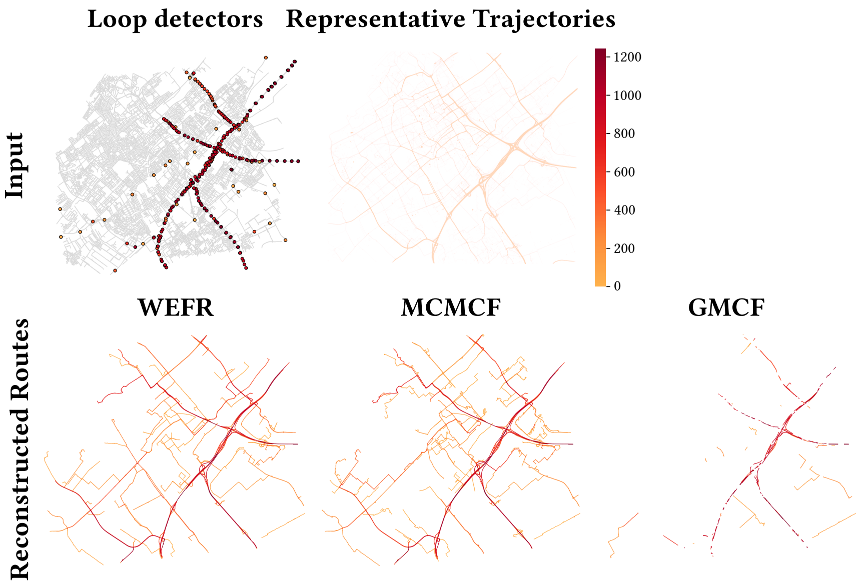


Figure 4.8 Case study: (Top) real-world loop-detector measurements and representative trajectories. (Bottom) Results of WEFR, MCMCF, and GMCF, visualized as the induced flow field derived from the routes and their coefficients. Here, again $\epsilon = 100m$.

Table 4.2 Quality measures for the case study.

	WEFR	MCMCF	GMCF
Deviation ($\times 10^7$)	3.89	4.03	3.07
Cardinality	109	646	271
Realism (m)	82.4	98.2	503.7
Running time (min)	153.2	26.9	1.3

value. For MCMCF one could consider to also include some form of minimization of the flow on edges not having a loop detector; however, preliminary tests suggest that this is not effective; we leave improving realism for MCMCF on incomplete flows to future work. Finally, we do not modify GMCF as it does not exhibit this problem and, as we shall see, such an approach may even exacerbate the issues of this method.

Analysis Fig. 4.8 shows the reconstructed routes as the derived flow field for all algorithms, and Table 4.2 shows applicable quality measures. It is clear in Fig. 4.8 that GMCF very greedily constructs routes: the routes are mostly short disjoint pieces that cover loop detector flow. For WEFR and MCMCF we see longer routes, extending beyond the immediate vicinity of the loop detectors: this we can attribute to the stricter realism requirements which cause *entire* (mostly) realistic routes to be selected. This is further supported by the deviation and realism of the different approaches (Table 4.2): GMCF performs best regarding deviation, but at the expense of low realism. WEFR and MCMCF perform better here, where WEFR balances the realism and deviation best of the two. This can be attributed to the more targeted search of WEFR for high-weight routes.

Contrasting our previous experiments with complete flows, WEFR now actually achieves better flow deviation than MCMCF, in spite of MCMCF having higher cardinality. The drawback is that WEFR scales less well with the higher number of representative trajectories, taking about five times as long as MCMCF – but reducing the number of iterations may partially alleviate this drawback.

▸ 4.6 Discussion

Our work shows that the studied problem is challenging even in "simple" forms, but our evaluation shows promising results. As such, it leads to various avenues for further research.

Representatives Our experiments show that more representatives can improve the performance of route reconstruction. Yet, this also comes at a computational cost. However, this is mostly as it diversifies behavior: adding very similar representatives will not improve the results drastically, if at all. As our methods do not require that the representatives are actually part of the traffic generating the flow data, we could preprocess representatives using clustering and central trajectories to reduce computation time of our algorithms. Furthermore, we could use the information from such clustering methods, or from map-matching accuracy, to vary the threshold ε per representative, to relate realism to the uncertainty in the data. We leave to future work to investigate how such techniques affect efficiency and quality.

Reconstruction MCMCF does not guarantee a bound on the Fréchet distance, and we observed that its very large basis contains many routes that exceeded the given realism parameter. We showed that we can post-process the routes to include only realistic ones to overcome this issue, but this increases computation time and flow deviation. An interesting direction of research could be to incorporate this directly into the route-reconstruction phase, to avoid or at least reduce this overhead.

As demonstrated, our techniques can be adapted to handle cases where not all edges have associated flow. We observed a need to incorporate information on the roads without flow data, to avoid unnatural cycles and detours in the reconstructed routes. Requiring strict simplicity may eliminate cycles, yet it is unlikely to fully address the problem and makes the problem significantly harder. Hence, we may want to consider other models to further restrict what defines a realistic route, beyond the Fréchet distance used here.

Time-varying flow data We considered only flow data that is static: an edge has a single value associated with it (if any) representing the amount of traffic in a certain time interval. However, traffic changes over time and hence flow data is time-varying. As we discussed in Section 4.1, the time-varying nature of the data poses several challenges that need to be overcome to reconstruct routes in this setting. We briefly sketch some ideas for future work that may tackle some of the challenges.

A first challenge that arises is that we should incorporate time into the reconstructed routes. To be able to determine what a reasonable route can be, we need to model the time it takes to traverse individual edges of the road network. A relatively straightforward approach is to add fixed travel times to each edge, which dictate how long an entity takes to traverse the edge. If we model these travel times as integral amounts, we can employ a *time-expanded network* [106][5]. This networks contains for each time unit a replication of the vertices of the road-network, a time layer. Vertices between layers are connected according to the travel times for the associated edges in the original network. Note that, depending on the time resolution, this may be a very memory intensive data structure. Selecting a sequence of connected edges in the time-expanded network now gives us time-dependent routes.

A second challenge is how to handle the time-varying checkpoint data. For checkpoint data, the number of vehicles passing the measurement location is often aggregated over fixed time intervals. Using this notion, we can define our flow deviation on the measurements of the individual time intervals, minimizing the squared difference between the measurement and the number of routes through the edge in question, at the specified time interval. If the time intervals are as small as the time unit for the time-expanded network, we can assign the measurements directly to

---

[5] This choice amounts to picking the so-called *discrete time* model for our time component.

edges in the time-expanded network, resulting in a setup very similar to the time-independent case. If, however, the time intervals are not exactly our time units, we need to take into account the number of routes over multiple edges in the time-expanded graph. Due to this coupling, the min-cost-flow-based methods are not applicable anymore.

A third challenge is picking the appropriate way of measuring realism, and adapting our reconstructed routes accordingly. If we simply assume that the route of the trajectories is what makes them representative, then polyline distance measures, such as the Fréchet distance that we use, can still be applied. Extending the Fréchet map-matching approach to a time-expanded network seems relatively straightforward. If, on the other hand, we also assume that the dynamics of the representative are indicative for the local traffic, then we need new distance measures to capture this requirement. The particular choice then will greatly affect how to generate any routes to incorporate into the solution.

Chapter 5

Coordinated Schematization for Visualizing Mobility Patterns on Networks

One key use for large data sets of trajectory data is to analyze and understand patterns of the moving objects. In the context of human trajectory data, these mobility patterns provide insight for traffic analysis and urban planning. Visualization of this trajectory data then makes it possible for experts to properly analyze the data. But, as noted before, the sheer volume of the data makes it challenging to render trajectory collections in a meaningful way as to show the general, overarching patterns. Simply plotting all trajectories results in the infamous “spaghetti heaps”. Heat maps [77, 104] and other aggregation techniques such as Voronoi aggregation [7] are helpful to “untangle” traffic locally, but they generally fail to capture structural patterns, such as important longer routes.

Summarizing trajectory collections visually, such that salient patterns emerge, inherently requires a form of aggregation or simplification of the data. That is, the level of detail and information shown should be scale-appropriate and avoid a cognitive overload, while still being able to provide insight into the overall mobility. There are various techniques to cluster trajectories and compute a representative for visualization [20, 79], or to simplify trajectories [67]. However, these techniques typically focus on trajectories in a general 2D space, whereas our focus lies on trajectory data on transportation networks, specifically vehicles on a road network. As such, the problem changes in nature, as selecting representative routes and performing simplification needs to be "network-aware". Stronger still, to reduce the visual complexity of the eventual visualization, not only the trajectories need to be simplified, but also the underlying street network. To arrive at meaningful results which show traffic patterns in the correct context, the simplification and aggregation of the trajectory collection and of the network have to go hand-in-hand: they need to be *coordinated*.

Contribution and organization We propose a coordinated fully-automated pipeline for computing a schematic overview of mobility patterns. Our pipeline consists of five steps, each utilizing well-known building blocks from GIS, automated cartography and trajectory analysis: map matching, road selection, schematization, movement patterns, and metro-map style rendering. We present our overall pipeline and its rationale in Section 5.1; the subsequent sections describe each step in more detail. Each of these sections also describes how we implemented the corresponding step in our proof-of-concept. For illustration we use a real-world dataset of vehicle trajectories around The Hague in the Netherlands. In Section 5.7 we discuss the results of our pipeline using a second real-world data set around Beijing. We close with a general discussion of our pipeline and future work in Section 5.8.

Related work We focus here on related work pertaining to the visualization of large volumes of trajectories and discuss related work for each step of the pipeline in the respective section. Most research in this area aims to provide an overview of space usage, without showing or using the temporal component of trajectories; see the two extensive surveys by Chen *et al.* [34] and Andrienko *et al.* [5]. The notable exception are space-time cubes [73, 110], though they do not scale well to large numbers of trajectories without some form of aggregation.

To identify larger patterns, one can focus on visualizing the origin-destination data only, that is, focus only on the endpoints of the trajectories, possibly with some form of spatial aggregation. There are various techniques to visualize such information, e.g., [113, 126, 132]. Visualizing OD-data shows patterns beyond local traffic, but typically does not show any information on the actual routes. As such it does not support understanding mobility from the viewpoint of traveling through a network. Indeed, these techniques are typically applied in situations where the exact trajectories or routes are not available or not of interest.

▸ 5.1 The pipeline

Our input is a set $\mathcal{T}$ of trajectories, and a network G. Our goal is a schematized representation of G together with the most salient mobility patterns in $\mathcal{T}$. Before we can describe our pipeline in more detail, we first give the necessary definitions.

Definitions Recall that a road network is a directed, embedded graph in $\mathbb{R}^2$. For our pipeline, we assume that the network is planar, that is, the edges do not cross except at common endpoints. This condition is not generally satisfied by actual road networks, but we can introduce extra vertices on these intersections to planarize the network. By marking these extra vertices, we can perform algorithms both on the planarized and original network. We further use $|G|$ to denote the complexity of the network, measured as its number of edges, though note that this assumption implies that $|G| = |E| = O(|V|)$.

We use *route* to refer to a (directed) path in the network. For example, map-matching a trajectory results in its route: the path in the network that the entity traverses as captured by the trajectory. We refer to such a path as *the* route of the trajectory.

A route does not have to correspond to a single or entire trajectory. Specifically, we say that a route is *supported by* a trajectory T if it is a subpath of the route of T. We use *bundle* to indicate a route that is supported by multiple trajectories, using the *support of a bundle* to indicate the number of supporting trajectories. Bundles should aim to capture mobility patterns; precise criteria to form meaningful bundles are discussed in Section 5.5.

The pipeline Our goal is to compute a schematic representation of G with salient bundles that are “supported by” $\mathcal{T}$. To achieve this, our pipeline consists of five steps, briefly sketched below. See Fig. 5.1 for an example of the results of these steps. In the subsequent sections we discuss each in more detail. Important in our treatment of the network and the trajectory information is to coordinate changes: changes in the network should be translated to changes in the trajectory information.

Step 1: Map-match We aim to visualize mobility patterns via bundles, frequent routes in the data. However, trajectories are not the same as routes, and thus cannot support a bundle. As such, we first map-match the trajectories to the network. The minimal input to this step is a single trajectory and the network, such that each trajectory can be processed individually. Map-matching computes the route associated with this trajectory. The result of this step is a set

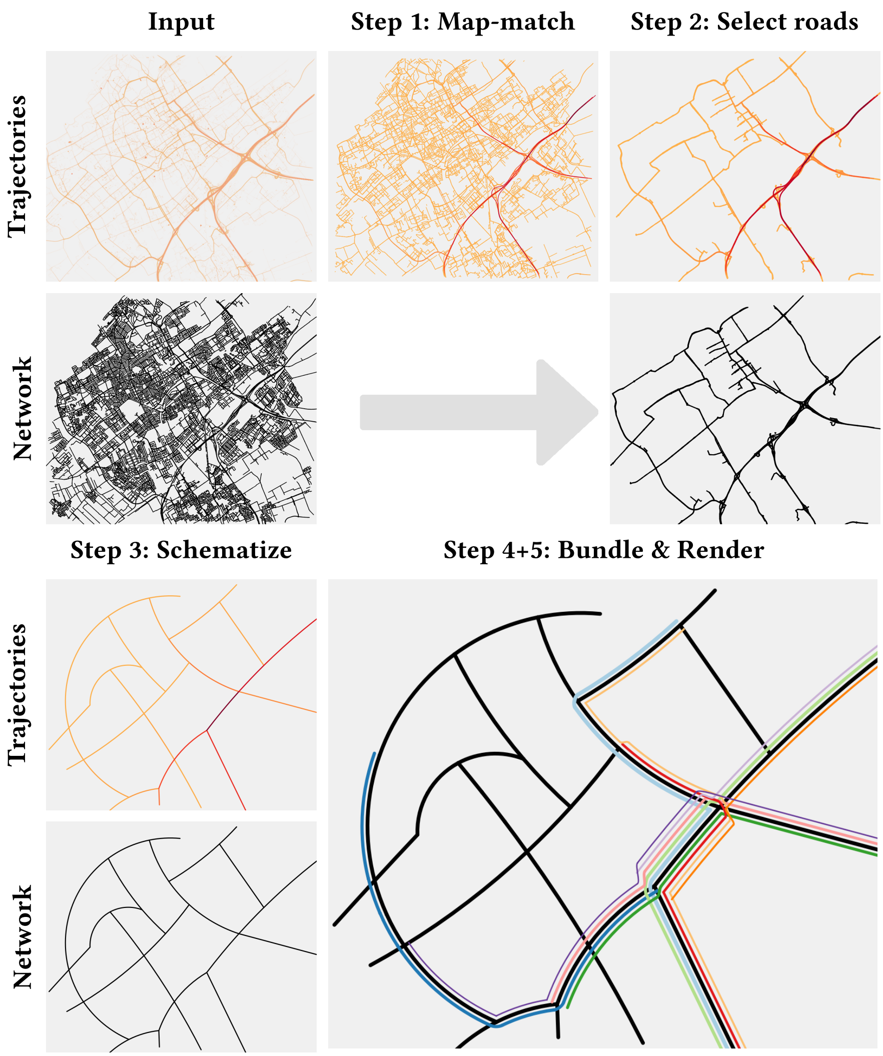


Figure 5.1 Our pipeline for coordinated schematization on the The Hague dataset. The input trajectories are shown as a density map. For the map-matched routes, we use a orange to red scale to convey low to high traffic volume per edge. We compute the bundles in Step 4 with $S_{min} = 500$, $L_{min} = 10000m$, $p = 0.5$ and shrunk edge lengths, see Section 5.5 for more details on these parameters.

$\mathcal{R}$ of routes, rather than trajectories. In our implementation, the trajectories are not used further. This step enables our pipeline to coordinate changes in the network and the routes.

Step 2: Select roads A typical road network is very detailed, much beyond the level of detail that we need to visualize the general mobility patterns and well-supported bundles. The minimal input to this step is the road network and the set of routes derived in Step 1. Note that we could, in principle, base selection purely on the network, and adapt the routes as necessary. However, we also choose to include all parts of the network which are frequented heavily by the routes. The result is a subset of the network and a mapping of the original routes to this selected network.

Step 3: Schematize To reinforce the summarizing nature of the eventual visualization, we reduce the visual complexity of the selected network via schematization. The input is the selected network and the mapping of the routes. The output is a strongly simplified version of this network. The mapping of the routes is maintained (coordinated) during this process. Optionally, the edges of this schematic network may be annotated with information about the length of the edge for the purpose of bundling.

Step 4: Bundle In our schematic representation, we find well-supported bundles. The input is the schematic network and the mapping of routes to this network. The output is a set of bundles that are well supported.

Step 5: Render We now have all ingredients for our visualization: the schematic network as well as salient mobility patterns (bundles). The result is the eventual visualization which shows these two pieces of information effectively.

▸ 5.2 Step 1: Map-match trajectories to the network

Desiderata To eventually visualize common routes in the network, we must ensure that our trajectory information is mapped to the network, that is, that each trajectory is translated into a route. This is the map matching problem.

Related work Map matching is a broadly studied topic in GIS; see [31] for a recent survey and Chapter 3 for more details. As discussed in the survey, the different approaches are categorized by their matching model. For our purpose, we consider a map-matching approach using the similarity model, since this requires less parameters. But in principle, any approach will work in our pipeline.

Our implementation We use the map-matching algorithm described by Alt *et al.* [3], which runs in $O(|G||T| \log |G||T| \log |G|)$ time on a network G and trajectory T. The algorithm ensures that the route found has minimal Fréchet distance to the trajectory and thus that it is geometrically close. Our motivation for doing so is to remain as close as possible to the information of the trajectory. That is, to the information "visible" if we were to simply draw all trajectories. This algorithm is able to handle noise relatively well, but in case of very sparsely sampled trajectory data, it may struggle to find the most natural route.

5.3 Step 2: Select roads

Desiderata We aim to select the roads for two somewhat distinct purposes. First, we want to select the roads where there is considerable traffic, to facilitate well-supported bundles. This purpose is thus inherently data-driven. But second, we want to select major roads to provide a frame of reference for the viewer as to how the mobility patterns are situated in space. It stands to reason that often, major roads also carry a large part of the traffic. However, this is not necessarily the case.

Related work Selection is an important part of road network generalization algorithms. The goal is to select the most important parts of the network, such that the remainder can be discarded in the simplification process. Different approaches exist to determine what features of the road-network are "salient". Examples of these approaches are using the mesh density [33], using user defined weights [76] and using areas of faces combined with semantic labels [108]. Different from the previous are approaches that focus on "strokes" through the network: lines of good continuation, that is, lines with small local curvature [109]. During generalization, these strokes

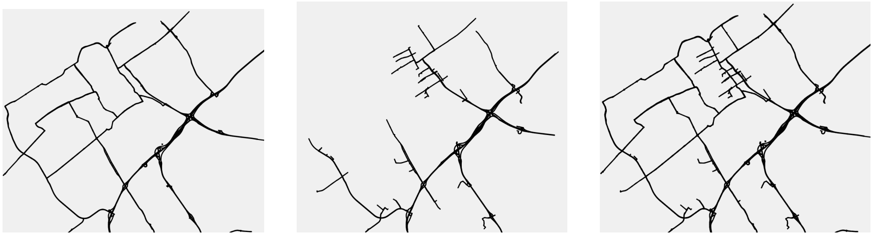


Figure 5.2 Selection by road type (left) and by traffic (middle), combination of both (right).

are considered atomic units and are selected based on their relative importance. This importance is often determined via network centrality measures [121, 130].

More recently, approaches focus on using traffic data to inform the selection process [112, 135]. Yu *et al.* propose an approach that is based on strokes, but during the selection process considers traffic flow from one stroke to the other, increasing the likelihood that strokes that give good traffic flow continuation are selected together. Van de Kerkhof *et al.* [112] follow a different approach, where the selection process is formulated as a covering problem, and the trajectories need to be covered by the selection of the road-network.

Our implementation We use the approach by van de Kerkhof *et al.* [112] which seeks to select a subgraph G' of the network with bounded length, such that the number of routes that are completely within this subgraph is maximized. Though the authors prove that this problem is NP-hard, they also describe a heuristic that runs in $O(|\mathcal{T}|^2 \log |\mathcal{T}| + |\mathcal{T}||G|)$ time; we use this heuristic in our implementation (see Fig. 5.2 (middle)). After selecting G', we add any edges that were not selected yet and have a large enough road type. If the resulting selected network is not connected, we optionally select the largest connected component; none of the later steps require the network to be connected (see Fig. 5.2 (left) and (right)).

5.4 Step 3: Schematize the network

Desiderata Even after selection, the network tends to contain more detail than necessary to provide a meaningful overview of mobility patterns: different lanes, cloverleaves, etc. Instead, we should get a high-level overview that communicates the main connectivity in the network, and as such create space to visualize mobility patterns (bundles). That is, we should collapse (aggregate) and simplify such local details. We do so beyond the need of target scale, instead focusing on *functional* detail: that is, we schematize the network. The network should remain spatially informative: roughly similar to the overall input geometry.

Related work In automated cartography, the process of schematization is used to render aesthetically pleasing networks or polygonal domains that are decluttered enough to convey important information on the schematic [87]. Compared to generalization, schematization commonly reduces the input to such an extent that it is not necessarily realistic anymore, albeit retaining important features to recognize the original. Note that in general, the input to these algorithms is a detailed map of the road-network, whereas we input a selection of the map based on data, thus making it a data-driven schematization.

One approach is to limit the type of the geometry in the output, thus naturally reducing detail. Here it is common to fix the allowed number of orientations of lines in the network [24, 124], particularly in the context of metro maps. Alternatively one can fix the geometric primitives that can be used, for instance circular arcs [114] or Bézier curves [55].

