Chapter 3
Efficient Mining of Volunteered Trajectory Datasets

Axel Forsch, Stefan Funke, Jan-Henrik Haunert, and Sabine Storandt

Abstract With the ubiquity of mobile devices that are capable of tracking positions (be it via GPS or Wi-Fi/mobile network localization), there is a continuous stream of location data being generated every second. These location measurements are typically not considered individually but rather as sequences, each of which reflects the movement of one person or vehicle, which we call trajectory. This chapter presents new algorithmic approaches to process and visualize trajectories both in the network-constrained and the unconstrained case.

Keywords Trajectories · Data mining · Indexing · Driving preferences · Map matching · Anonymization · Isochrones · Processing pipeline

3.1 Introduction

An abundance of volunteered trajectory data was made openly available in the last decades, fueled by the development of cheap sensor technology and widespread access to tracking devices. The OpenStreetMap project alone collected and published some 2.43 million GPS trajectories from around the world in the past 17 years. Such datasets enable a wealth of applications, as, e.g., movement pattern extraction, map generation, social routing, or traffic flow analysis and prediction.

Dealing with huge trajectory datasets poses many challenges, though. This is especially true for volunteered trajectory datasets which are often heterogeneous in terms of geolocation accuracy, duration of movement, or underlying transportation mode. A raw trajectory is typically represented as a sequence of time-stamped geolocations, potentially enriched with semantic information. To enable trajectory-based applications, the raw trajectories need to be processed appropriately. In the

A. Forsch (✉) · S. Funke · J.-H. Haunert · S. Storandt
Universität Bonn, Bonn, Germany
e-mail: forsch@igg.uni-bonn.de; funke@fmi.uni-stuttgart.de; haunert@igg.uni-bonn.de; storandt@informatik.uni-konstanz.de

D. Burghardt et al. (eds.), *Volunteered Geographic Information*,
https://doi.org/10.1007/978-3-031-35374-1_3

following, we describe the core steps of a general processing pipeline. Of course, this pipeline may be refined or adapted depending on the target application.

1. **Anonymization** Trajectories are personal data and can reveal sensitive information about the recording user. As such, the user's privacy needs to be protected.
2. **Preprocessing** Given the raw data, trajectories are usually first filtered and cleaned. For trajectories that stem from movement in an underlying network (e.g., car trajectories), the preprocessing typically includes map matching, which aims at identifying the path in the network that most likely led to the given movement sequence. Map matching helps to reduce noise that stems, e.g., from sensor inaccuracies and enables more efficient storage.
3. **Storing and Indexing** For huge trajectory sets, scalable storage systems and indexing methods are crucial. Storing often involves (lossy or lossless) compression of the trajectories. Furthermore, several query types, such as spatiotemporal range queries or nearest neighbor queries, should be supported by such a system to allow for effective mining later on. Therefore, indexing structures that allow for efficient retrieval of such query result sets need to be built.
4. **Mining** Trajectory mining tasks include, among others, clustering, classification, mode detection, and pattern extraction. Mining tasks often benefit from the availability of a diverse set of trajectory data, covering, for example, large spatial regions or time intervals. Most applications depend on successfully performing one or more mining tasks.
5. **Visualization** The results of the processing steps are often hard to interpret for non-expert users. To improve the accessibility of the results, appropriate visualizations are needed.

This chapter is a review of the achievements made inside the projects "Dynamic and Customizable Exploitation of Trajectory Data" and "Inferring personalized multi-criteria routing models from sparse sets of voluntarily contributed trajectories" inside the DFG priority program "Volunteered Geographic Information: Interpretation, Visualization, and Social Computing" (SPP 1894). We will discuss achievements for each step of the pipeline.

First, we describe a method to protect the privacy of the user who provided their trajectories. In specific, an algorithm is presented that anonymizes sensitive locations, such as the user's home location, by truncating the trajectory. For this, a formal attack model is introduced. To maximize the utility of the anonymized data, the algorithm truncates as little information as possible while still guaranteeing that the user's privacy cannot be breached by the defined attacks. This section is based on Brauer et al. (2022).