To retain recognizability, schematization typically limits the spatial distortion between input and output by fixing vertex locations [24] or minimizing the distance between input and output edges, for instance via the Fréchet distance [114]. In addition, it is common to maintain the topology of the input, which plays a key role in recognition of the output. We note that for our approach, we want to retain the topology of the network at a certain scale, thus small topological features should be removed prior to applying a topology-preserving schematization approach. An alternative would be to consider continuous scale generalizations [116, 117] and applying schematization at the desired scale. An important aspect is then to be able to retain a mapping from the edges in the selected network to the schematization.

Our implementation We first drastically simplify and collapse the selected network G', after which we apply the arc schematization algorithm by van Dijk *et al.* [114] for its aesthetic and clean representation, resulting in a schematic road-network $\mathcal{G}$.

Our implementation applies the sequence of operations described below. We use simple steps in an incremental fashion to facilitate coordination and maintain a mapping between the edges of the selected network and the schematic network. Our simple operations can result in fairly coarse approximations. However, since our target is a highly abstracted final map, the coarseness of the earlier operations is not an issue.

Collapse dead ends We first remove short paths in the network that do not increase the overall connectivity. Starting at a degree-1 vertex, we move along degree-2 vertices only to trace a visual "dead end" until we find a vertex that does not have degree 2. We compare its geometric length to a predefined parameter l_{max}, and collapse the path to this last vertex if its length falls below this threshold. We use $l_{max} = 100m$ initially, and $l_{max} = 1000m$ after the face-collapse operation. For coordination, we reroute any route along the collapsed edges to the endpoint that remains.

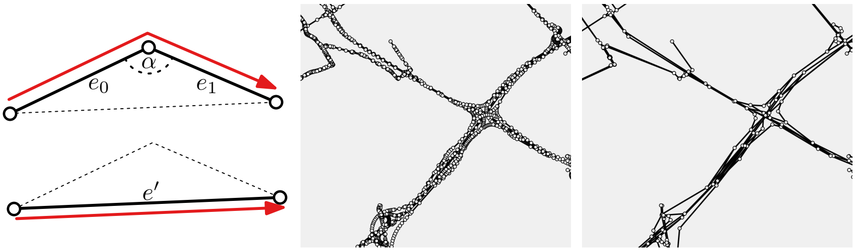


Figure 5.3 (left) Replacing shallow turns; (middle) network before; (right) network after.

Replace shallow turns A detailed input network frequently contains small bends. The final schematization would remove such detail, but we perform this step early to simplify the merge and collapse operations to follow. For every degree-2 vertex, we consider the smallest angle between its incident edges. If this angle exceeds some predefined limit β, we replace the vertex and its two incident edges with a single edge; see Fig. 5.3. We use $\beta = 150^\circ$ before and $\beta = 140^\circ$ after the face-collapse step. For coordination, we reroute any route on one or both of the replaced edges e_0 and e_1 to the new edge e'.

Merging vertices Junctions in the road-network are too detailed for our schematic; ideally we represent them by a single vertex. To this end, we fix a radius r within which we merge vertices. For a vertex v, the merge operation for v merges all vertices within distance r of v (including v itself) to a single new vertex, placed at the centroid of the merged vertices (see Fig. 5.4). We iteratively merge vertices, prioritized by the number of vertices within radius r. We use $r = 0.01D$ with D the length of the diagonal of the bounding box of the network. After merging faces, we use a larger radius of $r = 0.03D$.

For coordination, edges inside the merge radius are mapped to the new vertex (e.g., red edges to v_r in figure). Edges between different new vertices or between a new vertex and an unmerged vertex are consolidated to new edges (purple and blue in figure).

Merging vertices with edges A vertex can be close to an edge, even though it is not close to its endpoints. In such cases, we merge vertices into nearby edges that are not incident to the vertex itself. We use the distance r as specified earlier, and try to collapse a vertex v to the closest non-incident edge e that is within distance r, where we demand that v lies in the slab spanned by e (see

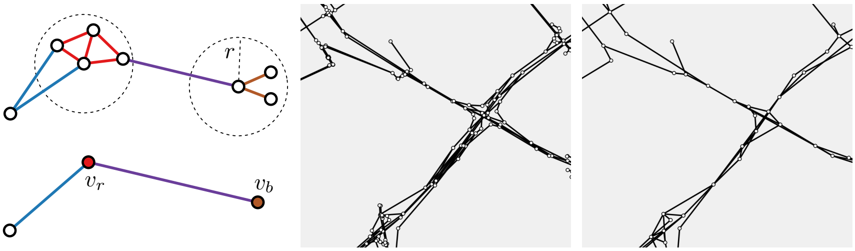


Figure 5.4 (left) Merging vertices; (middle) network before; (right) network after.

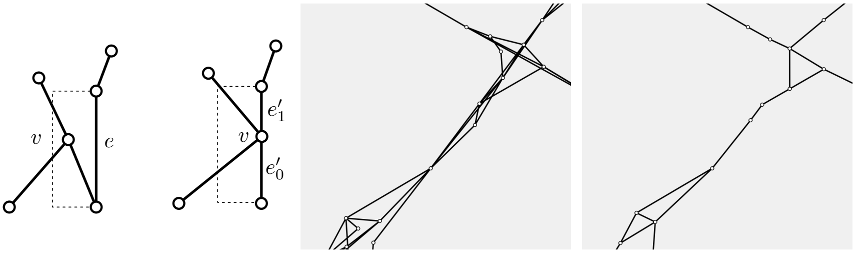


Figure 5.5 (left) Vertex-edge merging; (middle) network before; (right) network after.

Fig. 5.5). For coordination, any route using the former edge e is now using the two (possibly new) edges e'_0 and e'_1 from v to the endpoints of e.

Face collapse We now consider the faces of the network and collapse them onto a single geometry, if they are "small". Specifically, we collapse faces with an area of at most $a_{max} = 0.01A$, where A is the area of the bounding box of the network.

To collapse a small face F, we first collapse all interior degree-2 paths, independent of length. We then compute the minimal oriented bounding box of F, select its major axis, and cut F along this axis. In case of a non-convex face, we select the longest internal intersection between F and the axis (see Fig. 5.6). The cut introduces two cycles; we break both cycles by removing an edge or a path of degree-2 vertices from each; we can do so while maintaining the distances to the cut along the cycle for all higher-degree vertices. For coordination, we reroute any route that used the removed edges via the cut.

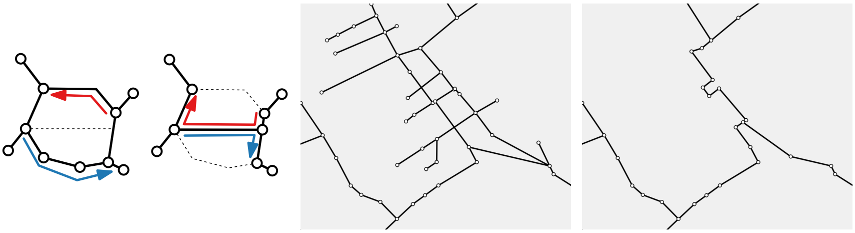


Figure 5.6 (left) Face collapse; (middle) network before; (right) network after.

Cleanup After face merging, we repeat all earlier operations once more to clean up the geometry and prepare for computing the final schematization.

Arc schematization For our final schematization, we chose the arc schematization algorithm by van Dijk *et al.* [114]. This algorithm produces a low complexity schematization using (circular) arcs, while maintaining the topology of network. To encourage the use of straight lines over very shallow arcs, we increase the relative importance of straight lines by reducing the Fréchet distance for straight lines by a factor 0.3. Because this algorithm only removes vertices, coordination is straightforward, similar to replacing shallow turns.

Time complexity The simplification operators run in roughly quadratic time with a straightforward implementation. The arc schematization algorithm dominates the operations to simplify the network. Thus, the time complexity is $O(n^2 h \log n)$, where h is the number of vertices with degree higher than three in the input simplification and n the number of vertices, which is greatly reduced from the input at this point. Coordinating the changes between the road network and a trajectory T takes roughly $O(|G||T|)$.

5.5 Step 4: Detect bundles

Desiderata We aim to detect bundles that are well-supported by the routes to provide insight into the overall mobility patterns. As such, we identify two desired properties of a bundle. First, it has to be supported by many routes. Specifically, we do not want a bundle representing a single trajectory (potentially an outlier), but rather the common behavior. Second, a bundle should be long in terms of (geometric) length. We aim to show mobility patterns that describe how vehicles move through the space. A long bundle is more descriptive of behavior than a short one; in the extreme case, a short bundle (even or especially if it is supported by many routes) may consist of one single edge (road segment), which does not help to communicate patterns beyond the existence of considerable local traffic. That is, we want to find long bundles that are supported by many routes.

As we aim to visualize not one but multiple bundles simultaneously, we further use three criteria to assess a set of bundles. First, we restrict our attention to *maximal* bundles only. That is, we consider only bundles to which we cannot add another route for its support, but specifically also not increase its length without having to reduce its support. Second, we generally want to see patterns of mobility that are *spatially diverse*: we prefer having bundles through different parts of the network, if the trajectories allow them. That is, we would like to avoid overlap between bundles as overlap communicates the same (local) behavior.

Third, a bundle fully contained within another may not provide much extra insight into mobility beyond showing a higher support for the contained route. We call a set of bundles *containment-free* if no bundle is a subroute of another. Note that containment-free bundles are not necessarily overlap-free and thus this is different from spatial diversity. Overlap-free bundles are containment-free, but we do not interpret spatial diversity as strict overlap-free.

In a containment-free set of maximal bundles each route may still support multiple bundles. If these are disjoint bundles, then we accept this as a bundle. However, as we are to eventually visualize the bundles, it may be misleading if well-supported bundles that share an edge are only well-supported because they share many trajectories. Thus, we choose to count the support for bundles in a disjoint manner: that is, routes through the overlap of bundles can be counted to support only one of these. We refer to this as the *disjoint* support.

Finally, note that we consider the network bidirectional, in the sense that each edge is present twice, once for each direction of travel. The considerations above should consider the direction of travel. That is, a bundle is directed and can, for example, only be supported by routes in the same direction. Two bundles that use the same set of edges, but travel in opposite direction, would hence be considered overlap-free.

Related work Our bundling is closely related to finding groups of trajectories and finding representatives of trajectories, so-called *centers*. In the context of spatio-temporal data, grouping structures are used to find common patterns of mobility [18]. The groups that are constructed essentially are trajectories that move close together for a sufficient amount of time and with enough members in the group. Since we work with map-matched trajectories, we consider routes close when they share edges in the network, and we require bundles to have at least some minimum length.

Note that finding a group does not directly result in finding a representation for the behavior of the group. To represent the behavior of groups, a common approach is to cluster trajectories according to some metric, resulting in centers describing common behaviour for the trajectories that are close to these centers under the used metric. This has been applied to the spatial component under different metrics and algorithmic approaches [20, 79, 93]; see [137] for a more comprehensive overview.

Since we work with map-matched routes, finding the bundles is similar to finding common substrings over a finite alphabet, where the alphabet is the set of edges and the routes form strings over this alphabet. Finding k-common substrings [10] is particularly related, since it demands a minimum number of matches k, similar to our high support requirement.

Our implementation We search for a spatially diverse set of up to k bundles, where each bundle is a maximal bundle with a minimum length $L_{\min}$ and minimum disjoint support $S_{\min}$. We construct this set by incrementally adding the “most informative” bundle.

To quantify this, we define the *importance* of a bundle in a way that allows a trade-off between length, disjoint support and spatial diversity. Specifically, the importance $I(B)$ of a bundle B is defined as $I(B) = L^p|S|^{1-p}$ for $p \in [0, 1]$, where L is the length of the bundle and S is the (disjoint) set of supporting routes. The rationale behind this is that, in the case of $p = 0$ or $p = 1$ we simply prioritize by total support or bundle length, respectively. However, if we set $p = 0.5$, then we prioritize the bundles by the total length of the route-set in its support. This allows a trade-off between length and disjoint support.

We define the length of a bundle B as $\sum_{e \in B} \ell(e)$ for some function ℓ. With $\ell(e) = |e|$ (the Euclidean length) we promote long bundles directly. As we use disjoint support, there is already some preference for spatially diverse bundles, but we observe that this still leads to very similar bundles. To promote spatial diversity further, we may alter ℓ. Specifically, we also allow for using $\ell(e) = |e|/(1 + b(e))$ where $b(e)$ is the number of selected bundles already using edge e. That is, conceptually, we “shrink” edges that have been used by other bundles. We thus refer to this setting as *shrunk edge lengths*.

We compute the most important bundle B by using a simple backtracking procedure. We then add B to our bundle set and remove its support from the complete set of routes. We repeat this process until k bundles have been found. If no bundle of sufficient length and support is found but we do not have k bundles yet, we halve S_{min} and repeat. We halve S_{min} at most twice, creating three *classes* of bundles ("thick", "medium" and "thin" bundles).

Intuitively, focusing on bundle length (higher values of p) is suitable for finding longer bundles that may have less support; useful to investigate longer mobility patterns. Reducing p focuses more on support and thus on patterns that are very frequent. In Fig. 5.7 (rendered according to our final step) we vary p and observe that reducing p (i.e., increasing importance of support) leads to less spatially diverse routes, but increases the bundle classes.

Changing the edge lengths may encourage diversity, but may result in bundles of lower classes, as routes through existing bundles are "shorter" and thus less important. In Fig. 5.8 we compare the result using both our settings. Euclidean edge lengths show mostly similar routes, but we observe that shrunk edge lengths increase the spatial diversity. It does not avoid overlap, but it does reduce containment slightly (from 5 to 2 contained pairs).

Computing one bundle takes $O(|\mathcal{T}||\mathcal{G}|)$ time, where $\mathcal{G}$ is the schematized network. This time includes the necessary changes to the information for computing the next bundle, so computing k bundles takes $O(k|\mathcal{T}||\mathcal{G}|)$ time.

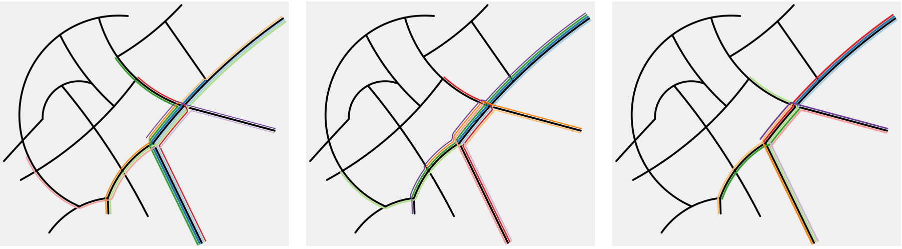


Figure 5.7 Ten bundles for different importance schemes: $p = 1$ (left), $p = 0.5$ (middle) and $p = 0$ (right). We used $S_{min} = 500$ and $L_{min} = 6000m$, with shrunk edge lengths.

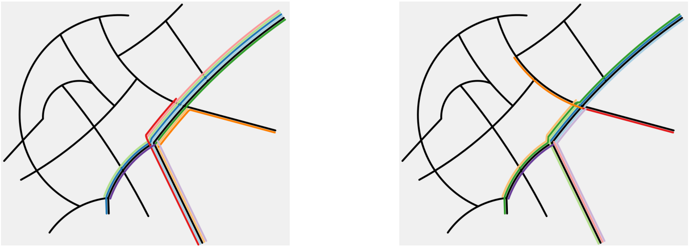


Figure 5.8 Ten bundles for different approaches to spatial diversity: Euclidean edge lengths (left), shrunk edge lengths (right). We used $S_{min} = 500$ and $L_{min} = 6000m$ and $p = 0.5$.

5.6 Step 5: Render schematic network with bundles

Desiderata We now aim to jointly render the bundles and the schematic network, such that the bundles are clearly conveyed. For this, the bundles should be easily identified. Common practice is to use colors to identify the separate bundles, and in addition separate them visually. However, the bundles need to be rendered such that it is also easy to determine from what parts of the network they originate. Thus, rendering them in close spatial proximity to the associated edges would also be beneficial to the readability of the schematic. In addition, it should be possible to identify the (approximate) support for a bundle.

Related work Rendering bundles in a network is similar to rendering the lines of a metro network, which is a well studied topic [127]. Common problems involve ordering lines on a common connection to avoid crossings and ensure good continuation at stations.

The offset rendering used in our implementation may want to avoid offsetting edges of a bundle with different distances, thus encouraging good continuation. If we allow gaps between bundles at an edge but disallow differences in offset within a single bundle, the problem is essentially to assign a layer to each bundle, minimizing the number of layers, such that two overlapping bundles use different layers. Even if the network would be a single cycle, this problem is NP-hard [53]; minimizing change is hence NP-hard as well.

Our implementation Our main approach is to render each bundle as a a curve that is slightly offset from the network, such that they do not coincide with the

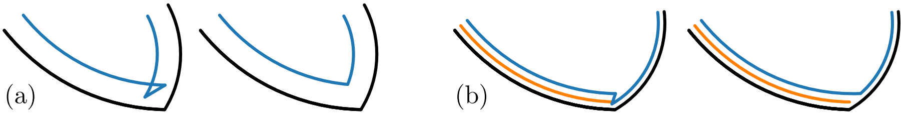


Figure 5.9 (a) A local self-intersection caused by offsetting and its resolution. (b) Local deviation from the direction of travel of the bundle and its resolution.

network, nor each other. We visualize the direction of a bundle by offsetting its curves to a particular side of the network edges. As our example datasets are both in areas where cars drive on the right side of the road, we hence locally offset to the right. This scheme allows for identifying the direction of a bundle, without relying on other visual cues such as arrow heads.

We lay out the bundles one at the time along the network. Each edge in the bundle gives an arc that is offset from the edge by a distance $d(b+1)$ for some constant $d > 0$, where b is the number bundles already placed at that edge. The bundle is now a sequence of arcs that do not quite connect correctly yet. We initially reconnect the arcs using straight segments. If this causes the curve to locally self-intersect (Fig. 5.9(a)) or cause small corners (Fig. 5.9(b)), directed opposite to the actual bundle direction, we simplify these artifacts as to achieve a simple curve that is always (roughly) directed in the direction of travel of the bundle. This operation takes $O(l \log l)$ time, where l is the total complexity of the rendered bundles. Finally, we slightly smooth the connecting segments by reducing the arcs by a small distance and using the old endpoints as control points for a Bézier curve instead. We render the resulting curves using colors of the “10-class Paired” qualitative scheme from ColorBrewer (`https://colorbrewer2.org/`), and use a line thickness based on bundle class. By using classes instead of support, it is primarily aimed at separating main from secondary patterns.

5.7 Results

We implemented our proposed pipeline in C++, using MoveTK[1] for trajectory processing and CGAL[2] for geometric operations.

The HR dataset has been used to illustrate and discuss our pipeline throughout

---

[1] `https://movetk.win.tue.nl/`

[2] `https://cgal.org/`

the chapter. It covers the area of The Hague, the Netherlands. The network is obtained from OpenStreetMap[3] and has 60 277 vertices and 66 895 edges. In step 2, we use 'primary' as the minimum road level for selection[4]. The GPS trajectories were provided by HERE Technologies (`https://www.here.com/`). As we are looking for patterns in mobility, we used only trajectories with a length of at least 10 000m that fall within the bounding box of the network. Trajectories partially within the bounding box were split and each part was treated as a separate trajectory. This left us with 3 795 trajectories as input.

The BJ dataset covers the metropolitan area of Beijing, China. The network is also obtained from OpenStreetMap and has 77 691 vertices and 132 167 edges. In step 2, we again use 'primary' as the minimum road level for selection. The GPS trajectories originate from the open Geolife trajectory dataset by Microsoft [140], constrained to this region. We apply the same filtering step as for HR; this left us with 7 520 trajectories as input. The result of our pipeline is shown in Fig. 5.10. The Geolife dataset mainly contains trajectories obtained from taxi-drivers. We can clearly see in the heat-map of the dataset that there is a high concentration in the top-left, and in our schematic we see that relatively small, loop-like routes, capture a lot of the traffic in that region. Moreover, we see that most bundles occur in pairs, that is, two roughly identical bundles but in opposite directions.

Our schematic map generally captures the outer ringroad, but struggles somewhat to capture the grid-like structure of the inner city. This is because the arc-based nature of our schematization algorithm is somewhat opposite to such structures. Future work may investigate hybrid approaches to schematizing such mixed networks of ring roads and non-grid-like structures with arcs, but grid-like structures with parallel segments.

▸ 5.8 Discussion and future work

We propose a coordinated pipeline to create abstract visual summaries of mobility patterns in trajectory data. Our proof-of-concept implementation shows that the pipeline is feasible and can fully automatically compute such schematic maps. The advantage of a pipelined approach is that we may improve upon steps individually to improve the eventual result. Below, we reflect on our choices in the pipeline design, and discuss future work.

---

[3] `https://www.openstreetmap.org/`

[4] `https://wiki.openstreetmap.org/wiki/Key:highway`

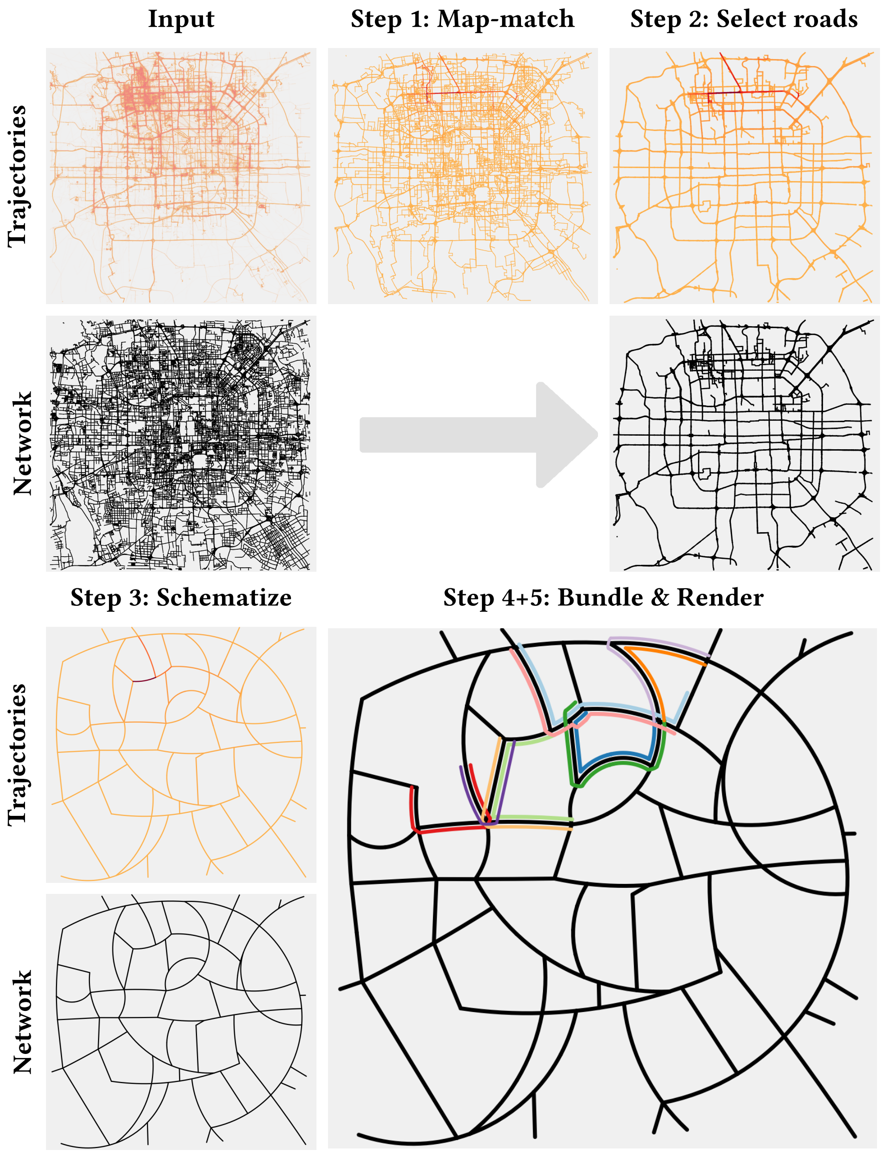


Figure 5.10 Our pipeline on the BJ data. The input trajectories are shown as a density map. For the map-matched routes, we use a orange to red scale to convey low to high traffic volume per edge. We compute the bundles in Step 4 with $S_{min} = 150$, $L_{min} = 8000m$, $p = 0$ and shrunk edge lengths.

Step order We could in principle also map-match (step 1) after selection (step 2). However, this would not allow for a data-driven approach for selection. Moreover, this forces all traffic to the selected roads, which may hide information on traffic that does not follow the selected (major) roads. For HR, the data-driven selection seems to not have made a large impact on the selected network; for BJ more extensive parts of the network were added because of the data-driven selection. We leave to future work to investigate whether inversion of these steps is able to provide meaningful visualizations.

Similarly, we may wonder whether we would want to detect bundles (step 4) before schematization (step 3). As schematization reduces the network complexity, it is more efficient to do afterwards. It may distort distances, but we can keep track of the original distances if desired – note that aggregation in step 3 does not make each route in the schematic network map to precisely one route in the original network. Another reason supporting our choice for the given order is that schematization may further aggregate dense areas of the network. By bundling afterwards, the support for such bundles grows since they are effectively representing more traffic that generally traverses the dense area in roughly the same way. We believe that our choice helps in promoting spatial diversity, as dense areas with low traffic per road may reduce to a single road with higher traffic. In light of our very spatially uneven datasets, this seems desirable. However, we leave the full exploration of the impact of this choice as future work.

Augmenting the schematic map We split the map-matched trajectories according to whether or not the route is on the selected network. This leaves us with parts of the trajectories that go through unselected parts of the network and are thus dropped from the schematic map. We intend to explore ways of visualizing these dropped subroutes to provide information on the traffic not part of the selected network. On a computational level, an approach we see for this is to track these subroutes relative to the faces of an embedding of the network. This, however, demands that we meticulously keep track of what happens to these faces during the simplification stages. But it also requires visual design: what do we want to show of these dropped routes, and how does that combine with the shown bundles?

While our mapping between simplifications is discrete in nature, an interesting direction of research would be to extend this to continuous mappings, where routes are also allowed to start in the middle of edges. An appropriate map-matcher should also be selected, since the Fréchet map-matching approach maps only to full edges, though we expect the overall impact of allowing continuous routes here to be minor, as it is performed on the original, detailed network. The primary question is how we can alter the schematization step to allow for high-quality continuous mapping through aggregation and simplification of the network.

Using the schematic map In our proposed pipeline, we do not incorporate the time component of the input trajectories explicitly. Given that the schematic shows strongly aggregated data in a concise way, we can easily use our approach to show small-multiples for different time frames in the data set. This leaves open the question what the best selection method of the road-network is in this case, which we defer to future work.

Our method hides any of the effects of sampling and noise in the data, as well as the deformation and aggregation that occurs in our pipeline, which may be unintuitive to end users. Though the schematic appearance aims to implicitly convey such information, it may be communicated more explicitly with additional uncertainty visualization, albeit at the cost of added visual complexity.

The final schematization is strongly influenced by the selected parameters in steps of our pipeline. We scale parameters by a typical size (e.g. bounding box diagonal or area) to be able to assign parameters independent of scale. Nevertheless, visualizing the impact of the parameters on the end result could help users pick appropriate values for their use cases.

Chapter 6

Schematizing Orthogonal Polygons

In this chapter, we seek to analyze the schematization of polygons under new similarity measures. These similarity measures are vital for the schematization process, since they largely determine what we deem high quality schematizations. Thus they should capture our intuition for nice schematizations well. Though we often use bottleneck distances such as the Hausdorff distance and Fréchet distance because of their easy implementation and interpretation, more complex distance measures may be needed to properly capture what constitutes good schematizations. This is especially so when we have to deal with outlying features of the polygon, to which the bottleneck measures are particularly sensitive. To this end, we can consider area-based similarity measures such as the area of symmetric difference. We use a more generalized version of this measure, namely the *minimum homotopy area* [30], that also incorporates how one shape is transformed into the other, which ties into our intuition that shapes should not deform too much if we need them to be similar [58].

To be able to more easily analyze our schematization techniques, we restrict ourselves to *orthogonal polygons* as input and output: any polygon we consider only has edges that are strictly horizontal or vertical; see Fig. 6.1 for two examples. We abbreviate them as ortho-polygons, similarly for polylines that are orthogonal to ortho-polylines. One can interpret these polygons as a rasterization of a general polygon, tracing the boundary of the polygon afterwards.

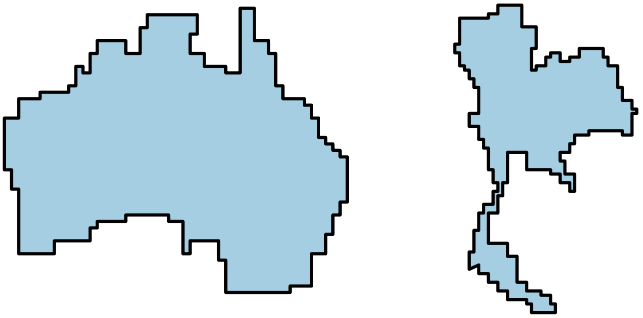


Figure 6.1 Examples of simple orthogonal polygons.

Minimum homotopy area Minimum homotopy area is a measure, proposed by Chambers and Wang [30] as a generalization of the Fréchet distance. It measures the similarity between (closed) curves by considering the deformation from one to the other. It has been used for comparing polygons as well as trajectories [29, 30]. To define homotopy area for our setting, consider an input polygon P that we schematize into a polygon S. We interpret the input polygon P and its schematization S as continuous functions mapping the unit interval $[0, 1]$ to $\mathbb{R}^2$ for an arbitrary fixed starting point on the polygon. Let $\pi : [0, 1] \longrightarrow [0, 1]$ be a bijective function that defines a matching between S and P using their respective parameterizations. Then, a *homotopy* $H : [0, 1] \times [0, 1] \longrightarrow \mathbb{R}^2$ between polygons P and S with some mapping π is defined as a continuous deformation from P to S over a time $t \in [0, 1]$ such that $H(a, t = 0) = P(a)$ and $H(a, t = 1) = S(\pi(a))$. Here, a and is the parameter for the parameterization of P.

The homotopy area of H is defined as the total area that is swept by the deformation, with multiplicity. That is, any area that is swept over multiple times is counted that many times. The minimal homotopy, H^*, between two curves is a homotopy with the smallest homotopy area; we denote its homotopy area by $\sigma(P, S)$. See Fig. 6.2 for an example of a minimum homotopy and its associated area.

If two polygon boundaries do not intersect, either their interiors are fully disjoint or one is fully inside another. In both cases, the minimal area of a deformation from one to the other is actually the area of symmetric difference between the two polygons. So only when the two polygons intersect do the minimum homotopy area and the area of symmetric difference differ. However, a key conceptual difference is that, whereas the area of symmetric difference looks at the area of the polygons only and any mismatch in that, the minimum homotopy consider the polygon boundaries

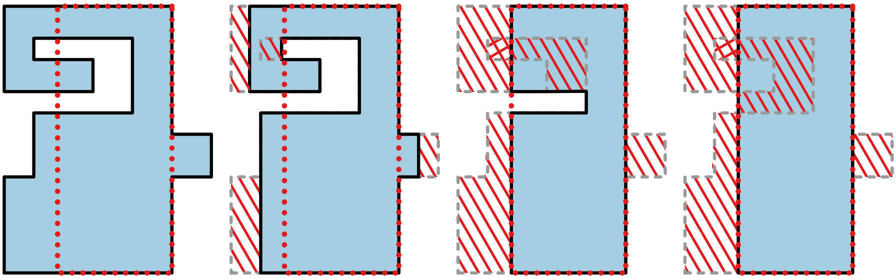


Figure 6.2 Homotopy and its associated area between two polygons, one defined by the black boundary and blue interior and the other by the dotted red boundary. Hatched area shows area contributing to the minimum homotopy area (doubly hatched is counted twice). Grey dashed line shows the original shape of the black and blue polygon.

to be important, which is useful in the schematization process. In that regard, the minimum homotopy area is more similar to Fréchet distance: both consider a continuous deformation between the curves. However, the Fréchet distance measures the maximum distance between the start and end location of any point on the curve, whereas the minimum homotopy area considers the total swept area.

As Meulemans [87] showed, the correct choice of distance measure for schematization is very important for a specific application. Bottleneck measures such as the Fréchet distance are particularly sensitive to noisy outliers of the input shape, or in general long spikes that may stick out of the shape. As shown in the example of Fig. 6.3, a bottleneck distance such as the Fréchet distance needs to shift towards the spike feature, whereas an area-based measure is less sensitive to the spike, since its area is small.

To make the schematization less sensitive to outliers and spikes, an area-based measure, such as the area of symmetric difference or the minimum homotopy area, thus can be chosen. However, as Meulemans [87] shows, the area of symmetric difference does not take into account some salient local topological features, such as the opening of the U-shape in Fig. 6.4. In this example, the desired schematization maintains the U-shape of the input curve, which amounts to maintaining the top gap in the input shape. However, for the area of symmetric difference, a C-shape is better. For the homotopy area, producing the C-shape requires moving the boundaries more, thus creating a large swept area, with a part being doubly swept. Hence, the U-shape is better for the homotopy area measure.

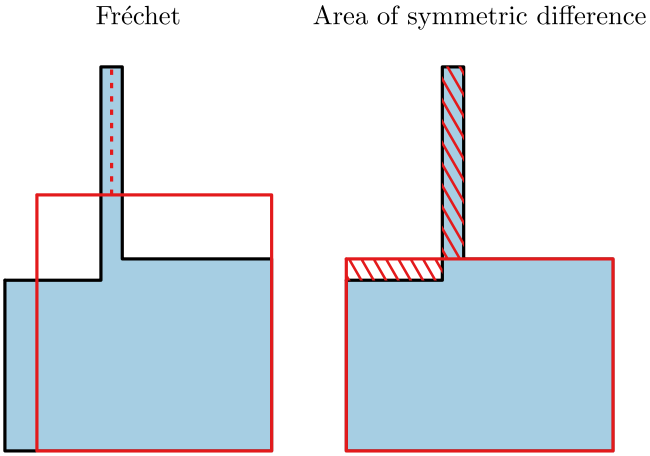


Figure 6.3 Schematizing a polygon with an outlying spike. The blue polygon is the input polygon, red curve shows the resulting polygon boundary after schematization. (left) Schematization under the Fréchet distance (bottleneck distance given by dashed line). (right) Schematization under the area of symmetric difference (hatched areas contribute to the area of symmetric difference).

Related work Schematization is a well studied topic for different geometric objects. Recall from the previous chapter that we schematize a geometric object to get a more concise representation that can be used for visualization. In this chapter, we consider the specific case of orthogonal schematization.

A particular way to remove irrelevant details of geometric objects is to *stylize* the object. In particular, one can limit the type of geometry that is allowed for the schematization. Many different styles have been developed in the past. C-oriented schematizations [24, 42] only allow straight lines to be used for the geometric object boundaries, where the orientations are limited to a constant number of orientations, defined in a set C. From this class of stylistic schematizations, orthogonal and octilinear schematizations are well-known examples [128]. Alternatively, one can restrict the geometry connecting vertices of the schematized object, such as only using circular arcs [43] or bounded-degree curves [50].

Apart from stylistic choices for the resulting geometry, it is important to pick the right similarity measures or criteria that we are optimizing for, such that a certain

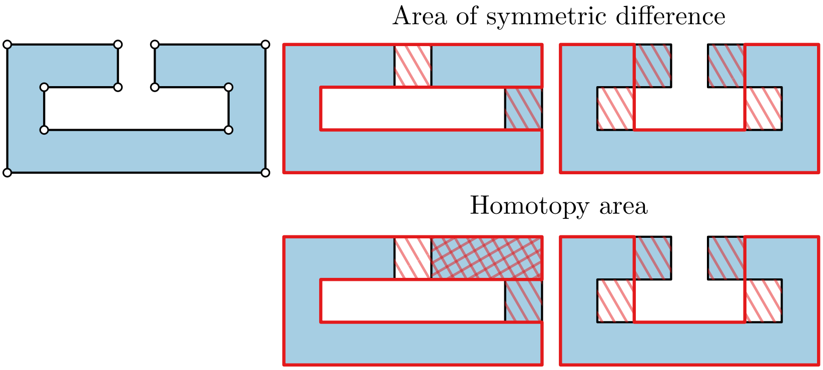


Figure 6.4 Orthogonal polygon with 10 edges (top left), with an unreasonable (middle columm) and reasonable (right column) schematization for $k = 8$. Red curve shows schematization, hatched areas show incurred area cost for the area of symmetric difference (top row) and homotopy area (bottom row). Doubly hatched areas are counted twice.

semblance to the input is retained for the resulting schematization and such that visualization goals are met. Measures such as the Fréchet distance and Hausdorff distance are used [43, 111]. Optimization criteria may include area preservation or minimizing bends [88, 96]. In this chapter, we consider schematization under the minimum homotopy area, which conceptually is similar to area of symmetric difference and the Fréchet distance. Additionally, taking into account topology is important for schematization. That is, we want the local connectivity of the polygon with itself and its surroundings to remain the same. In particular for polygons, when the input polygon is a simple polygon without holes, it stands to reason that the schematization should have these same properties (see Fig. 6.5a for an example of where simplicity is not retained). For many variants of the schematization problem, keeping the topology the same between input and schematization is actually an NP-hard problem [49, 63, 82]. We explore the implications of maintaining simplicity for the schematization under the minimum homotopy area, and conjecture that maintaining simplicity when the input polygon is simple is also an NP-hard problem.

While schematization can be applied to single polygons and polylines, it is common to consider a planar subdivision, such as in the case of countries on a continent. Algorithms have been developed for this particular setting as well, where in partic-

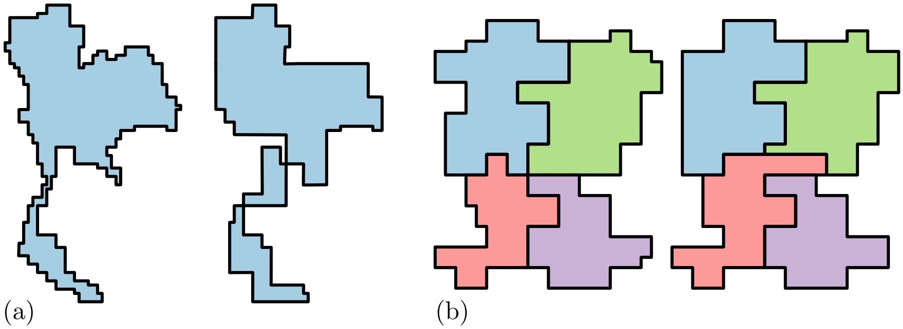


Figure 6.5 (a) Not retaining simplicity in the schematization may result in a schematization that is not meaningful (right side). (b) Connectivity in a planar subdivision is important to retain when schematizing.

ular the topology plays a large role: it is undesirable to have countries bordering new countries that they previously did not share a border with [88, 115]. Fig. 6.5b shows an example where the connectivity of the blue and green region with the purple region is erased in the schematization, which may result in incorrect interpretation of the connectivity between the regions. In this chapter, we limit ourselves to schematizing polygons only, though extending it to planar subdivision would be an interesting direction for future work.

Problem statement Formally, we seek to answer the following question in this chapter: given a simple orthogonal polygon P of complexity n, can we efficiently compute a orthogonal polygon S of complexity k, which we call its *schematization*, such that the minimum homotopy area between P and S is minimized?

Contributions In this chapter, we present a new algorithm that computes the optimal schematization of an orthogonal polygon under the minimum homotopy area in $O(n^4(k + \log n))$ time, where the resulting schematization S may be non-simple. To improve the running time of the algorithm, we consider a greedy algorithm that heuristically estimates the cost of a schematization step. We analyze the performance on *staircases*, defined as xy-monotone orthogonal polylines. We show that for staircases that have uniform edge lengths, the greedy approach yields a schematization that is a constant factor approximation for the optimal schematization with factor 36. This result generalizes to staircases where all horizontal edges are of the same length and all vertical edges are of the same length.

Organization In Section 6.1 we discuss how to compute the minimum homotopy area measure for completeness and analyze the results of schematization under the minimum homotopy area if simplicity is desired. We show how to compute an optimal schematization that is allowed to self-intersect in Section 6.2. Since the running time for this algorithm is high, we seek to approximate the optimal schematization. In Section 6.3 we propose a greedy approximation algorithm and analyze its performance for uniform staircases. Finally, we reflect on our results in Section 6.4.

6.1 Minimum homotopy area for schematization

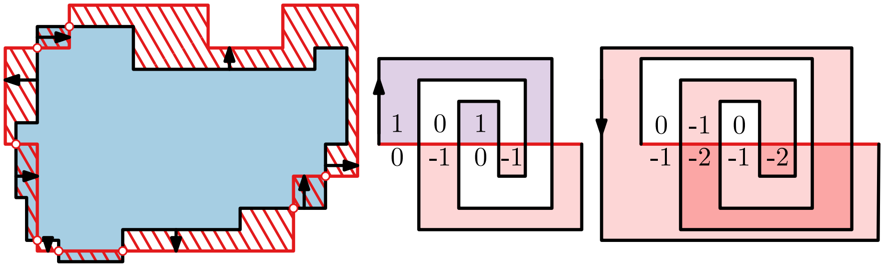
We first discuss how to compute the minimum homotopy area for completeness and to introduce important concepts that we reuse in our algorithm for schematizing polygons where the schematization may self-intersect. We then characterize schematizations under the minimum homotopy area, where we demand simplicity of the produced schematizations.

6.1.1 Computing the minimum homotopy area

For completeness, we describe here on a high level how to compute the minimum homotopy area between two polygons, as proposed by Chambers and Wang [30]. We also introduce important concepts that we use in our schematization algorithm in Section 6.2 during the process.

Consider two polygons P and S and let the minimal homotopy be defined as described in the previous section. We define *anchorpoints* for some homotopy H between P and S to be points on P and S that remain stationary throughout the deformation H. Since H maps P to S, this automatically means that anchorpoints can only be points where P and S intersect. Moreover, these anchorpoints are stationary in the minimal homotopy and must therefore occur in the same order along both curves (Observation 3.1 in [30]).

Chambers and Wang show that when the two polygons intersect, the minimum homotopy area has at least one anchor point. Each pair of consecutive anchor points (or the only anchor point twice) delimit a subcurve of both P and S that morph to each other via the homotopy. We call this deformation a *subhomotopy* of the (total) homotopy. It then follows that the minimal homotopy has subhomotopies that are minimal on their own for the subcurves of P and S (see Fig. 6.6a for an example).



(a) (b) (c)

Figure 6.6 (a) Morphing the blue polygon to the red polygon boundary. Subhomotopies morph over the hatched areas, with anchorpoints (red markers) separating the subhomotopies. Arrows show the local directions. (b-c) Homotopy between black and red polygon boundary that has inconsistent (b) and consistent (c) winding number. Numbers are given in the cells. Arrow on the boundary shows assumed direction.

To be able to compute the homotopy area for a single subhomotopy, Chambers and Wang show that this can be done by considering the concatenation of the subcurves of P and S that the subhomotopy morphs. When concatenating the two, we create a new, possibly non-simple, polygon. We consider the arrangement of this polygon, where all self-intersections are vertices. Each cell of this arrangement has an associated *winding number*: the number of times one wraps around when at a point in the cell and following the polygon boundary. As Chambers and Wang show, a subhomotopy can only be part of the minimal homotopy if it has *consistent winding numbers* (Lemma 3.2 and 4.1 in [30]), that is, all winding numbers in the cells of the arrangement are non-positive or all are non-negative (see Fig. 6.6b-c). If this is the case, the homotopy is also *sense-preserving*: the deformation consistently deforms the polygon locally either to the left or to the right. Finally then, the minimum homotopy area of the subhomotopy is the sum of the areas of all cells in the arrangement, weighted by their respective absolute winding number.