Then, we present a new map-matching method that is able to deal with so-called semi-restricted trajectories. Those are trajectories in which the movement is only partially bound to an underlying network, for example, stemming from a pedestrian who walked along some hiking path but crossed a meadow in between. The challenge is to identify the parts of the trajectory that happened within the network and to find a concise representation of the unrestricted parts. A method

based on the careful tessellation of open spaces and multi-criteria path selection that copes with this challenge is presented. This section is based on Behr et al. (2021).

Afterward, we present the PATHFINDER storage and indexing system. It allows dealing with huge quantities of map-matched trajectories with the help of a novel data structure that augments the underlying network with so-called shortcut edges. Trajectories are then represented as sequences of such shortcut edges, which automatically compresses and indexes them. Integrating spatial and temporal search structures allows for answering space-time range queries within a few microseconds per trajectory in the result set. This section is based on Funke et al. (2019).

Then, we review new algorithms for driving preference mining from trajectory sets. Based on a linear preference model, these algorithms identify the driving preferences from the trajectories and use this information to compress and cluster the trajectories. This section is based on Forsch et al. (2022) and Barth et al. (2021).

Finally, we present a method to visualize the results of preference mining to improve their interpretability. For this, an isochrone visualization is used. The isochrones show which areas are easy to access for a user with a specific routing profile and which areas are more difficult to access. This information is especially useful for infrastructure planning. This section is based on Forsch et al. (2021).

The overarching vision is to build an integrated system that enables uploading, preprocessing, indexing, and mining of trajectory sets in a flexible fashion and which scales to the steadily growing OpenStreetMap trajectory dataset. We conclude the chapter by discussing some open problems on the way to achieving this goal.

3.2 Protection of Sensitive Locations

Trajectories are personal data, and, as such, they come within the ambit of the General Data Protection Regulation (GDPR). Therefore, the user's privacy must always be considered when collecting or analyzing users' trajectories. For this, location privacy-preserving mechanisms (LPPMs) are developed. In this section, we review the LPPM presented in Brauer et al. (2022), which focuses on protecting sensitive locations along the trajectory.

Publishing trajectories anonymously, i.e., without giving the name of the user recording the trajectory, is not sufficient to protect the user's privacy. An adversary can still extract personal information from this data, such as the user's home and workplace. This extracted information can link the published trajectories back to the user's identity by using so-called re-identification attacks.

A widely used concept to prevent re-identification attacks is *k-anonymity* (Sweeney 2002). A k-anonymized trajectory cannot be distinguished from at least $k - 1$ other trajectories in the same dataset. LPPMs that k-anonymize trajectory datasets (e.g., Abul et al. 2008; Yarovoy et al. 2009; Monreale et al. 2010; Dong and Pi 2018) make use of generalization, suppression, and distortion. While k-anonymized trajectory datasets still retain the characteristics of the trajectories when analyzing the dataset as a whole, the utility of single trajectories in the

dataset is greatly diminished. To counteract this, anonymization can be restricted to protecting sensitive locations along the trajectories. Sensitive locations are either user-specific, e.g., the user's home or workplace, or they are universally sensitive, such as hospitals, banks, or casinos. LPPMs that focus on the protection of sensitive locations (e.g., Huo et al. 2012; Dai et al. 2018; Wang and Kankanhalli 2020) are more utility preserving and thus allow for better analysis in cases where no strict k-anonymity is needed. In this section, a truncation-based algorithm for protecting sensitive locations is reviewed that transfers the concept of k-anonymity to sensitive locations.

3.2.1 Privacy Concept

In this section, the problem of protecting the users' privacy is formalized. At first, a formal attacker model is introduced that defines the kind of adversary considered. Then, the privacy model to prevent the attacks defined in the attacker model is presented.