By formulating a dynamic program over the intersections between P and S, one can now find the subhomotopies such that each subhomotopy has consistent winding number and the total homotopy area of the subhomotopies is minimal. By using a smart data structure to check the consistency of winding numbers in the arrangements of subhomotopies, the total running time is $O(I^2 \log I + n \log n)$, using $O(I^2 + n)$ space, with I the number of intersections and n the total complexity of the curves.

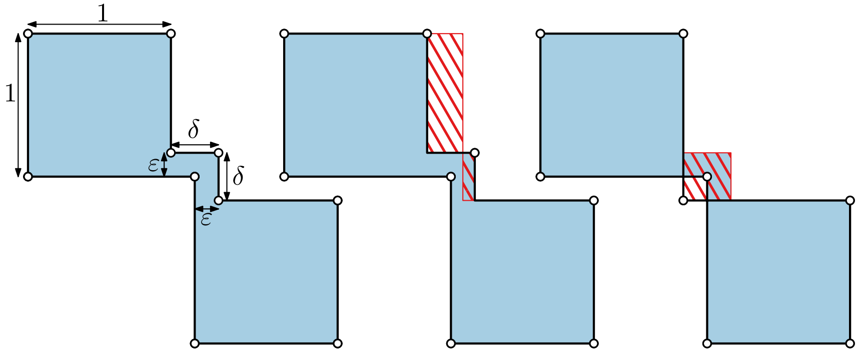


Figure 6.7 (left) Orthogonal polygon with 10 edges. (middle) Optimal simple schematization S has homotopy area $\Omega(\delta)$ for small ε. (right) Optimal nonsimple schematization S' with 8 edges self-intersects and has homotopy area δ^2.

6.1.2 Simplicity of a schematization under minimum homotopy area

An important property that we would like to preserve is the simplicity of the input shape: if the input polygon P is simple, can we guarantee that its schematization S is simple?

For the minimum homotopy area measure, we can show that this is not the case unfortunately. Moreover, we can characterize the difference in area between the optimal simple and non-simple schematization by the ratio δ of the shortest to longest edge length. Let P be an orthogonal polygon with a ratio $\delta \ll 1$ between the longest and shortest edge length as shown in Fig. 6.7. Let S and S' be the optimal simple and nonsimple schematization of P with complexity $|P| - 2$. Since the simple schematization needs to avoid creating an intersection in the middle, it needs to move one of the longer edges to eliminate two edges, thus incurring more area for the homotopy area. If it were to move the vertical short edge to the right or the horizontal one to the top, the minimum homotopy area would be worse than moving a longer edge, and thus not be optimal. Following this example, we see that the optimal non-simple schematization should have at least minimum homotopy area δ^2, whereas the simple version has homotopy area at least $(\delta - \varepsilon) \cdot (1 - \varepsilon) = \Omega(\delta)$ for small ε. Thus, the ratio of the areas of the optimal nonsimple and simple schematizations $\sigma(P, S)/\sigma(P, S')$ is lowerbounded by $\Omega(1/\delta)$.

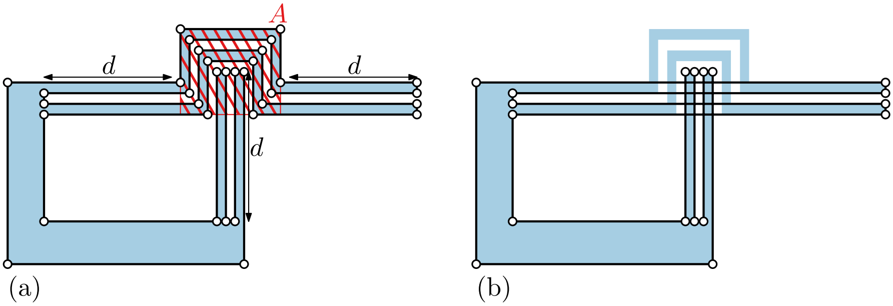


Figure 6.8 (a) Ortho-polygon with $n = 34$ and $\kappa = 4$. Length d is larger than area A; drawing shows d to be smaller for clarity of illustration. (b) Optimal nonsimple schematization S' for $k = 18$, with $\kappa^2 = \Omega(k^2)$ intersections.

A natural next question would be how many self-intersections an optimal schematization under the minimum homotopy area can have, as a way to quantify nonsimpleness. We show the following lemma.

6.1.1 Lemma. *Let P be a simple input polygon. The nonsimple schematization S' of P of complexity k can have $\Omega(k^2)$ self-intersections in the worst-case.*

Proof. Consider the example in Fig. 6.8. We construct the example such that the thin subpolygons of length d and width ϵ are such that $\epsilon \ll d$ and $d > A$. We now require that we remove enough edges such that the bumps in area A can all be flattened. Since the long thin subpolygons are too large to remove, the optimal minimum homotopy area schematization needs to remove the small bumps. It thus sweeps an area of A, resulting in a number of intersections that is quadratic in the number of removed edges.

If we now repeat the number of bumps at the top κ times, and similarly make κ teeth of the bottom comb, we get that there are at least $\Omega(\kappa^2)$ intersections if we require $k = 2\kappa$. We then have that $k = \Theta(\kappa)$, and thus the lemma follows. □

Note that this bound is trivially asymptotically tight.

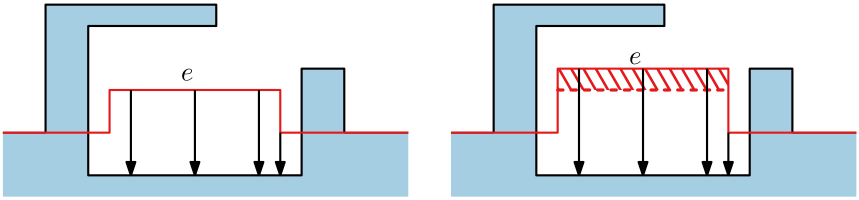


Figure 6.9 Illustration for proof that an anchorpoint is always need. (left) Edge e of the schematization (red) does not have an anchor point and thus can freely move locally in the opposite direction of the sense-preserving homotopy (black arrows). (right) Moving e in the opposite direction of the homotopy worsens the homotopy area by a certain amount(hatched red area), thus the edge must have an anchorpoint

▸ 6.2 Computing minimum homotopy area schematizations

From the previous, it follows that minimizing the minimum homotopy area only does not guarantee simplicity. We conjecture that actually computing a schematization that is simple and minimizes the minimum homotopy area over all possible simple schematizations is NP-hard, in line with hardness results for related schematization problems. Thus, we now compute an optimal schematization S^* for a given value of k that is allowed to self-intersect. Leveraging the results by Chambers and Wang [30], we first prove that there exists an optimal solution that has a canonical form. Using the fact that this canonical form exists, we propose a dynamic program for computing an optimal schematization in polynomial time.

▸ 6.2.1 Canonical form

We first show that the resulting schematization has a canonical form: there exists a schematization whose edges overlap edges of the input and whose minimum homotopy area is optimal. To show this, we first prove the following lemma for the optimal schematization S^*:

6.2.1 Lemma. *Each edge of S^* has at least one anchorpoint.*

Proof. Assume for contradiction that the optimal solution S^* has an edge e without any anchorpoints(see Fig. 6.9). Let H^* denote the minimal homotopy between S^* and P. Consider the subcurve S of S^* between two consecutive anchorpoints that

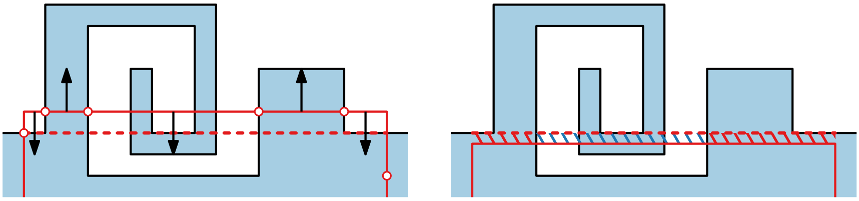


Figure 6.10 Illustration of the canonical form of an optimal solution. (left) an edge of the schematization (red horizontal segment) that has multiple anchorpoints (red markers), but does not have an anchor segment. The optimal placement is shown with the dashed red line. (right) Moving beyond the stable position incurs too much homotopy area (red hatched) versus the amount of area gained (blue hatched).

contains e with associated subhomotopy H. As H has no anchorpoints, it must be sense preserving (Lemma 3.1 of [30], using our observation that this generalizes to self-intersecting curves). Then, changing S^* by moving e slightly in the direction of deformation decreases the homotopy area of H and thus of H^*. This contradicts that S^* has minimal homotopy area and the lemma follows. □

Hence, in an optimal solution, subhomotopies span at most two edges in S^* and the matching subcurve is simple. Thus, self-intersections of S^* can occur only between edges separated by anchorpoints. We define an *anchorsegment* to be a nonempty subsegment of an edge e of S^* that coincides with an edge e' of P, such that all points on this anchorsegment are anchorpoints; the direction of e and e' must match, if this segment is more than a single point. We can now show the following.

6.2.2 **Lemma.** *There exists an optimal schematization S^* such that every edge of S^* has an anchorsegment.*

Proof. For a contradiction, assume that all optimal schematizations have an edge e that does not overlap an edge of P with the same direction (see Fig. 6.10 for an illustration). Let H^* denote the minimal homotopy between S^* and P. By Lemma 6.2.1, we know that e has one or more anchorpoints and thus we may consider the two or more subhomotopies $H_1, \dots, H_m$ that involve parts of e. Each H_i is between two simple curves and thus sense preserving (Lemma 3.2 of [30]).

Without loss of generality, assume e is horizontal. As the subcurves are simple, each subhomotopy area is the multiplication of area and winding number, summed over all faces in the arrangement (Lemma 4.1 and Lemma 4.3 of [30]). Consider moving e

up or down: this move may change the homotopy area for each H_i, but cannot cause local intersections in the subcurves. Hence, the total change Δh in the area of all minimal subhomotopies is the change in face area with winding-number multiplicity. Either Δh is zero in both directions, or Δh is positive in one direction and negative in the other.

In the latter case, we find a contradiction with the optimality of S^*, so assume $\Delta h = 0$. We may freely move the edge up or down, until one of the considered arrangements changes. At this moment e must overlap some edge e' of P, also considered in one of the original subhomotopies. Let S' denote the new schematization, with minimal homotopy H'. Now, either (a part of) this overlap is stationary in H' and thus an anchorsegment, or the entire edge still moves in H'. The former case implies an overlap in more than a single point: otherwise, Δh does not change. That is, the arrangement may have a face split, but these have the same winding numbers. In the latter case, we can continue shifting our edge e as Δh is zero (or positive in the same direction, contradicting optimality).

Note that we cannot make an edge of S^* disappear during this motion. □

Lemma 6.2.2 implies a canonical form for S^*, in which each edge is anchored to an edge of P through its anchorsegment. Consequently, every vertex of S^* lies on the grid G induced by the edges of P. This now allows us to define an algorithm for computing this optimal solution.

▸ 6.2.2 Computing the optimal schematization

We are now ready to compute the optimal schematization of a polygon P under the minimum homotopy area. For easier explanation, we define the *forward halfline* of an edge e_i as the halfline originating from v_i, overlapping e_i. Similarly, the *backward halfline* of an edge e_i originates from v_{i+1}, overlapping e_i. An edge e of S^* must then start on the backward halfline of its anchored edge in P and end on the forward halfline, and the direction of e matches the direction of its anchored edge.

We compute the optimal schematization as follows. First, we pick a midpoint of an edge of G that is on an edge of P to cut P into an orthogonal polyline P'. Next, we select only vertices of G as intermediate vertices for a potential schematization. Thus, an optimal schematization of P' with $k + 1$ edges is a schematization of P with k edges. Testing all $O(n^2)$ midpoints yields the optimal schematization S^*, as any anchorsegment must contain such a midpoint. Let P be an orthogonal polyline in the remainder of this section, to be schematized with k edges. We now describe the structure that a partial solution takes, allowing us to formulate a dynamic program to compute the optimal schematization for P'.

Partial solutions. We define a partial solution to be an ortho-polyline S where each edge has an anchorsegment. It represents a schematic up to some edge of P with a fixed starting edge of P (chosen by the midpoint that we split). Its first anchorsegment starts at v_1 on e_1. The last anchorsegment anchors to some edge e_j, but is not yet complete. In particular, the one before last vertex of S lies on a gridpoint $\gamma(i,j)$, defined by the intersection of the forward halfline of the edge e_i that the one before last edge of S anchors to, and the backward halfline of edge e_j that the last edge of S anchors to. Let $g(j,m)$ be the m-th gridpoint on edge e_j, indexed from start to end in the canonical direction. Then, the last vertex of S is the gridpoint $g(j,m')$ of G such that $g(j,m'-1)$ and $g(j,m')$ form a subsegment of e_j, encountered along the forward halfline of e_j starting from $\gamma(i,j)$(see Fig. 6.11). Note that $g(j,m'-1)$ coincides with $\gamma(i,j)$ for cases (c) and (d) in the figure.

Between two anchorsegments of S we can compute the minimal subhomotopy area, even if the last is not yet complete. That is, consider two subsequent anchorsegments of S, anchored to edges e_i and e_j of P with $i<j$. The corresponding subhomotopy $\sigma(i,j)$ can be computed purely from this information, since we only need the subcurves between the anchorsegments to compute the minimal subhomotopy area. If the forward halfline of e_i does not intersect the backward halfline of e_j, we call such a pair *incompatible* and use $\sigma(i,j)=\infty$. As anchorsegments are directed and ordered, we need to consider only pairs of edges that are compatible ($\gamma(i,j)$ exists).

However, we must be careful not to reverse an edge of S. Suppose S ends with edges anchored at $e_{i''}$, $e_{i'}$ and e_i, such that the first two edges are compatible, as well as the last two. If $\gamma(i',i)$ is after $\gamma(i'',i')$ in the direction of $e_{i'}$, then the edge of S anchored

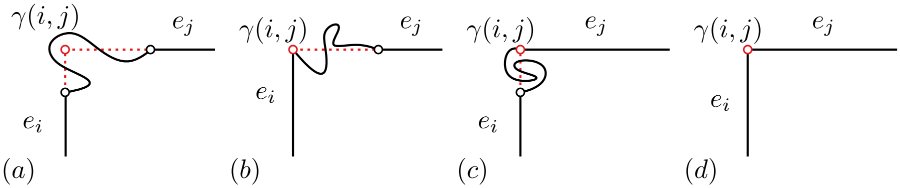


Figure 6.11 Four cases for compatible edges e_i and e_j to compute $\sigma(i,j)$. The gridpoint $\gamma(i,j)$ (red marker) can be outside both edges (a), on one of both edges (b,c), or on both edges (d).

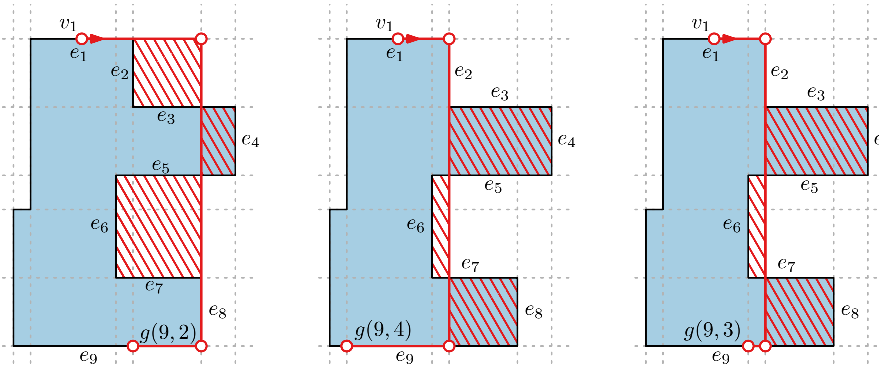


Figure 6.12 Partial solutions for input P (blue) with v_1 at the top. Shown in red are $D[9, g, 3]$ for various gridpoints g. Homotopy areas are given by the red, hatched area.

on $e_{i'}$ is directed along $e_{i'}$. However, if this is not the case, this edge of S is reversed and does not follow the canonical form. Hence, we do not need to consider these cases.

To decide whether we can extend partial solution S ending at e_j thus depends on where the last vertex of S is located with respect to e_j. We thus pair edge indices j with gridpoints g. We call a pair (i, g) a *compatible predecessor* for (j, g') if $i < j$, edges e_i and e_j are not incompatible, g is on or before $\gamma(i, j)$ along the forward halfline of e_i, and g' occurs before $\gamma(i, j)$ along the backward halfline of e_j.

Dynamic program. With the definition of our partial solution, we can now formulate a dynamic program to find the optimal schematization. Recall that $g(i, m)$ is the m-th gridpoint on edge e_i, when following its canonical direction. We characterize a subproblem of our dynamic program as $D[i, g(i, m), l]$, the minimal homotopy area for the optimal partial solution S with l edges, such that $g(i, m)$ is the last vertex of S. Note that by definition of the partial solution, $g(i, m - 1)$ is also a gridpoint on e_j. Thus, j is always at least 2. Fig. 6.12 shows examples for partial solutions. The main question is how to compute $D[j, g(j, m), l]$ based on "smaller" instances $D[i, g(i, m'), l']$. We are effectively choosing anchorsegments one at a time. As these occur in order, smaller instances have $1 \leq i < j$ and $l' = l - 1$.

Consider two different gridpoints defined by $g_1 = g(i, m_1), g_2 = g(i, m_2), m_1 < m_2$, for the previous solution ending at edge e_i with length $l - 1$. g_2 is less restrictive for

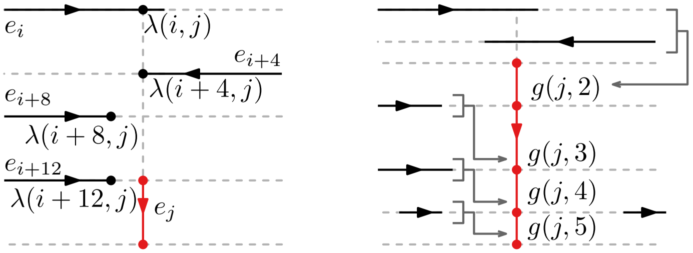


Figure 6.13 (left) Most restrictive predecessor gridpoints for edge e_j, starting at several edges (black lines). Arrows show canonical edge direction, dashed grey lines the induced grid. (right) Edges (black lines) for which a gridpoint on the edge e_j is the most restrictive gridpoint (association via the grey brackets).

the partial solution: it allows for more possible previous partial solutions, including the ones for g_1. Moreover, the homotopy area for a partial solution ending at g_1 that is extended from g_1 to g_2 does not change: the segment between g_1 and g_2 is on the input and cannot incur any homotopy area. Hence, it must be that $D[i, g(i, m_1), l - 1] \leq D[i, g(i, m_2), l - 1]$ for any $m_1 < m_2$. Thus, it suffices to only check the gridpoint furthest along e_i that is still compatible with (j, g) when computing $D[j, g, l]$. We denote this least restrictive predecessor by $\lambda(i, j)$ (see Fig. 6.13, left).

In the same line of argumentation, we can look at the different gridpoints $g' \in G$ in entries $D[j, g', l]$. From the previous argumentation, it follows that we do not need to recompute the minimum homotopy area for every gridpoint over all previous edges for entries $l - 1$. Instead, we only have to compute the minimum homotopy area for the gridpoint g' on e_j that is the *most restrictive* (see Fig. 6.13,right). That is, consider previous edge e_i. We only have to compute the minimum homotopy area for the next partial solution extending from e_i by looking at gridpoint $\gamma(i, j)$ at e_j, or the first gridpoint along edge e_j if $\gamma(i, j)$ is not on e_j. For any following gridpoint along e_j, the minimum homotopy area for any partial solution extending from e_i is exactly the same by the previous argumentation. The minimum homotopy area $D[j, g(i, m), l]$ is then the minimum of the minimum homotopy areas of solutions to gridpoints $g(j, m')$ for $m' < m$, and the homotopy areas for previous entries $D[i, g', l - 1]$ for which $g(i, m)$ is the most restrictive gridpoint.

We obtain the following dynamic program. If $j < l$, the schematization should have more edges than the input so far, thus we set $D[j, g, l]$ to ∞. If $l = j = 1$, we have not created a second anchorsegment and thus $D[j, g, l] = 0$. Also, we set $D[j, g, l] = \infty$ for all $g = g(j, 1)$, since those do not represent valid partial solutions. Otherwise, we set $D[j, g, l]$ to

$$\begin{aligned} D[j, g(j, m), l] \quad = \quad & \min(\\ & \quad D[j, g(j, m-1), l], \\ & \quad \min_{i \in I(j,m)} \sigma(i, j) + D[i, \lambda(i, j), l-1] \\ &) \end{aligned}$$

where $I(j, m)$ is the set of all i such that $i < j$ and $g(j, m)$ is the most restrictive gridpoint for edge e_i.

To efficiently compute the entries of $D[j, g(j, m), l]$ for fixed (j, l), we first precompute extra information to determine the least restrictive predecessor and the most restrictive gridpoints in $O(1)$ time. To do this, we proceed as follows. We assign an index to each horizontal and each vertical gridline of the induced grid, increasing for upwards for horizontal gridlines and towards the right for vertical gridlines. Then, with each edge e_i of the input we associate the indices $I_f(i)$ and $I_l(i)$ of first resp. last gridline that intersects the edge perpendicularly. Additionally, we store the index of the gridline that the edge defines, $I_e(i)$. This way, we can associate gridpoint $g(i, m)$ on edge e_i with the gridline index that intersects it, namely $I_f(i) + m - 1$. We can compute indices I_f, I_l and I_e for all edges of the input in $O(n \log n)$ time.

Next, to compute the least restrictive predecessor for a gridpoint g on e_j given a previous edge e_i, we do the following. We can use the index $I_e(j)$ to find the gridpoint of e_i that is at that gridline index. If such a gridpoint exists, this must be the least restrictive predecessor. If no such gridpoint exists, the last gridpoint of e_i must be the least restrictive predecessor.

Finally, we need to compute most restrictive gridpoints. This essentially is the reverse of the least restrictive predecessor. For a given edge e_j and previous edge e_i, the least restrictive predecessor is either the gridpoint of e_j at the gridline with index $I_e(i)$ or $g(j, 2)$ if the gridpoint at index $I_e(i)$ does not exist or is $g(j, 1)$.

Running time. We can first precompute all $\sigma(i, j)$ values. Per value, we spend $O(n^2 \log n)$ time to compute $\sigma(i, j)$ using the algorithm by Chambers and Wang [30]: we have at most $O(n)$ intersections, as the schematic part consists of two segments. Thus, this precomputation takes $O(n^4 \log n)$ time and requires $O(n^2)$ space to store the results. To compute the gridline indices associated with the edges, we spend $O(n \log n)$ time, which is dominated by the homotopy area computations.

For computing the $D[i, g, l]$ entries for fixed (i, l) and all gridpoints on e_i, we use the prestored gridline indices associated with the edges to lookup least restrictive predecessors and most restrictive gridpoints in $O(1)$ time per lookup. For each gridpoint g, we first compute all homotopy areas for the edges for which g is the most restrictive gridpoint, taking $O(n)$ time for all gridpoints of (i, l). Then, we take the minimum of the stored minimum area and the minimum area of the previous gridpoint for increasing gridpoints in $O(n)$ time. Thus, computation of all cells $D[i, g, l]$ for fixed (i, l) takes $O(n)$ computation time in total. Repeating the dynamic program for every of the $O(n^2)$ possible midpoints yields the our desired result:

6.2.3 Theorem. *Given a simple ortho-polygon P with n vertices and an integer $k < n$, we can compute the schematization S^* of at most k vertices with minimal homotopy area to P in $O(n^4(k + \log n))$ time, if S^* is allowed to self-intersect.*

▸ 6.3 Greedy approximation algorithm for staircases

The algorithm presented in the previous section gives us an exact solution to the schematization problem, at the expense of a rather long, though polynomial, running time and no guarantees on simplicity. To be able to make the algorithm more feasible in practice, we consider approximating the optimal solution. To this end, we propose a very intuitive greedy algorithm. We draw inspiration from Buchin, Meulemans, Van Renssen and Speckmann [88, 22], who also consider schematization of orthogonal polygons. They describe different configurations that can occur in orthogonal polygons and define moves to eliminate the configurations, which reduces the complexity of the polygon. We choose to apply the moves one at a time, and do not take into account the area-preservation that is present in their approach. Moreover, for our setting, we see that we can represent the subhomotopy for two consecutive edges of the schematization as a sequence of eliminations of so called C-configurations when considering the concatenation of the subcurves of the homotopy (see Fig. 6.14), thus connecting their moves to the minimum homotopy area. Eliminating a C- or S-configuration can be seen as moving the middle edge until one of the other edges is eliminated, as well as the middle edge (e_i and e_{i+1} in Fig. 6.14). Such an elimination then defines a *move*.

We can define a move formally by first interpreting the orthogonal polygon as a list of alternating x and y coordinates, that is $P = \langle x_1, y_1, \ldots, x_{n/2}, y_{n/2} \rangle$, such that vertex v_i has coordinates $(x_{\lceil (i+1)/2 \rceil}, y_{\lceil i/2 \rceil})$. Let e_i and e_{i+1} be the edges that define some move m. The coordinates of the vertices of these edges are four consecutive elements in the xy-representation of P, we denote these by $p_i, \ldots, p_{i+3}$. We now define the

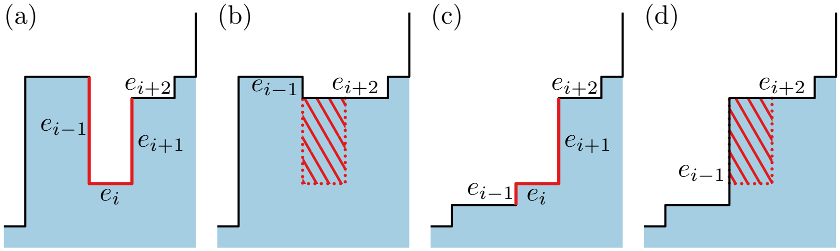


Figure 6.14 (a) A C-configuration (green lines) that we can eliminate. (b) The resulting schematization. Here the hatched area is the minimum homotopy area between the before and after state, the dotted outline shows the boundary of the homotopy, which we call the box of the move. (c) An S-configuration can also be eliminated by moving the middle edge resulting in (d).

resulting polygon P' after applying m on polygon P to be the original polygon P with p_{i+1}, p_{i+2} eliminated from the xy-representation[1]. Intuitively, we can see this as follows: take the axis-aligned box with sides e_i and e_{i+1} (we refer to this box as the *box of the move*). We now replace e_i and e_{i+1} by the other two sides of this box and eliminate any collinear vertices and degenerate edges. From the definition that uses the xy-representation, it is also easy to see that any schematization still lies on the induced grid, since the x- and y-coordinates that define the schematization are a subset of the ones present in P. Note that the consecutive elements in P that define the move only have to be consecutive in the schematization to which we apply the move.

Greedy algorithm With the move now formally defined, we can present the greedy algorithm we consider in this section. We start with a polygon P of complexity n and want to schematize it to a complexity of k. The key idea is that we pick $(n - k)/2$ moves such that we get close to the optimal minimum homotopy area schematization. For this, we propose to select the move of the current schematization that has smallest *area*. We define the area of the move m as the area swept by moving edge, which is the area of the axis aligned box with as sides e_i and e_{i+1} that together define move m (see Fig. 6.14). Note that this is simply the area of the box of the move. We then apply this move with minimal move area, resulting

---

[1] Note that in principle degenerate edges may arise, though we may assume unique x and y coordinates for the edges of P or simply eliminate the degenerate edges as well

in a new schematization. For this new schematization, we then again select the move with smallest area. We continue this process until the schematization is of the right complexity. We note that the greedy algorithm is very similar to the approach by Buchin, Meulemans, Van Renssen and Speckmann [88, 22], though we do not take into account area preservation nor simplicity and by our choice of moves the resulting schematization is on the induced grid of the input polygon.

Analyzing this greedy approach proves difficult for simple polygons, since there is no direct obvious lowerbound on the minimum homotopy area for optimal schematizations that we can relate the minimum homotopy area of the greedy approach to. Thus, to gain more insight we analyze the performance on orthogonal *staircases*: orthogonal polylines that are xy-monotone. A simple polygon can be interpreted as a sequence of staircases, albeit not all oriented in the same direction, thus staircases could provide more insight into the problem for polygons. We consider staircases where the first edge is horizontal, the last edge is vertical, and the first of last edge are much larger than any intermediate edge, such that they are never considered in a move.

In the remainder of this section, we show that our proposed greedy algorithm approximates the optimal approach by a constant factor, when we consider uniform staircases. We conjecture that the same holds for non-uniform staircases and for potentially simple polygons.

▸ 6.3.1 Properties staircase schematizations under the greedy and optimal algorithm

We briefly show some key properties of staircase schematizations that we need for our main result.

We say that an edge e of the schematization is *supported* by another edge e' if e' is a subsegment of e. We now have the following:

6.3.1 **Lemma.** *Every edge in a move-based schematization is supported by an input edge.*

Proof. We can prove this by induction on the number of moves. Initially, before any move is applied, all edges of the schematization coincide with the input, thus the claim trivially holds.

Then, suppose it holds for all previous move and consider a next move m. When we apply m, we remove the two edges that define m and extend the two edge before and after these edge. This means that we only need to consider whether the two extended

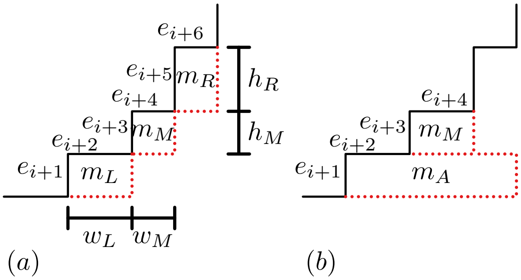


Figure 6.15 (a) Sketch for the greedy property for a move m_M we are about to take, relative to the neighboring moves m_L and m_R. (b) Setup for a later move m_A relative to some move m_M.

edges are still supported by the input. By definition, the edges are supported when an edge of the input is a subsegment of the current edges. By the induction hypothesis, we know that all edges are supported before the move. Since we are only extending the edges, the new edges are strict supersegments of the original edges. It must hold that the new edges are supported by an input edge. Thus the claim still holds after a next move. So we can conclude by induction that the claim must hold. □

Note that this lemma does not directly apply when the polyline is not strictly xy-montone, since some edges of the schematization may contract when applying the move.

6.3.2 Lemma. *Let $\langle m_1, \dots, m_l \rangle$ be the sequence of moves to get from input polygon P to the target schematization S using the greedy algorithm. The area of m_i is less than or equal to the area of m_j for $i < j$.*

Proof. Applying a move strictly increases the length of two edges in the schematization, as we argued in the proof of Lemma 6.3.1. Since the area of the moves are simply the area of the boxes of the edges defining the moves, i.e. the multiplication of the edge lengths of the defining edges, the area of moves of the next schematization can only be the same or larger. □

For argumentation purposes, let w_A and h_A be the width and height of the axis aligned box of the edges that define a move m_A, whose area is $A_A = w_A h_A$.

6.3.3 Lemma. *Consider six consecutive edges $e_{i+1}, \dots e_{i+6}$ of a staircase, where e_{i+1} is a horizontal edge, and let m_L, m_M, m_R be the moves defined by the three consecutive edge pairs of these six edges (see Fig. 6.15a). Suppose we are about to apply move m_M. The following must hold*

$$w_M \le w_L, \qquad h_M \le h_R.$$

Proof. Since m_M is the greedy move we are applying, any other move has cost at least the cost of m_M, thus the area of the box of m_M. In particular, then, the area of the move between m_L and m_M is greater than or equal to m_M. It is easy to see that the area of this box is $h_M w_L$, since it shares edges with m_L and m_M. It then follows that $h_M w_M \le h_M w_L$, and the first claim follows. The second claim we can argue in a symmetrical fashion, now considering the move between m_M and m_R. □

Note that a similar lemma applies when the first edge is vertical, where we now need to exchange w and h in the result.

We can extend the lemma by considering actual moves that are applied later:

6.3.4 Lemma. *Consider a move m_M that we are about to apply, and let S be the current schematization. Suppose there is a move m_A that depends on m_M and whose edges overlap with the two edges directly before the edges defining m_M in S (see Fig. 6.15b). We claim*

$$A_M \le A_A - h_A w_M$$

Proof. By Lemma 6.3.3, we know that the area of m_M must be smaller than the area of the box defined by e_{i+1}, e_{i+2}. Moreover, since m_A depends on m_M, its width w_A must be greater than or equal to $|e_{i+2}| + |e_{i+4}| = |e_{i+2}| + w_M$. Finally, its height must be greater than or equal to $|e_{i+1}|$ by construction. Thus, we have that

$$\begin{aligned} A_M &\le |e_{i+1}| \cdot |e_{i+2}| \\ &\le h_A |e_{i+2}| \\ &\le h_A(w_A - w_M) = A_A - h_A w_M \end{aligned}$$

Thus, the lemma follows. □

▸ 6.3.2 Minimum greedy move size

To be able to show that our greedy approach approximates the optimal schematization, we need to relate the minimum homotopy area to the input from the greedy

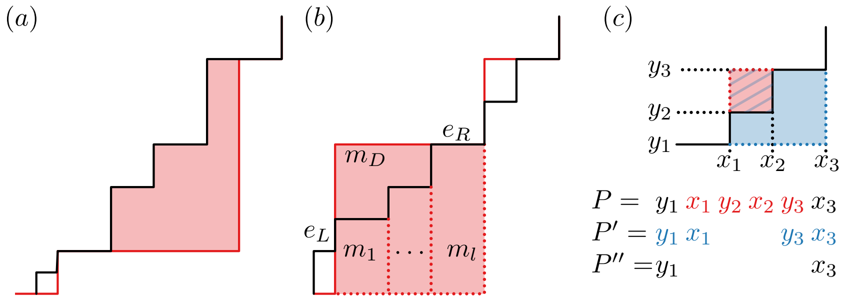


Figure 6.16 (a) A bracket and its associated input. The interior of the bracket is shown with the green region. (b) Applying m_D gives a higher maximum area, namely the total red area, than applying $m_1, \dots, m_l$ instead. (c) The blue move (with coordinates y_1, x_1, y_3, x_3, blue area plus hatched blue area) depends on the red move (with coordinates x_1, y_2, x_2, y_3, red area).

schematization to the areas of the moves that occurred in some way. To this end, we first define a *bracket* of a schematization to be the subsegments of two consecutive edges that are connected and do not intersect any part of the input (see Fig. 6.16a). By the *associated input of the bracket* we mean the input that is between the edges of the input that the edges of bracket overlap with.

6.3.5 **Lemma.** *Consider a bracket of a schematization and its associated input. We can minimize the maximum area over all moves needed to simplify to the bracket by only applying moves that result in a schematization that lies closer to the corner of the bracket.*

Proof. Assume w.l.o.g. that the bracket consists of a horizontal followed by a vertical segment. This then means that its associated input lies above and to the left of the segments of the bracket. Now suppose we applied some move that did not result in a schematization that is closer to the bracket. This means that the move locally moved the previous schematization edges upwards and to the left, resulting in a schematization that has a corner that lies above the input. But since we ultimately need to move the input to the bracket of the schematization, this corner must be moved downwards at some point. Consider now the schematization just before we move this corner downwards by some move m_D (see Fig. 6.16). Let e_L and e_R be the input edges that support the edges that define move m_D. If we were to apply all moves left of e_L and right of e_R that were taken before m_D exactly in the same

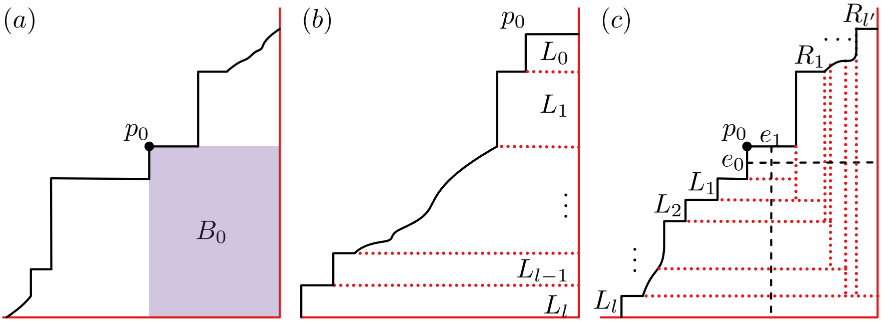


Figure 6.17 (a) We consider a corner p_0 and the area A of the box B_0 between p_0 and the bracket corner. (b) Special case where all moves are to the left of p_0. (c) Defining L and R type moves using rays starting from edges e_0 and e_1 that define move m_0. This move eliminates corner p_0.

order, but instead of applying the moves that lead up to m_D, we would apply the moves from the input downwards (moves $m_1, \dots, m_L$ in Fig. 6.16), their total area is less than the area of move m_D. It then immediately follows that replacing the moves that lead up to m_D and m_D itself with these downwards moves uses strictly lower maximum move area. □

To more easily analyze the moves, we introduce the notion of a *dependent move.* Suppose we are simplifying polygon P to some target polygon S with a sequence of moves $\langle m_1, \dots, m_l \rangle$. We say that a move m_j *depends* on a move m_i if $i < j$ and m_i must be applied to be able to apply m_j. More specifically, consider the four elements of the xy-representation of P that define move m_j, $c_i, \dots, c_{i+3}$. m_j can only be applied when these coordinates are consecutive in the current schematization. If m_j depends on move m_i, the elements of m_i must lie between or on one of the pairs $(c_i, c_{i+1}), (c_{i+1}, c_{i+2}), (c_{i+2}, c_{i+3})$ in P, which hence must have been applied for $c_i, \dots, c_{i+3}$ to be consecutive in the current schematization (see Fig. 6.16c).

6.3.6 Lemma. *Consider a bracket of a greedy schematization and let A be the area of some box between the bracket and its associated input. It must be that a move of area at least cA occurred during the schematization to this bracket, with $c = \frac{1}{2}$.*

Proof. By Lemma 6.3.5 we know that the maximum area over all moves in a bracket is minimized by only considering moves towards the bracket, hence we only consider these moves. We assume w.l.o.g. that the bracket consists of a horizontal and then a vertical segment, the complementary situation is completely symmetrical. This then means that we only consider moves that locally move the schematization downwards and to the right.

So consider the situation as shown in Fig. 6.17a. Let p_0 be some corner of the associated input of the bracket. The area A is then the area of the box B_0 between the corner of the bracket and p_0. We now consider the schematization of the input at the point where we are about to move corner p_0 by a move m_0. We then only consider moves that depend on m_0. Since this set of moves contains the last move in the bracket, which by Lemma 6.3.2 is the move with largest area, it must be that the moves that do not depend on m_0 do not influence c.

We can split the moves that depend on m_0 into two types: $L_1, \dots, L_l$, the moves such that the downwards ray, starting at the middle of e_1 intersects its box, and $R_1, \dots, R_{l'}$, the moves such that the rightwards ray, starting at the middle of e_0 intersects its box (see Fig. 6.17c). For simplicity of notation we set $L_0 = R_0 = m_0$.

Suppose that the moves that depend on m_0 are only the moves $L_1, \dots, L_l$, where L_l is adjacent to the bracket corner(see Fig. 6.17b). Since there are no moves of type R, it must be that all dependent moves are adjacent to the vertical bracket edge. Let w_i, h_i denote the width and height of move L_i. By Lemma 6.3.4, the following must hold

$$A_i \leq A_{i+1} - h_{i+1} w_i, \quad \text{for all } 0 \leq i < l$$

Repeatedly applying this inequality gives

$$A_0 \leq A_l - \sum_{i=0}^{l-1} h_{i+1} w_i = A_l - \sum_{i=1}^{l} h_i w_{i-1}, \quad \Longleftrightarrow \quad \sum_{i=1}^{l} h_i w_{i-1} \leq A_l - A_0$$

By construction, each move L_{i+1} depends on move L_i, hence it must be that $w_i \leq w_{i+1}$. Finally, we know that $A = w_0 \sum_{i=0}^{l} h_i$. Combining these relations, we can bound A as follows.

$$A = w_0 \sum_{i=0}^{l} h_i = w_0 h_0 + w_0 \sum_{i=1}^{l} h_i \leq A_0 + \sum_{i=1}^{l} h_i w_{i-1} \leq A_0 + A_l - A_0 = A_l$$

Hence there must have been a move, L_l, of area A_l that was greater than or equal to A. Thus the lemma holds in this case with $c = 1$. Note that if we only have $R_1, \dots, R_{l'}$, and no L moves, we can make a symmetrical argument.

We now consider the situation where there are moves of both types, so we have moves $L_1, \dots, L_l$ and $R_1, \dots, R_{l'}$ for $l, l' \geq 1$, such that they all depend on $m_0 = L_0 = R_0$. To be able to prove the lemma for this situation, we first prove another claim.

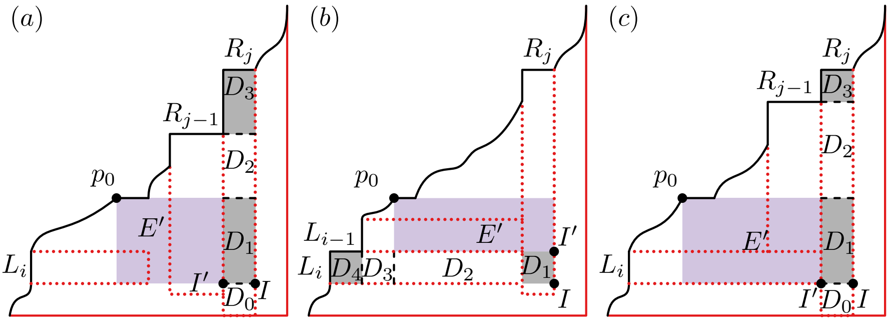


Figure 6.18 The three different cases we consider for showing that the area $E(i,j)$ remains less than the sum of the areas $A_L(i,j)$ and $A_R(i,j)$. Note that the configurations are schematic versions, the proportions are not correct. I' marks the intersection of the configuration before L_i, R_j. Red dotted boxes mark the moves, the purple region marks E', the region of the previous moves.

Consider two moves L_i and R_j. By extending the bottom of the box of L_i to the right and the right side of the box of R_j downwards, we get an intersection point I. Let $E(i,j)$ be the box between I and p_0. We now claim that the area of $E(i,j)$ is at most the sum of the areas of the parts of L_i and R_j that lie to the left of and above I. We denote the areas of these parts of L_i and R_j by $A_L(i,j)$ resp. $A_R(i,j)$, such that our claim becomes $E(i,j) \leq A_L(i,j) + A_R(i,j)$ for any i,j for which the above holds. We prove this claim by structural induction on i and j.

First, however, we show that the intersection point must always be on the box of one of the two moves. Suppose this is not the case. The moves are then disjoint in both x and y coordinates, or they may touch in a single point. This implies that, in the xy-representation of the input polygon P, the moves share a single coordinate, or they are separated. Since both are dependent on m_0, the coordinates of m_0 must be between one of their consecutive coordinate pairs. But since the moves do not overlap in P, it cannot be that both are dependent on m_0, giving a contradiction.

Consider first the base case where $i = j = 0$. The areas $A_L(i,j)$ and $A_R(i,j)$ and $E(i,j)$ are all the same as the area of move m_0. Thus the claim trivially holds.