Attacker Model The attacker has access to a set of trajectories $\Theta = \{T_1, \ldots, T_l\}$ that have a common destination $\tilde{s}$. Furthermore, the attacker knows a set of sites S that is guaranteed to contain $\tilde{s}$. In this context, sites can be any collection of points of interest, such as addresses or buildings, and S could contain all buildings for a given region. The attacker's objective is to identify $\tilde{s}$ based on Θ and S. For this, the attacker utilizes several attack functions $f_1, \ldots, f_a$. For each trajectory T, an attack function f yields a set of candidate sites: $f(S, T) \subseteq S$. By applying all attack functions to all trajectories in Θ, the attacker collects a joint set of candidates, which enables him to infer $\tilde{s}$. For example, the attacker could assume that $\tilde{s}$ is the site that occurs most often in the joint set.

Privacy Model Brauer et al. (2022) introduced a privacy concept called k-site-unidentifiability which transfers the concept of k-anonymity to sites. Given a set S of sites and a destination site $\tilde{s} \in S$, k-site-unidentifiability requires that $\tilde{s}$ is indistinguishable from at least $k - 1$ other sites in S. Put differently, if k-site-unidentifiability is satisfied, $\tilde{s}$ is hidden in a set $C(\tilde{s})$ of k sites, which is termed *protection set*.

Recall that the attacker's attack functions return sets of candidate sites. The sites in the protection set $C(\tilde{s})$ are indistinguishable with respect to a given attack function f if either all sites or none of the sites in $C(\tilde{s})$ are returned as part of the candidate set for f. In order to preserve k-site-unidentifiability, this property must be guaranteed for all attack functions. The sites in S are available to the attacker and thus cannot be changed. Therefore, this can only be done by altering the trajectories in Θ. In conclusion, the problem is defined as follows:

Given a set of sites S, a set of trajectories Θ with a common destination $\tilde{s} \in S$, and a set of attack functions F, transform Θ into a set of trajectories Θ' that fulfills k-site-unidentifiability for all attack functions in F, i.e., either all or none of the sites in $C(\tilde{s})$ are part of the attack's result for each trajectory in Θ.

In the following, an algorithm that truncates trajectories such that they fulfill k-site-unidentifiability is presented.

3.2.2 The S-TT Algorithm

In this section, the **S**ite-dependent **T**rajectory **T**runcation algorithm (S-TT) is explained. This algorithm truncates a trajectory T such that k-site-unidentifiability with respect to a set F of given attack functions is guaranteed. The truncated trajectory T' is obtained by iteratively suppressing the endpoint of T until each attack function either contains all sites of $C(\tilde{s})$ in its candidate set or none of them. The S-TT algorithm is simple to implement, yet it guarantees that none of the attack functions can be used to single out $\tilde{s}$ from the other sites in $C(\tilde{s})$. For using the algorithm, two further considerations need to be made. Firstly, the protection set $C(\tilde{s})$ needs to be selected. Secondly, assumptions on the used attack functions in F need to be made. In the following, both of these aspects are discussed.

Obtaining the Protection Sets The choice of the protection set $C(\tilde{s})$ greatly influences the quality of the anonymized data. There are two requirements for the protection sets to guarantee good anonymization results. Firstly, the sites in the protection set should be spatially close to each other. This maximizes the utility of the anonymized data, as the truncated part of the trajectory gets minimized. Secondly, the choice of the protection set should not depend on $\tilde{s}$. Otherwise, an attacker can infer $\tilde{s}$ by reverse engineering based on its protection set. Both requirements can be fulfilled by computing a partition of S into pairwise disjoint subsets, where each subset contains at least k sites. Each of these subsets becomes the protection set for all sites included in it. The partition of the sites is a clustering problem with minimum cluster size k and spatial compactness as the optimization criterion. Possible approaches can be purely geometric-based, e.g., by computing a minimum-weight spanning forest of the sites such that each of its trees spans at least k sites (e.g., Imielińska et al. 1993), or they can take additional information into account, such as the road network (e.g., Haunert et al. 2021). A polygonal representation of the protection sets, called *protection cell*, is obtained by unioning the Voronoi cells of the sites in the protection set.