Next, we prove that the claim holds, assuming that one step back the claim held. For this, assume w.l.o.g. that the intersection point I always lies on the box of R_j. We now consider three separate cases (Fig. 6.18):

(a) L_i and R_j do not touch, we only have to consider the case where the previous move was R_{j-1}

(b) L_i and R_j do touch, and the previous move was L_{i-1}.

(c) L_i and R_j do touch, and the previous move was R_{j-1}.

First, we look at case (a). We only have to consider the case where the claim held for L_i, R_{j-1}, since there must always be one or more moves before R_j that close the gap between L_i and R_j. This cannot be done using any moves of type L, since those are either above or below L_i by definition, and thus cannot close the gap to the right of L_i. Now, we assume that the claim held for L_i, R_{j-1}, we now show that it holds for L_i, R_j. Let E' be the enclosed area of L_i, R_{j-1}. We need to show that the enclosed area $E(i, j)$ of L_i and R_j satisfies $E(i, j) \leq A_L(i, j) + A_R(i, j)$. We can vertically split the move R_j in four regions D_0, D_1, D_2, D_3 such that the union of these regions is R_j, where we split on the horizontal lines defined by the y-coordinate of the bottom of L_i, the y-coordinate of p_0 and the y-coordinate of the top of R_{j-1}. D_0 can be an empty region. By definition now, $E(i, j) = E' + D_1$. Also, from Lemma 6.3.4, we know that the area of R_{j-1}, $A(R_{j-1})$, is less than the area of region D_3(Fig. 6.18a). Then, we have that

$$\begin{aligned} E(i,j) = E' + D_1 &\leq A_L(i, j-1) + A_R(i, j-1) + D_1 \\ &\leq A_L(i, j-1) + A(R_{j-1}) + D_1 \\ &\leq A_L(i, j-1) + D_1 + D_3 \\ &\leq A_L(i, j) + A_R(i, j) \end{aligned}$$

where we use that in this case, $A_L(i, j) = A_L(i, j-1)$. Thus, the claim still holds.

We now consider case (b). Here, we consider that the claim held for L_{i-1}, R_j and prove that it holds for L_i, R_j. We split L_i into regions D_2, D_3, D_4, using the vertical lines with as x-coordinate the left x-coordinate of L_{i-1}, and the x-coordinate of p_0. In addition, let D_1 be the region of R_j that lies between the top and bottom of L_i (see Fig. 6.18b). We see that $E(i, j) = E' + D_1 + D_2$. We also know that $A_L(i-1, j) = L_{i-1} \leq D_4$. Finally, $A_R(i-1, j) + D_1 = A_R(i, j)$. Thus, we have that

$$\begin{aligned} E(i,j) = E' + D_1 + D_2 &\leq A_L(i-1, j) + A_R(i-1, j) + D_1 + D_2 \\ &\leq A_R(i-1, j) + D_1 + D_2 + D_4 \\ &\leq A_R(i, j) + D_2 + D_4 \\ &\leq A_L(i, j) + A_R(i, j) \end{aligned}$$

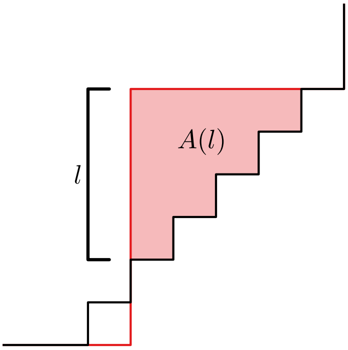


Figure 6.19 Area $A(l)$ for a bracket with a side length l. The lengths of the sides of the bracket are the same when the staircase is uniform.

Finally, we consider case (c). Here, we apply the same split as with case (a) for R_j, resulting in regions D_0, D_1, D_2, D_3. We can then apply the exact same argumentation as for case (c).

By structural induction now, we can conclude that $E(i,j) \leq A_L(i,j) + A_R(i,j)$ for any valid i, j. In particular then, the claim holds for $i = l$ and $j = l'$, the last L move and the last R move. Since both moves border the bracket, $A_L(i,j)$ and $A_R(i,j)$ are the areas of these moves. Also, $E(l, l')$ is exactly the area A that we want to bound.

The intersection point for the moves is exactly the corner of the bracket, hence one of these moves has to be the last move before arriving at the bracket. This move must then be larger than the other one. Assume w.l.o.g. that this move is L_l of size $A(L_l)$. It then follows that $A = E(l, l')$, the area between p_0 and the bracket corner, is at most $2A(L_l)$. So, there must have been a move, move L_l, with area at least $\frac{1}{2}A$, proving the lemma with $c = \frac{1}{2}$. □

6.3.7 Corollary. *Let A_{max} be the area of the largest box in a greedy bracket. There must have been a move of $\frac{1}{2}A_{max}$ in the bracket.*

▸ 6.3.3 Approximation factor for uniform staircase

Using the insights from the previous section, we now show that the greedy algorithm is a constant factor approximation algorithm, if we restrict our attention to uniform staircases. So consider a uniform staircase of complexity n, consisting of an infinitely long first and last edge and $n - 2$ edges of unit length. We claim the following.

6.3.8 Lemma. *The optimal schematization of size k under the minimum homotopy area for a uniform staircase of complexity n has minimum homotopy area of at least $\frac{1}{2}(k-1)l'(l'+1)$ with $l' = \frac{1}{2}\frac{n-k}{k-1}$.*

Proof. Consider a bracket for a uniform staircase. Since all edges (except the first and last) are of unit length, the sides of a bracket are of equal size (see Fig. 6.19). Moreover, given a side length l of a bracket, it is easy to see that the bracket contributes $\frac{1}{2}l(l+1)$ to the homotopy area.

The total minimum homotopy area then is the sum of the contributions of all brackets, giving

$$A(O) = \frac{1}{2}\sum_{b \in B} l_b(l_b+1) \tag{6.1}$$

with B all brackets of the current schematization (one per move) and l_b the length of a side of the bracket. Note that there are $k-1$ moves, and thus brackets. Since the total side length is fixed and we are minimizing a sum of squares over the lengths, it must be that $A(O)$ is minimized when all l_b are approximately equal. It is maximized when one bracket has a maximal side length while all others are assigned zero length.

Excluding the start and end, the input staircase has width and height $\frac{1}{2}(n-2)$. Since the schematization has the same width and height as the input, it must be that

$$\frac{1}{2}(k-2) + \sum_{b \in B} l_b = \frac{1}{2}(n-2). \tag{6.2}$$

To distribute the lengths as evenly as possible, we set the base length for l_c to $l_{\text{base}} = \left\lfloor \frac{1}{2}\frac{n-k}{k-1} \right\rfloor$ and distribute the remainder of unassigned length over the corners, giving at most $k-2$ corner lengths of $l_{\text{base}}+1$. Taking the sum over all the brackets of these side lengths, we get the optimal schematization area

$$A(O) = \frac{1}{2}\sum_{b \in B}\left(\left\lfloor \frac{1}{2}\frac{n-k}{k-1} \right\rfloor + \delta_b\right)\left(\left\lfloor \frac{1}{2}\frac{n-k}{k-1} \right\rfloor + \delta_b + 1\right)$$

where δ_b is 1 for at most $k-2$ corners and zero otherwise.

By definition of the sum of squares, having all lengths exactly equal will result in a lower value than having some values being slightly larger, when we force the sum of all values that we are squaring to be a fixed value. Thus, it must be that

$$A(O) \geq \frac{1}{2}\sum_{b \in B}\frac{1}{2}\frac{n-k}{k-1}\left(\frac{1}{2}\frac{n-k}{k-1} + 1\right) = \frac{1}{2}(k-1)l'(l'+1)$$

with $l' = \frac{1}{2}\frac{n-k}{k-1}$. In the last step, we use that the total number of brackets is $k-1$. □

We now proceed to show that the greedy approach results in a constant factor approximation for the optimal schematization.

6.3.9 Theorem. *The resulting schematization S_G of the greedy algorithm of a uniform staircase is a 36-approximation of the optimal schematization S_O.*

Proof. Let the *length* of a move be the sum of the lengths of the edges that define the move. Consider all moves on a single side of the staircase (either top or bottom). There are at most $k/2$ moves on one side of the staircase. However, the total length of the staircase remains the same under moves. Thus, the total length of the staircase (ignoring the first and last edge) is $n-2$. Since the sum of lengths of all moves on one side is exactly the total staircase length, it must be that there exists a move with maximum length $\frac{n-2}{k/2}$. Given a fixed length for a move, its area is maximized if we make the edges defining the move of equal size. Thus, its area is at most $\left(\frac{1}{2}\frac{n-2}{k/2}\right)^2 = \left(\frac{n-2}{k}\right)^2$.

With this bound and the result in Lemma 6.3.2, we know that all moves applied to get to the current schematization must have had an area of at most $\left(\frac{n-2}{k}\right)^2$. By Lemma 6.3.6, it must then also hold that the largest box between the input and any bracket of the greedy schematization is at most $2\left(\frac{n-2}{k}\right)^2$.

Consider a bracket with side length l. The largest box that fits in a bracket of a uniform staircase has close to equal sides and has an area A_{Box} of at most $\left\lceil\frac{l+1}{2}\right\rceil \cdot \left\lfloor\frac{l+1}{2}\right\rfloor$. Let A_B be the area between the bracket and its associated input, which must be $\frac{1}{2}l(l+1)$. It then follows that

$$A_{Box} = \left\lceil\frac{l+1}{2}\right\rceil \cdot \left\lfloor\frac{l+1}{2}\right\rfloor \geq \frac{1}{2} \cdot \frac{1}{2} l(l+1) = \frac{1}{2} A_B,$$

provided that l is a positive integer. From the inequality, we see that the total area between the bracket and its associated input is at most twice the area of the largest box that fits between the bracket and its associated input. Using the previous bound on this largest box, it follows that the area between any bracket and its associated input is at most $4\left(\frac{n-2}{k}\right)^2$.

From Lemma 6.3.8, we know a lowerbound on the optimal schematization minimum homotopy area. We upperbound the minimum homotopy area for the greedy approach using $k-1$ times the area of the largest area for a single bracket. We now take the ratio of this area and a lowerbound on the total minimum homotopy area of the optimal solution, to get

$$\frac{A(G)}{A(O)} \leq \frac{k-1}{k-1}\frac{4\left(\frac{n-2}{k}\right)^2}{l'(l'+1)/2} = 8\frac{\left(\frac{n-2}{k}\right)^2}{l'(l'+1)} = 8\frac{\left(\frac{n-2}{k}\right)^2}{\frac{1}{2}\frac{n-k}{k-1}(\frac{1}{2}\frac{n-k}{k-1}+1)}.$$

Let $\rho_1(n,k)$ be the rightmost expression. Taking the partial derivative of $\rho_1(n,k)$ to k yields

$$\frac{\partial\rho_1}{\partial k} = \frac{64(k-1)(n-2)^2}{k^3(k-n)^2(k+n-2)^2}(3k-3k^2+k^3+n(n-2))$$

Whether the partial derivative is positive or negative is then solely determined by $3k-3k^2+k^3+n(n-2)$. Assuming $n \geq 4$ and $k \geq 2$, this is always positive, thus using higher k upperbounds $\rho_1(n,k)$ for all lower k regarding the ratio of areas of symmetric difference.

We can give an alternative upperbound on the ratio of the two areas. We reuse the bound for the maximum area for any bracket of the greedy schematization. We know that the number of moves applied is $\frac{n-k}{2}$, since any move removes two edges of the schematization it is applied to. Then, it must be that the number of non-empty brackets of the greedy schematization must also be at most $\frac{n-k}{2}$, since a non-empty bracket can only be created by one or more moves that the associated move of the bracket depends on. Hence, we can upperbound $A(G)$ by $\frac{n-k}{2}\cdot 4\left(\frac{n-2}{k}\right)^2$. Additionally, we can trivially lowerbound $A(O)$ by $\frac{n-k}{2}$, since by construction, the minimum homotopy area for staircases is non-decreasing and thus each move constributes at least area 1 to the minimum homotopy area. Then, we have the second ratio

$$\frac{A(G)}{A(O)} \leq \frac{(n-k)/2}{(n-k)/2}\cdot 4\left(\frac{n-2}{k}\right)^2 = 4\left(\frac{n-2}{k}\right)^2$$

Let the rightmost expression be $\rho_2(n,k)$. We now have

$$\frac{\partial\rho_2}{\partial k} = -\frac{8(n-2)^2}{k^3}$$

which is negative for positive k.

Since both upperbounds should hold, we can derive a new bound by taking the minimum of $\rho_1(n,k)$ and $\rho_2(n,k)$ as an upperbound, that is

$$\frac{A(G)}{A(O)} \leq \min(\rho_1(n,k), \rho_2(n,k)).$$

We can derive that $\rho_1(n,k)$ and $\rho_2(n,k)$ intersect at $k = \frac{4-n}{3}$ and $k = \frac{2+n}{3}$. Since we assume $k \geq 2$, the first result is not of interest. Now, since $\rho_1(n,k)$ is strictly increasing with higher k and $\rho_2(n,k)$ is strictly decreasing for $k \geq 2$ it must be that the maximum of the two functions is along the line $k = \frac{2+n}{3}$. By filling in k in $\rho_2(n,k)$ we see that the maximum value is

$$\min(\rho_1(n, \frac{2+n}{3}), \rho_2(n, \frac{2+n}{3})) = 36\frac{(n-2)^2}{(n+2)^2} \leq 36.$$

Hence, we have that $\frac{A(G)}{A(O)} \leq \min(\rho_1(n,k), \rho_2(n,k)) \leq 36$. Thus the greedy algorithm results in a schematization that is a 36-approximation of the optimal schematization. □

With this theorem, we generalize the result a bit. For this, we define a *non-uniformly scaled* staircase as a staircase whose horizontal edges are of unit length (except for the first edge), and whose vertical edges are of length y (except for the last edge). Then, we have the following generalization

6.3.10 Theorem. *The resulting schematization S_G of the greedy algorithm of a non-uniformly scaled staircase is a* 36*-approximation of the optimal schematization S_O.*

Proof. Consider a schematization produced by the greedy approach on the uniform staircase of the required target complexity. Scaling the areas of the moves that together produced the schematization does not change the relative order of the moves. Thus the greedy approach would produce the same result on the non-uniformly scaled staircase, only scaled by the factor of the non-uniformly scaled staircase.

For the optimal schematization, we see that a bracket still spans the same number of edges of the input horizontally and vertically, but now the vertical edges are scaled by y. Careful analysis of the area of a bracket then shows that the area of a single bracket is exactly the scaled area of the uniform bracket. Hence, the area of the optimal schematization is also simply scaled.

Since the approximation factor is the ratio of the areas of S_G and S_O, the non-uniform scaling factor is eliminated, thus we get the same resulting factor in this setting. □

▸ 6.4 Conclusion

In this chapter, we have considered different ways of improving schematization of orthogonal polygons by considering other distance measures and a move based model that potentially alleviate issues with other measures. We proposed an algorithm for computing orthogonal schematizations under the minimum homotopy area and showed that a very simple greedy algorithm exists for staircases that approximates this schematization within a constant factor.

We gave an algorithm to compute the optimal schematization for orthogonal polygons under the minimum homotopy area, allowing self-intersections. It would be interesting to see if this algorithm generalizes to other C-regular settings or even for arbitrary simple polygons. The challenge is then to find a new canonical form, since the it is not directly obvious how and if the canonical form generalizes to these settings. Similarly, one may consider the schematization of planar subdivisions, and see if we can generalize the algorithm to that setting, for orthogonal polygons or more general polygons if possible.

As we showed when analyzing the characteristics of the homotopy area, we can construct examples where simplicity of the input polygon is not preserved and we actually can get the asymptotically worst case number of intersections. However, these examples hinge on the fact that the polygon contains very narrow parts. This seems to suggest that if we bound the narrowness of these corridors in the polygon, better bounds may be achieved. This may then be used to try to formulate an algorithm that does actually maintain simplicity under these assumptions. Alternatively, a hybrid measure could be beneficial in this particular case, where the area of these small corridors is used to weigh particular moves. This then resembles using the erosion thickness of the medial axis of a polygon for simplification [129]. We envision that by considering simultaneous moves, we may be able to achieve a similar simplification when considering moves.

For future work, one can consider the non-uniform staircases and simple polygons under the move-based model. In particular, though we highly suspect a constant factor approximation for the greedy approach when considering non-uniform staircases, we have not yet found a fully viable proof. A natural extension beyond staircases is to look at skyline polygons and finally simple polygons. A key challenge there is that move costs for the greedy approach may also decrease. Hence, it will be more difficult to argue what the behavior of greedy exactly is without having specific instances.

In addition to extending the approach, we aim to improve the approximation factor for the uniform staircase. From exploratory experiments of the schematizations of staircases using the greedy approach, it seems likely that a large class of staircases exhibit a better approximation factor, where 2 seems to be the most likely candidate. This, however, requires even deeper insight into the problem and carefully formulated proof strategies. We leave this for future work.

Chapter 7

Conclusion and Future Work

In this thesis, we discussed several approaches where we use contextual data to enrich algorithms and derive new information that can be used for further analysis to understand movement. We in particular consider the road network, physical properties of entities and the local traffic situation as context to our algorithms.

▸ 7.1 Main results

Using physical properties of the entities of the trajectories as contextual data, we proposed a new outlier detection method that finds inconsistent measurements in a trajectory. In particular, we leverage the maximum speed and acceleration of entities to determine what possible paths they could have traversed, and then see if this concurs with the measurements that are present in the trajectory. We propose a method for concatenable consistency, where we use the maximum speed of an entity to efficiently find outliers in 1D and 2D. Furthermore, we propose a 1D method for detecting outliers that also incorporates acceleration bounds, thus resulting in conditional concatenability. We extensively evaluate the proposed methods against base line algorithms and investigate the sensitivity of the parameters.

We then continued using physical properties of the entities, but also introduced the road network and local traffic laws to expand the contextual data we had for the entities. Using this contextual data, we proposed a map matching algorithm that takes into account the physical properties of the entities, thus limiting the potential paths an entity could have taken. By leveraging a consistency check for paths through the network, we were able to use a k-fastest paths approach for generating potential map matched paths and eliminate inconsistent paths, which allowed us to afterwards heuristically select a good path. We argued that this approach is beneficial when we need a certificate that the map matched path obeys certain physical properties.

We investigated using contextual information of the local traffic situation to see if we could leverage trajectory data to reconstruct routes in a road network. For this, we used local vehicle counts, recorded at fixed positions in the road network, as contextual data, interpreting our trajectories as a representative set for the traffic in the network. We then required the reconstructed routes to be realistic, i.e. that they are close to the representative trajectories under the Fréchet distance. We showed that trying to reconstruct routes such that all counts are matched is NP-hard under certain optimization criteria. Thus we developed heuristics for solving the problem. Experimentally, we showed that our proposed heuristic guarantees realism of the reconstructed routes while requiring more running time and in general matching less of the traffic counts, whereas baseline approaches are faster, but do not result in realistic routes though they may better match the traffic counts.

To be able to visualize major routes in a trajectory data set, we leveraged the road network as context to schematize the trajectory data set and road network in a coordinated fashion. That is, we proposed a pipeline that ensured that any simplification of the road network is reflected in the trajectory data set, such that a mapping between the road network and trajectories remain across the simplification operations. After significant simplification of the road network, we can look for major routes by finding bundles, which are routes that are long and are a subtrajectory of a high number of trajectories in the data set. We explored the results of the pipeline on two real-world data sets.

Finally, we proposed to use the minimum homotopy area as a similarity measure for schematizing orthogonal polygons. This measure retains more of the intuitive topology of the input in the schematization. We provided an algorithm to compute the optimal schematization under the minimum homotopy, where we do not enforce simplicity of the schematization when the input is simple. Due to the high running time for this algorithm, we considered a greedy algorithm to approximate the optimal schematization. We then showed that this algorithm is actually a constant factor approximation when we consider orthogonal staircases. We conjectured that this result generalizes to non-uniform staircases.

▸ 7.2 Future work

We investigated different approaches of using contextual data to process and visualize trajectory data for understanding mobility patterns. We next explore directions that may be of interest for future research.

▸ 7.2.1 Using the temporal component of trajectories and context

In the first two chapters of this thesis, we truly leverage the fact that our trajectories are spatio-temporal data: we can use the timestamps in conjunction with the locations of the measurements to derive where the entity could have been at a next timestamp, assuming it was somewhere at some time previously. However, in the chapters regarding route reconstruction and coordinated schematization for trajectory visualization, we mainly ignore the temporal component of the data, reinterpreting the trajectories as directed polylines. The general question when analyzing trajectories is how important the temporal component is. If we are interested in routes only, then the dynamics of the trajectories are not that important, and the interpretation as polylines is warranted. When the dynamics are of concern however, the question is then how to use the temporal component in conjunction with the spatial component: is the behavior that we expect periodic (e.g. rush hours during weekdays) or is it not? Do we expect similar behavior to be dependent on the location? Context then plays an important role in distinguishing between these different situations.

▸ 7.2.2 Anonymization, privacy and contextual data

One of the challenges when working with trajectory data derived from human mobility is that it is very privacy-sensitive: full, unprocessed trajectories contain a lot of personal information or can lead to a lot of personal information when combined with the appropriate sources. This is one of the reasons why accurate open data sets are scarce. To be able to publish trajectory data, algorithms have been developed that anonymize the trajectories, making it harder to link multiple collected trajectories to the same person or even deriving what locations of interest a person visited. However, on the other hand we want to derive meaningful information from the trajectory data, expanding our understanding of the underlying mobility of people. Thus, there is a balance to be struck between how much privacy is guaranteed and the amount of meaningful knowledge we can extract from the trajectory data. Looking at context-aware algorithms, it is then also important to know how much we can still infer from the trajectory data, given certain contextual information, knowing that the trajectory data is anonymized. Or, in the reverse case, it is important to investigate how much we can infer from contextual data sources, such that we can build policies on this knowledge to avoid a potential breach of privacy.