Geometric Attack Functions The S-TT algorithm truncates a trajectory based on the attack functions in F. In the following, a geometric-based S-TT algorithm is presented by defining attack functions for a geometric inference of the trajectories' destination site $\tilde{s}$. Two important characteristics of a trajectory that can be used to identify its destination site are the proximity to and the direction toward the

destination site. In the following, attacks using these characteristics are formalized into attack functions.

A trajectory T most likely ends very close to its destination site. Therefore, it is reasonable to assume that the trajectory's endpoint $\text{end}(T)$ is closer to $\tilde{s}$ than to all other sites in S. Thus, the proximity-based attack function f_p returns the site closest to $\text{end}(T)$ as the candidate site:

$$f_\text{p}(S, T) := \underset{s \in S}{\arg\min}\, \text{d}(\text{end}(T), s), \tag{3.1}$$

where the function d is the Euclidean distance. Note that this attack function can easily be extended to return the n-nearest sites to $\text{end}(T)$ to introduce some tolerance.

A second aspect that can be used to infer the destination is the direction in which the trajectory is headed. Specifically, the trajectory's last segment e is of interest. However, the segment e, most likely, does not point exactly toward $\tilde{s}$. Therefore, a V-shaped region $R(t)$ anchored in $\text{end}(T)$ that is mirror-symmetric with respect to e and has an infinite radius is considered (Fig. 3.1). The opening angle of the V-shape is fixed and denoted with α.

The direction-based attack function f_d returns the sites that are inside $R(T)$ as its candidate set:

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

The attacks f_p and f_d are simple, yet common sense suggests that they may be fairly successful. Figure 3.2 displays the geometric S-TT algorithm on an example trajectory. In the starting situation (Fig. 3.2a), the closest site to $\text{end}(T)$ is $\tilde{s}$. Thus, the algorithm suppresses the endpoint of T until the site closest to its endpoint is not part of the protection set $C(\tilde{s})$ of $\tilde{s}$ anymore (Fig. 3.2b). At this point, some, but not all, of the sites in $C(\tilde{s})$ are part of the candidate set for the direction-based attack (Fig. 3.2b), violating k-site-unidentifiability for this attack function. Thus, truncation continues until either all the sites or none of the sites in $C(\tilde{s})$ are part of

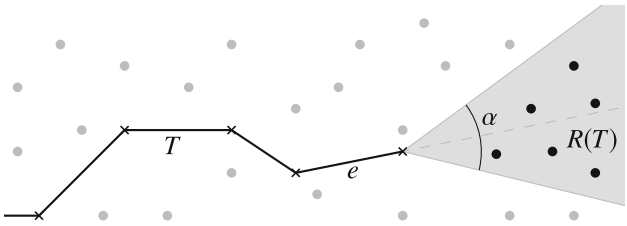

Fig. 3.1 Schematic representation of a direction-based attack. The sites (round dots) inside $R(T)$ (colored black) are candidate sites this attack function returns. $R(T)$ is based on the terminating segment e of trajectory T