▸ 7.2.3 Realistic input models

As we saw in this thesis, the problems we aim to solve for movement analysis often are NP-hard in the general case. However, since we have a very specific application area, it could make sense to leverage realistic input models for further analysis and algorithm development to solve the problems [41, 90]. Effectively, we can analyze our input and characterize its properties, which we can then use to develop more efficient algorithms. Such properties for a road network could be for instance a small crossing number, low maximum vertex degree and low density of its geometry in general. For trajectories, examples are c-packedness of curves [45] or low density of curves [41]. These properties can be leveraged to make more efficient algorithms, such as for computing the Fréchet distance [45]. While the properties discussed by De Berg *et al.* [41] are mainly meant for analyzing geometric problems, it is interesting to see if contextual data such as the local traffic situation or traffic laws exhibit similarly useful properties as well. This may then be combined with geometric realistic properties to further refine and analyze context-aware algorithms for movement analysis.

Bibliography

[1] Mohammad Abam, Shervin Daneshpajouh, Lasse Deleuran, Shayan Ehsani, and Mohammad Ghodsi. Computing homotopic line simplification. *Computational Geometry*, 47(7):728–739, 2014. DOI: `10.1016/j.comgeo.2014.02.002`.

[2] Ravindra K Ahuja. *Network Flows: Theory, Algorithms, and Applications.* Pearson Education, 1993.

[3] Helmut Alt, Alon Efrat, Günter Rote, and Carola Wenk. Matching planar maps. *Journal of Algorithms*, 49(2):262–283, 2003. DOI: `10.1016/S0196-6774(03)00085-3`.

[4] Helmut Alt and Michael Godau. Computing the fréchet distance between two polygonal curves. *International Journal of Computational Geometry & Applications*, 5(01n02):75–91, 1995. DOI: `10.1142/S0218195995000064`.

[5] Gennady Andrienko, Natalia Andrienko, Wei Chen, Ross Maciejewski, and Ye Zhao. Visual Analytics of Mobility and Transportation: State of the Art and Further Research Directions. *IEEE Transactions on Intelligent Transportation Systems*, 18(8):2232–2249, 2017. DOI: `10.1109/TITS.2017.2683539`.

[6] Gennady Andrienko, Natalia Andrienko, and Marco Heurich. An event-based conceptual model for context-aware movement analysis. *International Journal of Geographical Information Science*, 25(9):1347–1370, 2011. DOI: `10.1080/13658816.2011.556120`.

[7] Natalia Andrienko and Gennady Andrienko. Spatial generalization and aggregation of massive movement data. *IEEE Transactions on Visualization and Computer Graphics*, 17(2):205–219, 2010. DOI: `10.1109/TVCG.2010.44`.

[8] Stefano Ardizzoni, Luca Consolini, Mattia Laurini, and Marco Locatelli. Efficient solution algorithms for the bounded acceleration shortest path problem. In *Proceedings of the 60th IEEE Conference on Decision and Control (CDC 2021)*, pages 5729–5734, 2021. DOI: `10.1109/CDC45484.2021.9683191`.

[9] Stefano Ardizzoni, Luca Consolini, Mattia Laurini, and Marco Locatelli. Solution algorithms for the bounded acceleration shortest path problem. *IEEE Transactions on Automatic Control*, 2022. DOI: `10.1109/TAC.2022.3172169`.

[10] Michael Arnold and Enno Ohlebusch. Linear time algorithms for generalizations of the longest common substring problem. *Algorithmica*, 60(4):806–818, 2011. DOI: `10.1007/s00453-009-9369-1`.

[11] Jon Bentley and James Saxe. Decomposable searching problems I. static-to-dynamic transformation. *Journal of Algorithms*, 1(4):301–358, 1980. DOI: `10.1016/0196-6774(80)90015-2`.

[12] Lasse Bergroth, Harri Hakonen, and Timo Raita. A survey of longest common subsequence algorithms. In *Proceedings of the 7th International Symposium on String Processing and Information Retrieval*, pages 39–48, 2000. DOI: `10.1109/SPIRE.2000.878178`.

[13] Donald J Berndt and James Clifford. Using dynamic time warping to find patterns in time series. In *KDD workshop*, volume 10(16), pages 359–370, 1994.

[14] Norbert Blum and Kurt Mehlhorn. On the average number of rebalancing operations in weight-balanced trees. *Theoretical Computer Science*, 11:303–320, 1978. URL: `https://publikationen.sulb.uni-saarland.de/bitstream/20.500.11880/26129/1/fb14_1978_06.pdf`.

[15] Quirijn W Bouts, Irina Irina Kostitsyna, Marc van Kreveld, Wouter Meulemans, Willem Sonke, and Kevin Verbeek. Mapping polygons to the grid with small Hausdorff and Fréchet distance. In *Proceedings of the 24th Annual European Symposium on Algorithms (ESA 2016)*, volume 57 of *LIPIcs*, pages 22–1, 2016. DOI: `10.4230/LIPIcs.ESA.2016.22`.

[16] Michael S. Braasch. In Peter Teunissen and Oliver Montenbruck, editors, *Springer handbook of global navigation satellite systems*. Volume 10, chapter 15, pages 443–468. Springer, 2017. DOI: `10.1007/978-3-319-42928-1`.

[17] Rasmus Bro and Sijmen De Jong. A fast non-negativity-constrained least squares algorithm. *Journal of Chemometrics: A Journal of the Chemometrics Society*, 11(5):393–401, 1997. DOI: `10.1002/(SICI)1099-128X(199709/10)11:5<393::AID-CEM483>3.0.CO;2-L`.

[18] Kevin Buchin, Maike Buchin, Marc van Kreveld, Bettina Speckmann, and Frank Staals. Trajectory grouping structure. In *Proceedings of the Workshop on Algorithms and Data Structures (WADS 2013)*, pages 219–230. Springer, 2013. DOI: `10.1007/978-3-642-40104-6_19`.

[19] Kevin Buchin, Bram Custers, Ivor van der Hoog, Maarten Löffler, Aleksandr Popov, Marcel Roeloffzen, and Frank Staals. Segment visibility counting queries in polygons. *arXiv preprint arXiv:2201.03490*, 2022.

[20] Kevin Buchin, Anne Driemel, Joachim Gudmundsson, Michael Horton, Irina Kostitsyna, Maarten Löffler, and Martijn Struijs. Approximating (k, l)-center clustering for curves. In *Proceedings of the 30th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2019)*, pages 2922–2938. SIAM, 2019. DOI: `10.1137/1.9781611975482.181`.

[21] Kevin Buchin, Irina Kostitsyna, Bram Custers, and Martijn Struijs. A sampling-based strategy for distributing taxis in a road network for occupancy maximization (GIS cup). In *Proceedings of the 27th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, pages 616–619, 2019. DOI: `10.1145/3347146.3363348`.

[22] Kevin Buchin, Wouter Meulemans, André Van Renssen, and Bettina Speckmann. Area-preserving simplification and schematization of polygonal subdivisions. *ACM Transactions on Spatial Algorithms and Systems*, 2(1), April 2016. ISSN: 2374-0353. DOI: `10.1145/2818373`.

[23] Maike Buchin, Somayeh Dodge, and Bettina Speckmann. Similarity of trajectories taking into account geographic context. *Journal of Spatial Information Science*, (9):101–124, 2014. DOI: `10.5311/JOSIS.2014.9.179`.

[24] Sergio Cabello, Mark de Berg, and Marc van Kreveld. Schematization of networks. *Computational Geometry*, 30(3):223–238, 2005. DOI: `10.1016/j.comgeo.2004.11.002`.

[25] Peng Cao, Tomio Miwa, Toshiyuki Yamamoto, and Takayuki Morikawa. Bilevel generalized least squares estimation of dynamic origin–destination matrix for urban network with probe vehicle data. *Transportation research record*, 2333(1):66–73, 2013. DOI: `10.3141/2333-08`.

[26] Ennio Cascetta. Estimation of trip matrices from traffic counts and survey data: a generalized least squares estimator. *Transportation Research Part B: Methodological*, 18(4-5):289–299, 1984. DOI: `10.1016/0191-2615(84)90012-2`.

[27] Enrique Castillo, Antonio J Conejo, José María Menéndez, and Pilar Jiménez. The observability problem in traffic network models. *Computer-Aided Civil and Infrastructure Engineering*, 23(3):208–222, 2008. DOI: `10.1111/j.1467-8667.2008.00531.x`.

[28] Erin Chambers, Eric de Verdiere, Jeff Erickson, Sylvain Lazard, Francis Lazarus, and Shripad Thite. Homotopic Fréchet distance between curves or, walking your dog in the woods in polynomial time. *Computational Geometry*, 43(3):295–311, 2010. DOI: `10.1016/j.comgeo.2009.02.008`.

[29] Erin Chambers, Irina Kostitsyna, Maarten Löffler, and Frank Staals. Homotopy measures for representative trajectories. In *Proceedings of the 24th Annual European Symposium on Algorithms (ESA 2016)*, LIPIcs, 2016. DOI: `10.4230/LIPIcs.ESA.2016.27`.

[30] Erin Wolf Chambers and Yusu Wang. Measuring similarity between curves on 2-manifolds via homotopy area. *Journal of Computational Geometry*, 10(1):96–126, May 2019. DOI: `10.20382/jocg.v10i1a4`.

[31] Pingfu Chao, Yehong Xu, Wen Hua, and Xiaofang Zhou. A survey on map-matching algorithms. In *Proceedings of the Australasian Database Conference*, LNISA 12008, pages 121–133, 2020. DOI: `10.1007/978-3-030-39469-1_10`.

[32] Mingliang Che, Yingli Wang, Chi Zhang, and Xinliang Cao. An enhanced hidden Markov map matching model for floating car data. *Sensors*, 18(6), 2018. DOI: `10.3390/s18061758`.

[33] Jun Chen, Yungang Hu, Zhilin Li, Renliang Zhao, and Liqiu Meng. Selective omission of road features based on mesh density for automatic map generalization. *International Journal of Geographical Information Science*, 23(8):1013–1032, 2009. DOI: `10.1080/13658810802070730`.

[34] Wei Chen, Fangzhou Guo, and Fei-Yue Wang. A survey of traffic data visualization. *IEEE Transactions on Intelligent Transportation Systems*, 16(6):2970–2984, 2015. DOI: `10.1109/TITS.2015.2436897`.

[35] Bram Custers, Jeff Erickson, Irina Kostitsyna, Wouter Meulemans, Bettina Speckmann, and Kevin Verbeek. Orthogonal schematization with minimum homotopy area. In *Proceedings of the 36th European Workshop on Computational Geometry (EuroCG 2020)*, pages 451–457, 2020. URL: `https://www1.pub.informatik.uni-wuerzburg.de/eurocg2020/data/uploads/papers/eurocg20_paper_64.pdf`.

[36] Bram Custers, Mees Van De Kerkhof, Wouter Meulemans, Bettina Speckmann, and Frank Staals. Maximum physically consistent trajectories. *ACM Transactions on Spatial Algorithms and Systems*, 7(4), 2021. DOI: `10.1145/3452378`.

[37] Bram Custers, Mees van de Kerkhof, Wouter Meulemans, Bettina Speckmann, and Frank Staals. Maximum physically consistent trajectories. *ACM Transactions on Spatial Algorithms and Systems*, 7(4):1–33, 2021. DOI: `10.1145/3452378`.

[38] Bram Custers, Wouter Meulemans, Marcel Roeloffzen, Bettina Speckmann, and Kevin Verbeek. Physically consistent map-matching. In *Proceedings of the 30th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, 2022. To appear.

[39] Bram Custers, Wouter Meulemans, Bettina Speckmann, and Kevin Verbeek. Coordinated schematization for visualizing mobility patterns on networks. In *Proceedings of the 11th International Conference on Geographic Information Science (GIScience 2021)*, LIPIcs, 2021. DOI: `10.4230/LIPIcs.GIScience.2021.II.7`.

[40] Bram Custers, Wouter Meulemans, Bettina Speckmann, and Kevin Verbeek. Route reconstruction from traffic flow via representative trajectories. In *Proceedings of the 29th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, pages 41–52, 2021. DOI: `10.1145/3474717.3483650`.

[41] Mark de Berg, Frank van der Stappen, Jules Vleugels, and Matthew Katz. Realistic input models for geometric algorithms. *Algorithmica*, 34(1):81–97, 2002. DOI: `10.1007/s00453-002-0961-x`.

[42] Daniel Delling, Andreas Gemsa, Martin Nöllenburg, Thomas Pajor, and Ignaz Rutter. On d-regular schematization of embedded paths. *Computational Geometry*, 47(3):381–406, 2014. DOI: `10.1016/j.comgeo.2013.10.002`.

[43] Thomas van Dijk, Arthur van Goethem, Jan-Henrik Haunert, Wouter Meulemans, and Bettina Speckmann. Map schematization with circular arcs. In *Proceedings of the International Conference on Geographic Information Science (GIScience 2014)*, volume 8728 of *LNISA*, pages 1–17. Springer, 2014. DOI: `10.1007/978-3-319-11593-1_1`.

[44] Somayeh Dodge, Robert Weibel, and Anna-Katharina Lautenschütz. Towards a taxonomy of movement patterns. *Information visualization*, 7(3-4):240–252, 2008. DOI: `10.1057/PALGRAVE.IVS.9500182`.

[45] Anne Driemel, Sariel Har-Peled, and Carola Wenk. Approximating the fréchet distance for realistic curves in near linear time. *Discrete & Computational Geometry*, 48(1):94–127, 2012. DOI: `10.1007/s00454-012-9402-z`.

[46] Matt Duckham, Marc van Kreveld, Ross Purves, Bettina Speckmann, Yaguang Tao, Kevin Verbeek, and Jo Wood. Modeling checkpoint-based movement with the earth mover's distance. In Jennifer A. Miller, David O'Sullivan, and Nancy Wiegand, editors, *Geographic Information Science*, pages 225–239, Cham. Springer International Publishing, 2016. ISBN: 978-3-319-45738-3. DOI: `10.1007/978-3-319-45738-3_15`.

[47] Laura Eboli, Gabriella Mazzulla, and Giuseppe Pungillo. Combining speed and acceleration to define car users' safe or unsafe driving behaviour. *Transportation Research Part C: Emerging Technologies*, 68:113–125, 2016. DOI: `10.1016/j.trc.2016.04.002`.

[48] Herbert Edelsbrunner, Leonidas Guibas, and Jorge Stolfi. Optimal point location in a monotone subdivision. *SIAM Journal on Computing*, 15(2):317–340, 1986. DOI: `10.1137/0215023`.

[49] Regina Estkowski and Joseph Mitchell. Simplifying a polygonal subdivision while keeping it simple. In *Proceedings of the 17th annual Symposium on Computational Geometry (SoCG 2001)*, pages 40–49, 2001. DOI: `10.1145/378583.378612`.

[50] Martin Fink, Herman Haverkort, Martin Nöllenburg, Maxwell Roberts, Julian Schuhmann, and Alexander Wolff. Drawing metro maps using bézier curves. In *Proceedings of the International Symposium on Graph Drawing (GD 2012)*, pages 463–474. Springer, 2012. DOI: `10.1007/978-3-642-36763-2_41`.

[51] Marco Fiore, Panagiota Katsikouli, Elli Zavou, Mathieu Cunche, Françoise Fessant, Dominique Le Hello, Ulrich Aivodji, Baptiste Olivier, Tony Quertier, and Razvan Stanica. Privacy in trajectory micro-data publishing: a survey. *Transactions on Data Privacy*, 13:91–149, 2020. URL: `http://www.tdp.cat/issues16/tdp.a363a19.pdf`.

[52] Steven Fortune. A sweepline algorithm for Voronoi diagrams. *Algorithmica*, 2(1-4):153, 1987. DOI: `10.1007/BF01840357`.

[53] Michael Garey, David Johnson, Gary Miller, and Christos Papadimitriou. The complexity of coloring circular arcs and chords. *SIAM Journal on Algebraic Discrete Methods*, 1(2):216–227, 1980. DOI: `10.1137/0601025`.

[54] Yong Ge, Hui Xiong, Zhi-hua Zhou, Hasan Ozdemir, Jannite Yu, and Kuo Chu Lee. Top-eye: top-k evolving trajectory outlier detection. In *Proceedings of the 19th ACM International Conference on Information and Knowledge Management*, pages 1733–1736, 2010. DOI: `10.1145/1871437.1871716`.

[55] Arthur van Goethem, Wouter Meulemans, Andreas Reimer, Herman Haverkort, and Bettina Speckmann. Topologically safe curved schematisation. *The Cartographic Journal*, 50(3):276–285, 2013. DOI: `10.1179/1743277413Y.0000000066`.

[56] Chong Yang Goh, Justin Dauwels, Nikola Mitrovic, Muhammad Tayyab Asif, Ali Oran, and Patrick Jaillet. Online map-matching based on hidden Markov model for real-time traffic sensing applications. In *Proc. 15th Int. IEEE Conf. on Intelligent Transportation Systems*, pages 776–781, 2012. DOI: `10.1109/ITSC.2012.6338627`.

[57] Manish Gupta, Jing Gao, Charu C Aggarwal, and Jiawei Han. Outlier detection for temporal data: a survey. *IEEE Transactions on Knowledge and Data Engineering*, 26(9):2250–2267, 2014. DOI: `10.1109/TKDE.2013.184`.

[58] Ulrike Hahn, James Close, and Markus Graf. Transformation direction influences shape-similarity judgments. *Psychological science*, 20(4):447–454, 2009. DOI: `10.1111/j.1467-9280.2009.02310.x`.

[59] Binh Han, Ling Liu, and Edward Omiecinski. Neat: road network aware trajectory clustering. In *Proceedings of the 32nd IEEE International Conference on Distributed Computing Systems (ICDCS 2012)*, pages 142–151. IEEE, 2012. DOI: `10.1109/ICDCS.2012.31`.

[60] Jean-Francois Hangouet. Computation of the Hausdorff distance between plane vector polylines. In *AUTOCARTO-CONFERENCE*, pages 1–10, 1995.

[61] Tzvika Hartman, Avinatan Hassidim, Haim Kaplan, Danny Raz, and Michal Segalov. How to split a flow? In *2012 Proceedings IEEE INFOCOM*, pages 828–836. IEEE, 2012. DOI: `10.1109/INFCOM.2012.6195830`.

[62] Jan-Henrik Haunert and Benedikt Budig. An algorithm for map matching given incomplete road data. In *Proceedings of the 20th ACM SIGSPATIAL International Conference on Advances in Geoegraphic Information Systems*, pages 510–513, 2012. DOI: `10.1145/2424321.2424402`.

[63] Jan-Henrik Haunert, Alexander Wolff, et al. Optimal simplification of building ground plans. In *Proceedings of 21st International Society for Photogrammetry and Remote Sensing Congress (ISPRS 2008)*, pages 372–378, 2008.

[64] Victoria Hodge and Jim Austin. A survey of outlier detection methodologies. *Artificial Intelligence Review*, 22(2):85–126, 2004. DOI: `10.1023/B:AIRE.0000045502.10941.a9`.

[65] Yu-Ling Hsueh and Ho-Chian Chen. Map matching for low-sampling-rate GPS trajectories by exploring real-time moving directions. *Information Sciences*, 433:55–69, 2018. DOI: `10.1016/j.ins.2017.12.031`.

[66] Zhenfeng Huang, Shaojie Qiao, Nan Han, Chang-an Yuan, Xuejiang Song, and Yueqiang Xiao. Survey on vehicle map matching techniques. *CAAI Transactions on Intelligence Technology*, 6(1):55–71, 2021. DOI: `10.1049/cit2.12030`.

[67] Hiroshi Imai and Masao Iri. Computational-geometric methods for polygonal approximations of a curve. *Computer Vision, Graphics, and Image Processing*, 36(1):31–41, 1986. DOI: `10.1016/S0734-189X(86)80027-5`.

[68] Hiroshi Imai and Masao Iri. Polygonal approximations of a curve - formulations and algorithms. *Computational Morphology*:71–86, 1988. DOI: `10.1016/B978-0-444-70467-2.50011-4`.

[69] Muhammad Usman Iqbal and Samsung Lim. Privacy implications of automated GPS tracking and profiling. *IEEE Technology and Society Magazine*, 29(2):39–46, 2010. DOI: `10.1109/MTS.2010.937031`.

[70] George R Jagadeesh and Thambipillai Srikanthan. Online map-matching of noisy and sparse location data with hidden markov and route choice models. *IEEE Transactions on Intelligent Transportation Systems*, 18(9):2423–2434, 2017. DOI: `10.1109/TITS.2017.2647967`.

[71] Dongmin Kim, Suvrit Sra, and Inderjit S Dhillon. Tackling box-constrained optimization via a new projected quasi-newton approach. *SIAM Journal on Scientific Computing*, 32(6):3548–3563, 2010. DOI: `10.1137/08073812X`.

[72] Kyle Kloster, Philipp Kuinke, Michael P O'Brien, Felix Reidl, Fernando Sánchez Villaamil, Blair D Sullivan, and Andrew van der Poel. A practical FPT algorithm for flow decomposition and transcript assembly. In *Proceedings of the 20th Workshop on Algorithm Engineering and Experiments (ALENEX 2018)*, pages 75–86. SIAM, 2018. DOI: `10.1137/1.9781611975055.7`.

[73] Menno-Jan Kraak. The space-time cube revisited from a geovisualization perspective. In *Proceedings of the 21st International Cartographic Conference*, pages 1988–1996, 2003. URL: `https://www.icaci.org/files/documents/ICC_proceedings/ICC2003/Papers/255.pdf`.

[74] John Krumm, Julie Letchner, and Eric Horvitz. Map matching with travel time constraints. In *SAE World Congress*, pages 16–19, 2007. DOI: `10.4271/2007-01-1102`.