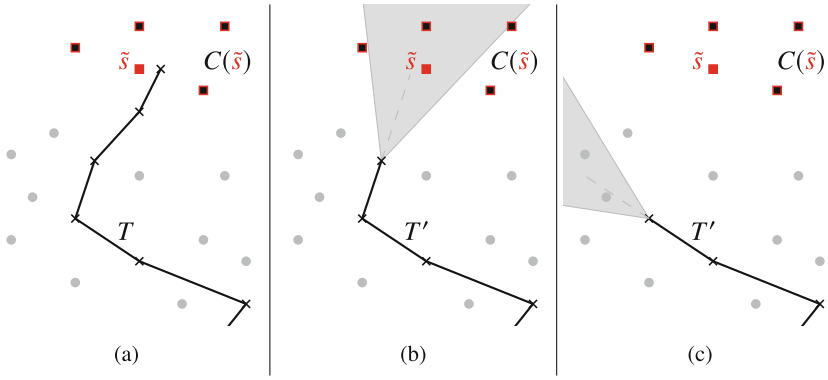

Fig. 3.2 Schematic illustration of the geometric S-TT algorithm. (**a**) The original trajectory T leads to a destination site $\tilde{s}$. This destination has to be hidden among the sites in the protection set $C(\tilde{s})$, depicted as squares. All other sites are shown as gray points. The S-TT algorithm suppresses the trajectory's endpoint until neither (**b**) the site closest to the endpoint is part of the protection set $C(\tilde{s})$ nor (**c**) a V-shaped region that is aligned with the last segment of the truncated trajectory T' contains some, but not all, of the sites in $C(\tilde{s})$

the candidate set of f_{d} anymore (Fig. 3.2c). Note that at this point, neither f_{p} nor f_{d} allow for a distinction between the sites in $C(\tilde{s})$. Thus, k-site-unidentifiability is preserved, and the algorithm is done.

3.2.3 Experimental Results

In this section, experimental results carried out with an implementation of the geometric-based S-TT algorithm are presented. The evaluation focuses on the algorithm's utility and the amount of data lost during anonymization. In this experimental study, it is assumed that the origin and destination of the trajectories were sensitive by default and should be protected under k-site-unidentifiability. For this, the S-TT algorithm is applied to both ends of the trajectories.

Input Data A dataset consisting of 10,927 synthetically generated trajectories in Greater Helsinki, Finland, is used for the evaluation. The trajectories are generated in a three-step process. First, the trajectory endpoints are randomly sampled using a weighted random selection algorithm with the population density as weight. This means that locations in densely populated areas are more likely to be selected as endpoints than locations in sparsely populated areas. In the second step, the endpoints are connected to the road network using a grid-based version of Dijkstra (1959)'s shortest path algorithm. Finally, using generic Dijkstra's algorithm, the shortest path between the two points on the road network is computed and used to connect the two endpoints to a full trajectory.

Selection of the Protection Sets In most scenarios, the sites of origin and destination of the trajectories are not explicitly given. For truncation, the S-TT algorithm does not require the destination site $\tilde{s}$ but only its protection set $C(\tilde{s})$. A naive approach to selecting $C(\tilde{s})$ would be to select the site cluster belonging to the site closest to the trajectory's endpoint. However, trajectory data is prone to positioning uncertainties. Using the naive approach, these uncertainties could lead to selecting a protection set that does not contain $\tilde{s}$, severely diminishing the anonymization quality. A circular buffer of radius b around the endpoint is used to account for these uncertainties. The protection sets of all protection cells intersecting this buffer are unioned into one large protection set. This unioned protection set is then used as the protection set for the trajectory.

Evaluation The geometric S-TT algorithm, as outlined in Sect. 3.2.2, has three major configuration parameters: the minimum size of the protection sets k, the buffer radius b, and the opening angle α for the direction-based attack function. In the following, the effect of these three parameters is analyzed.

Figure 3.3 displays the mean length $\overline{\delta}_{\text{supp}}$ of the truncated trajectory parts for different parameterizations. The parameters k and b influence the size of the protection cell. Raising these parameters has a significant impact on the results, with the suppressed length of the trajectories increasing almost at a linear rate (Fig. 3.3a and c). While these results show that small values for k and b preserve more of the data, the choice of these parameters mainly depends on the application and the needed level of anonymity. For example, to eliminate the effect of GNSS accuracies, values for b of approximately 50 meters should be sufficient, while trajectory datasets with a higher inaccuracy need a larger buffer radius.

Regarding the parameterization of the V-shaped region for the direction-based attack, the results indicate that the highest data loss lies around $\alpha = 40°$ to $60°$ (Fig. 3.3b). The reason for this non-monotone behavior is that the direction-based attack function f_{d} can be satisfied by two conditions: either all or none of the sites in the protection set must be covered by the V-shaped region. In the case that α is

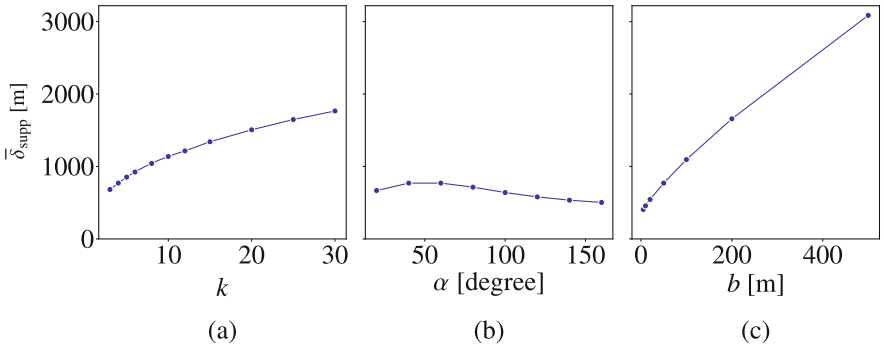

Fig. 3.3 Mean length of suppressed trajectory parts over different parametrization (**a**) of k, $\alpha = 60°$, $b = 50$m; (**b**) of α, $k = 4$, $b = 50$m; and (**c**) of b, $k = 4$, $\alpha = 60°$

very small, and therefore the V-shaped region is very narrow, it is very likely that the V-shape covers none of the sites in the protection set. Analogously, if α is very large, the V-shape is very broad, and all the sites are likely to be covered. Comparing the influence of α to the protection set parameters k and b, α has a significantly smaller impact on $\overline{\delta}_{\text{supp}}$.

For each suppressed trajectory point, the attack function that triggered the suppression is stored to evaluate the importance of the two attack functions against each other. Given the parameters $k = 4$, $\alpha = 60°$, and $b = 0$ m, the direction-based attack function did not trigger the suppression of any trajectory points in 24% of the cases. In other words, the algorithm stopped truncating the trajectories at the first trajectory point located outside their protection cell. Likewise, the share of trajectory points suppressed due to the direction-based criterion was 38%. This means that 62% of the suppressed trajectory points were located in the protection cells.

3.3 Map Matching for Semi-restricted Trajectories

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

- Location measurements are usually imprecise. Thus, constraining the trajectory to a path in a network is useful to get a more faithful movement representation.
- Storing raw data is memory-intensive (especially with high sampling densities). On the other hand, paths in a given network can be stored very compactly; see also Sect. 3.4.
- Matching a trajectory to the road network enables data mining techniques that link attributes of the road network to attributes of the trajectories. This is used to, e.g., deduce routing preferences from given trajectories; see also Sect. 3.5.

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

Hence, the goal is to design an approach that allows for sensible map matching of trajectories that possibly contain on- and off-road sections, also referred to as

semi-restricted trajectories. In the following, an algorithm introduced by Behr et al. (2021) is reviewed that deals with this problem.

3.3.1 Methodology

The algorithm is based on a state transition model where each point of a given trajectory is represented by a set of matching candidates—the possible system states, i.e., possible positions of a moving subject. This is similar to hidden Markov model (HMM)-based map-matching algorithms (Haunert and Budig 2012; Koller et al. 2015; Newson and Krumm 2009), which aim to find a sequence of positions maximizing a product of probabilities. In contrast, the presented algorithm minimizes the sum of energy terms of a carefully crafted model.

Input The algorithm works on the spatial components of a given trajectory $T = \langle p_1, \ldots, p_k \rangle$ of k points in $\mathbb{R}^2$, referred to as GPS points in the following.