[75] Bart Kuijpers, Harvey J Miller, and Walied Othman. Kinetic prisms: incorporating acceleration limits into space–time prisms. *International Journal of Geographical Information Science*, 31(11):2164–2194, 2017. DOI: `10.1080/13658816.2017.1356462`.

[76] Lars Kulik, Matt Duckham, and Max Egenhofer. Ontology-driven map generalization. *Journal of Visual Languages & Computing*, 16(3):245–267, 2005. DOI: `10.1016/j.jvlc.2005.02.001`.

[77] Ove Lampe and Helwig Hauser. Interactive visualization of streaming data with kernel density estimation. In *Proceedings of the 2011 IEEE Pacific Visualization Symposium (PACIFICVIS 2011)*, pages 171–178, 2011. DOI: `10.1109/PACIFICVIS.2011.5742387`.

[78] Jae-Gil Lee, Jiawei Han, and Xiaolei Li. Trajectory outlier detection: a partition-and-detect framework. In *Proceedings of the 24th International Conference on Data Engineering*, pages 140–149, 2008. DOI: `10.1109/ICDE.2008.4497422`.

[79] Jae-Gil Lee, Jiawei Han, and Kyu-Young Whang. Trajectory clustering: a partition-and-group framework. In *Proceedings of the 2007 ACM SIGMOD Conference on Management of Data*, pages 593–604, 2007. DOI: `10.1145/1247480.1247546`.

[80] Wang-Chien Lee and John Krumm. Trajectory preprocessing. In Y. Zheng and X. Zhou, editors, *Computing with spatial trajectories*, pages 3–33. Springer, 2011. DOI: `10.1007/978-1-4614-1629-6_1`.

[81] Xiaolei Li, Jiawei Han, Sangkyum Kim, and Hector Gonzalez. Roam: rule- and motif-based anomaly detection in massive moving object data sets. In *Proceedings of the 2007 SIAM International Conference on Data Mining*, pages 273–284, 2007. DOI: `10.1137/1.9781611972771.25`.

[82] Maarten Löffler and Wouter Meulemans. Discretized approaches to schematization. In *Proceedings of the 29th Canadian Conference on Computational Geometry (CCCG 2017)*, pages 220–225, 2017.

[83] Jed A Long. Kinematic interpolation of movement data. *International Journal of Geographical Information Science*, 30(5):854–868, 2016. DOI: `10.1080/13658816.2015.1081909`.

[84] Yin Lou, Chengyang Zhang, Yu Zheng, Xing Xie, Wei Wang, and Yan Huang. Map-matching for low-sampling-rate GPS trajectories. In *Proceedings of the 17th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, pages 352–361, 2009. DOI: `10.1145/1653771.1653820`.

[85] Rupak Majumdar and Vinayak Prabhu. Computing the skorokhod distance between polygonal traces. In *Proceedings of the 18th International Conference on Hybrid Systems: Computation and Control*, pages 199–208, 2015. DOI: `10.1145/2728606.2728618`.

[86] Shigeru Masuyama, Toshihide Ibaraki, and Toshiharu Hasegawa. The computational complexity of the m-center problems on the plane. *IEICE TRANSACTIONS (1976-1990)*, 64(2):57–64, 1981.

[87] Wouter Meulemans. *Similarity Measures and Algorithms for Cartographic Schematization*. PhD thesis, TU Eindhoven, 2014.

[88] Wouter Meulemans, André van Renssen, and Bettina Speckmann. Area-preserving subdivision schematization. In *Proceedings of the International Conference on Geographic Information Science (GIScience 2010)*, volume 6292 of *LNISA*, pages 160–174. Springer, 2010. DOI: `10.1007/978-3-642-15300-6_12`.

[89] Harvey J Miller. Time geography and space-time prism. *International encyclopedia of geography: People, the earth, environment and technology*, 1, 2017. DOI: `10.1002/9781118786352.wbieg0431`.

[90] Esther Moet, Marc van Kreveld, and Frank van der Stappen. On realistic terrains. *Computational Geometry*, 41(1-2):48–67, 2008. DOI: `10.1016/j.comgeo.2007.10.008`.

[91] Mohamed Mokbel, Louai Alarabi, Jie Bao, Ahmed Eldawy, Amr Magdy, Mohamed Sarwat, Ethan Waytas, and Steven Yackel. Mntg: an extensible web-based traffic generator. In *Proceedings of the 9th International Symposium on Spatial and Temporal Databases (SSTD 2013)*, pages 38–55, Munich, Germany. Springer, Springer, 2013. DOI: `10.1007/978-3-642-40235-7_3`.

[92] Marcello Montanino and Vincenzo Punzo. Trajectory data reconstruction and simulation-based validation against macroscopic traffic patterns. *Transportation Research Part B: Methodological*, 80:82–106, 2015. DOI: `10.1016/j.trb.2015.06.010`.

[93] Abhinandan Nath and Erin Taylor. K-Median clustering under discrete Fréchet and Hausdorff distances. *arXiv*, 2020. URL: `https://arxiv.org/pdf/2004.00722.pdf`.

[94] Paul Newson and John Krumm. Hidden markov map matching through noise and sparseness. In *Proceedings of the 17th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, pages 336–343, 2009. DOI: `10.1145/1653771.1653818`.

[95] Jürg Nievergelt and Edward M Reingold. Binary search trees of bounded balance. *SIAM Journal on Computing*, 2(1):33–43, 1973. DOI: `10.1145/800152.804906`.

[96] Martin Nollenburg and Alexander Wolff. Drawing and labeling high-quality metro maps by mixed-integer programming. *IEEE Transactions on Visualization and Computer Graphics*, 17(5):626–641, 2010. DOI: `10.1109/TVCG.2010.81`.

[97] OpenStreetMap contributors. Planet dump retrieved from osm.org. `https://www.openstreetmap.org`, 2020.

[98] Takayuki Osogami and Rudy Raymond. Map matching with inverse reinforcement learning. In *Proceedings of the 23rd International Joint Conference on Artificial Intelligence (IJCAI 2013)*, pages 2547–2553, 2013.

[99] Mark H Overmars and Jan van Leeuwen. Worst-case optimal insertion and deletion methods for decomposable searching problems. *Information Processing Letters*, 12(4):168–173, 1981. DOI: `10.1016/0020-0190(81)90093-4`.

[100] Vincenzo Punzo, Maria Teresa Borzacchiello, and Biagio Ciuffo. On the assessment of vehicle trajectory data accuracy and application to the Next Generation SIMulation (NGSIM) program data. *Transportation Research Part C: Emerging Technologies*, 19(6):1243–1262, 2011. DOI: `10.1016/j.trc.2010.12.007`.

[101] Mohammed A Quddus, Washington Y Ochieng, and Robert B Noland. Current map-matching algorithms for transport applications: state-of-the art and future research directions. *Transportation Research Part C: Emerging Technologies*, 15(5):312–328, 2007. DOI: `10.1016/j.trc.2007.05.002`.

[102] Ahmed El-Rabbany. *Introduction to GPS: the global positioning system*. Artech house, 2002.

[103] Kotagiri Ramamohanarao, Hairuo Xie, Lars Kulik, Shanika Karunasekera, Egemen Tanin, Rui Zhang, and Eman Bin Khunayn. Smarts: scalable microscopic adaptive road traffic simulator. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 8(2):1–22, 2016. DOI: `10.1145/2898363`.

[104] Roeland Scheepens, Niels Willems, Huub van de Wetering, and Jarke Van Wijk. Interactive visualization of multivariate trajectory data with density maps. In *Proceedings of the 2011 IEEE Pacific Visualization Symposium (PACIFICVIS 2011)*, pages 147–154, 2011. DOI: `10.1109/PACIFICVIS.2011.5742384`.

[105] Mohammad Sharif and Ali Alesheikh. Context-aware movement analytics: implications, taxonomy, and design framework. *Wiley Interdisciplinary Reviews: Data mining and knowledge discovery*, 8(1):e1233, 2018. DOI: `10.1002/widm.1233`.

[106] Martin Skutella. *An introduction to network flows over time.* In *Research Trends in Combinatorial Optimization: Bonn 2008.* Springer Berlin Heidelberg, Berlin, Heidelberg, 2009, pages 451–482. ISBN: 978-3-540-76796-1. DOI: `10.1007/978-3-540-76796-1_21`.

[107] Roniel de Sousa, Azzedine Boukerche, and Antonio Loureiro. Vehicle trajectory similarity: models, methods, and applications. *ACM Computing Surveys (CSUR)*, 53(5):1–32, 2020. DOI: `10.1145/3406096`.

[108] Radan Šuba, Martijn Meijers, and Peter Van Oosterom. Continuous road network generalization throughout all scales. *ISPRS International Journal of Geo-Information*, 5(8):145, 2016. DOI: `10.3390/ijgi5080145`.

[109] Robert Thomson. The 'stroke' concept in geographic network generalization and analysis. In *Progress in Spatial Data Handling*, pages 681–697. Springer, 2006. DOI: `10.1007/3-540-35589-8_43`.

[110] Christian Tominski, Heidrun Schumann, Gennady Andrienko, and Natalia 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`.

[111] Mees van de Kerkhof, Irina Kostitsyna, Maarten Löffler, Majid Mirzanezhad, and Carola Wenk. Global curve simplification. In *Proceedings of the 27th Annual European Symposium on Algorithms (ESA 2019)*, LIPIcs, page 67, 2019. DOI: `10.4230/lipics.esa.2019.67`.

[112] Mees van de Kerkhof, Irina Kostitsyna, Marc van Kreveld, Maarten Löffler, and Tim Ophelders. Route-preserving road network generalization. In *Proceedings of the 28th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, pages 381–384, 2020. DOI: `10.1145/3397536.3422234`.

[113] Stef van den Elzen and Jarke van Wijk. Multivariate network exploration and presentation: from detail to overview via selections and aggregations. *IEEE Transactions on Visualization and Computer Graphics*, 20(12):2310–2319, 2014. DOI: `10.1109/TVCG.2014.2346441`.

[114] Thomas van Dijk, Arthur van Goethem, Jan-Henrik Haunert, Wouter Meulemans, and Bettina Speckmann. Map schematization with circular arcs. In *Proceedings of the International Conference on Geographic Information Science (GIScience 2014)*, pages 1–17, 2014. DOI: `10.1007/978-3-319-11593-1_1`.

[115] Arthur van Goethem, Wouter Meulemans, Andreas Reimer, Herman Haverkort, and Bettina Speckmann. Topologically safe curved schematization. *The Cartographic Journal*, 50(3):276–285, 2013. DOI: `10.1179/1743277413Y.0000000066`.

[116] Marc van Kreveld. Smooth generalization for continuous zooming. In *Proceedings of the 20th International Cartography Conference*, pages 2180–2185, 2001. URL: `https://icaci.org/files/documents/ICC_proceedings/ICC2001/icc2001/file/f13042.pdf`.

[117] Peter van Oosterom. Variable-scale topological data structures suitable for progressive data transfer: the gap-face tree and gap-edge forest. *Cartography and Geographic Information Science*, 32(4):331–346, 2005. DOI: `10.1559/152304005775194782`.

[118] Benedicte Vatinlen, Fabrice Chauvet, Philippe Chrétienne, and Philippe Mahey. Simple bounds and greedy algorithms for decomposing a flow into a minimal set of paths. *European Journal of Operational Research*, 185(3):1390–1401, 2008. DOI: `10.1016/j.ejor.2006.05.043`.

[119] László A Végh. A strongly polynomial algorithm for a class of minimum-cost flow problems with separable convex objectives. *SIAM Journal on Computing*, 45(5):1729–1761, 2016. DOI: `10.1137/140978296`.

[120] Michail Vlachos, George Kollios, and Dimitrios Gunopulos. Discovering similar multidimensional trajectories. In *Proceedings of the 18th International Conference on Data Engineering*, pages 673–684, 2002. DOI: `10.1109/ICDE.2002.994784`.

[121] Roy Weiss and Robert Weibel. Road network selection for small-scale maps using an improved centrality-based algorithm. *Journal of Spatial Information Science*, 2014(9):71–99, 2014. URL: `https://josis.org/index.php/josis/article/view/53/53`.

[122] Carola Wenk, Randall Salas, and Dieter Pfoser. Addressing the need for map-matching speed: localizing global curve-matching algorithms. In *Proceedings of the 18th International Conference on Scientific and Statistical Database Management (SSDBM 2006)*, pages 379–388, 2006. DOI: `10.1109/SSDBM.2006.11`.

[123] Niels Willems, Huub van de Wetering, and Jarke van Wijk. Visualization of vessel movements. In *Computer Graphics Forum*, volume 28 of number 3, pages 959–966. Wiley Online Library, 2009. DOI: `10.1111/j.1467-8659.2009.01440.x`.

[124] Alexander Wolff. Drawing subway maps: a survey. *Informatik-Forschung und Entwicklung*, 22(1):23–44, 2007. DOI: `10.1007/s00450-007-0036-y`.

[125] Jung-Im Won, Sang-Wook Kim, Ji-Haeng Baek, and Junghoon Lee. Trajectory clustering in road network environment. In *Proceedings of the IEEE Symposium on Computational Intelligence and Data Mining (CIDM 2009)*, pages 299–305. IEEE, 2009. DOI: `10.1109/CIDM.2009.4938663`.

[126] Jo Wood, Jason Dykes, and Aidan Slingsby. Visualisation of origins, destinations and flows with OD maps. *The Cartographic Journal*, 47(2):117–129, 2010. DOI: `10.1179/000870410X12658023467367`.

[127] Hsiang-Yun Wu, Benjamin Niedermann, Shigeo Takahashi, and Martin Nöllenburg. A survey on computing schematic network maps: the challenge to interactivity. In *Proceedings of the 2nd Schematic Mapping Workshop*, 2019.

[128] Hsiang-Yun Wu, Benjamin Niedermann, Shigeo Takahashi, Maxwell Roberts, and Martin Nöllenburg. A survey on transit map layout–from design, machine, and human perspectives. In *Proceedings of the Computer Graphics Forum*, volume 39 of number 3, pages 619–646. Wiley Online Library, 2020. DOI: `10.1111/cgf.14030`.

[129] Yajie Yan, Kyle Sykes, Erin Chambers, David Letscher, and Tao Ju. Erosion thickness on medial axes of 3d shapes. *ACM Transactions on Graphics (TOG)*, 35(4):1–12, 2016. DOI: `10.1145/2897824.2925938`.

[130] Bisheng Yang, Xuechen Luan, and Qingquan Li. Generating hierarchical strokes from urban street networks based on spatial pattern recognition. *International Journal of Geographical Information Science*, 25(12):2025–2050, 2011. DOI: 10.1080/13658816.2011.570270.

[131] Can Yang and Gyozo Gidofalvi. Fast map matching, an algorithm integrating hidden markov model with precomputation. *International Journal of Geographical Information Science*, 32(3):547–570, 2018. DOI: 10.1080/13658816.2017.1400548.

[132] Yalong Yang, Tim Dwyer, Sarah Goodwin, and Kim Marriott. Many-to-many geographically-embedded flow visualisation: an evaluation. *IEEE transactions on visualization and computer graphics*, 23(1):411–420, 2016. DOI: 10.1109/TVCG.2016.2598885.

[133] Jin Y Yen. Finding the k shortest loopless paths in a network. *Management Science*, 17(11):712–716, 1971. DOI: 10.1287/mnsc.17.11.712.

[134] Byoung-Kee Yi, Hosagrahar Jagadish, and Christos Faloutsos. Efficient retrieval of similar time sequences under time warping. In *Proceedings 14th International Conference on Data Engineering (ICDE 1998)*, pages 201–208. IEEE, 1998. DOI: 10.1109/ICDE.1998.655778.

[135] Wenhao Yu, Yifan Zhang, Tinghua Ai, Qingfeng Guan, Zhanlong Chen, and Haixia Li. Road network generalization considering traffic flow patterns. *International Journal of Geographical Information Science*, 34(1):119–149, 2020. DOI: 10.1080/13658816.2019.1650936.

[136] Guan Yuan, Penghui Sun, Jie Zhao, Daxing Li, and Canwei Wang. A review of moving object trajectory clustering algorithms. *Artificial Intelligence Review*, 47(1):123–144, 2017. DOI: 10.1007/s10462-016-9477-7.

[137] Guan Yuan, Penghui Sun, Jie Zhao, Daxing Li, and Canwei Wang. A review of moving object trajectory clustering algorithms. *Artificial Intelligence Review*, 47(1):123–144, 2017. DOI: 10.1007/s10462-016-9477-7.

[138] Guan Yuan, Shixiong Xia, Lei Zhang, Yong Zhou, and Cheng Ji. Trajectory outlier detection algorithm based on structural features. *Journal of Computational Information Systems*, 7(11):4137–4144, 2011. URL: https://www.researchgate.net/profile/Zhou-Yong-15/publication/266603308_Trajectory_Outlier_Detection_Algorithm_Based_on_Structural_Features/links/5631921e08ae13bc6c35814c/Trajectory-Outlier-Detection-Algorithm-Based-on-Structural-Features.pdf.

[139] Yu Zheng. Trajectory data mining: an overview. *ACM Transactions on Intelligent Systems and Technology*, 6(3):29, 2015. DOI: `10.1145/2743025`.

[140] Yu Zheng, Lizhu Zhang, Xing Xie, and Wei-Ying Ma. Mining interesting locations and travel sequences from GPS trajectories. In *Proceedings of the 18th conference on World wide web (WWW 2009)*, pages 791–800, 2009. DOI: `10.1145/1526709.1526816`.

Chapter 8

Summary

Spatio-temporal trajectories are a ubiquitous source for analyzing patterns of mobility of people and animals. They arise in fields like traffic management in cities and the analysis of bird migrations. A vast body of research exists that provides the tools and algorithms to analyse such trajectories. The major challenges when working with these trajectories are to get clean, complete data and to visualize and summarize the vast amount of data that is available. To address these challenges in a meaningful way, we seek to provide algorithms that take into account the context of the trajectories: the environment and setting in which the trajectories were measured. In this thesis, we provide novel algorithms and approaches that take into account the spatial environment, namely the road-network, for traces of vehicles and considers the real-world physical context of trajectories.

In the context of detecting outliers in spatio-temporal trajectories, we introduce a model that considers the consistency of the spatio-temporal data with physical reality. We consider models where we bound the maximum speed of the entity that underlies the measured trajectory and where we bound both the maximum speed and acceleration. Using this approach, we can detect outlying measurements in the trajectory, by considering the measurements that are not part of the largest consistent subtrajectory to be outliers. This gives us a means to cleanup trajectories to be used later in processing. For the speed-bounded model, we introduce efficient algorithms to compute them in both the 1D and 2D setting. For the acceleration-bounded model, we derive an algorithm for 1D. The resulting algorithms are analyzed by employing a real-world data set, where we compare the resulting trajectories with results from benchmark and state-of-the-art algorithms. Furthermore, we use our physics models to tackle the problem of gap-filling a trajectory.

Next, we consider the problem of reconstructing routes in a road network. By combining different sources of data that describe the traffic situation in the network, we can infer more information about the underlying traffic that is represented by these data sources. We are particularly interested in reconstructing routes from different data sources. We employ a set of GPS trajectories that we deem representative of the traffic and a set of checkpoint data: measurements of the traffic volume at fixed locations in the road network. The set of trajectories is a sample of the total traffic in the network whereas the checkpoint data measures the total traffic volume, but does not provide information on routes. Thus, combining them can provide a more complete picture of the routes of traffic in the network. We reconstruct routes from these data sources, such that the routes align with the representative trajectories, while capturing as much of the checkpoint traffic volume as possible. We measure the alignment to the representative trajectories by employing a geometric distance measure between reconstructed routes and representative trajectories. We show that variants of this problem are NP-hard. Hence, we introduce a heuristic approach for computing the reconstructed routes. We apply this to a real-world and synthetic dataset and compare to baseline algorithms how well the routes align to the checkpoint data and the representative trajectories.

With the cleaned up and reconstructed routes, we now consider the problem of visualizing mobility patterns on road networks. We visualize a (large) set of spatio-temporal trajectories in a summarized way, allowing us to easily see major patterns in the data set. To this end, we apply schematization: the act of simplifying data for visualization, where the summarizing power is favored over high-fidelity reproduction of the data. To be able to see the relation between the original location of the trajectories and the summarized patterns, we use the underlying road-network to provide context for the visualization. We present a pipeline that coordinates schematization between the road-network and the trajectory data set. This pipeline consists of simple to grasp steps that modify the road network and set of trajectories simultaneously. Finally, we derive patterns from the trajectories in the low complexity road network that we visualize in a metro map style on the schematized road-network. We explore the result of this schematization on two different real-world data sets, which show that the approach seems promising for distinguishing different patterns of mobility.

Continuing on the schematization, we consider what makes a good distance measure that captures the intuition behind schematization. We restrict ourselves to orthogonal polygons, in line with previous work on schematization of planar subdivisions and road-networks. We investigate schematization under the homotopy area, a distance measure that captures topological changes between schematization input and output. We present an algorithm to optimally compute schematizations under this distance measure that runs in polynomial time. To improve upon this running time, we consider a move-based model that captures salient aspects of the homotopy area. We propose a greedy algorithm for this model based on previous work. We show with careful geometric arguments that the result of this algorithm approximates the optimal solution.

Curriculum Vitae

Bram Custers was born on the 9th of November, 1992, in Utrecht, the Netherlands. He finished secondary education at Christelijk Gymnasium Utrecht in Utrecht, the Netherlands in 2010. He then studied Physics and Astronomy at the Utrecht University in Utrecht, obtaining his Bachelor's degree in 2014 (cum laude). He obtained his Master's degree in Game and Media Technology (cum laude) in 2018. Since 2018, he has been a PhD student under the supervision of Bettina Speckmann and Kevin Verbeek, co-supervised by Wouter Meulemans. The results of his research as a PhD student are the contents of this dissertation.