As base input, we are given a directed, edge-weighted graph $G(V, E)$ that models the underlying transport network. Every directed edge $uv \in E$ corresponds to a directed straight-line segment representing a road segment with an allowed direction of travel. For an edge e, let $w(e)$ be its *weight*.

Additionally, we assume to be given a set of open spaces such as public squares, parks, or parking lots represented as polygonal areas. The given transport network is extended by triangulating all open spaces and adding tessellation edges as arcs into the graph (Fig. 3.4).

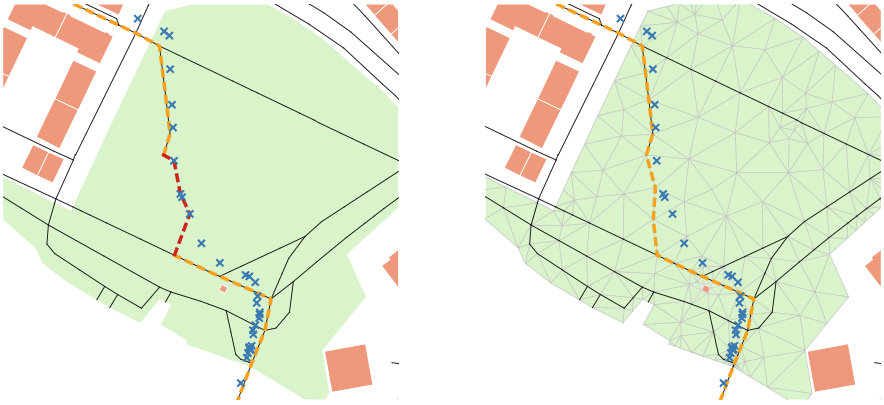

Fig. 3.4 Left: Movement data (blue crosses) on the open space (green) cannot be matched appropriately (dashed orange). An unmatched segment (dashed red) remains. Right: Extended network where open spaces are tessellated to add appropriate off-road candidates. Note that the green polygon has a hole, which remains untessellated since it is not open space

To accurately represent polygons of varying degrees of detail and to provide an appropriate number of off-road candidates, CGAL's meshing algorithm (Boissonnat et al. 2000) is used with an upper bound on the length of the resulting edges and a lower bound on the angles inside triangles.

System States For every p_i, a set of matching *candidates* in the given network is computed by considering a disk D_i of prescribed radius r around p_i and extracting the set of *nodes* $V_i \subseteq V$ within the disk. Furthermore, tessellation nodes (i.e., triangle corners) inside the disk are considered possible matches and included in V_i. Finally, the GPS point itself is added as a candidate to have a fallback. The optimization model ensures that input points are only chosen as output points if all network and tessellation candidates are too far away to be a sensible explanation for this measurement.

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

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

Note that we have three different sets of edges in the graph: the original edges of the road network E_r, the edges incident to unmatched candidate nodes E_u, and the off-road edges on open spaces E_t. On-road matches are preferred over off-road matches, while unmatched candidates are used as a fallback solution and thus should only be selected if no suitable path over on- and off-road edges can be found. To model this, the energy function is adapted by changing the edge weighting w of the graph. Two weighting terms α_t and α_u are introduced that scale the weight $w(e)$ of each edge e in E_t or E_u, respectively.

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

$$\text{Minimize} \quad \mathcal{E}(\langle p_1, \ldots, p_k \rangle, \langle v_1, \ldots, v_k \rangle) = \alpha_\mathrm{c} \cdot \sum_{i=1}^{k} \|p_i - v_i\|^2 + \sum_{i=1}^{k-1} w(P_{i,i+1}) \tag{3.3}$$

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

Matching Algorithm An optimal path, minimizing the introduced energy term, is computed using k runs of Dijkstra's algorithm on a graph that results from augmenting G with a few auxiliary nodes and arcs. More precisely, an incremental algorithm that proceeds in k iterations is used. In the ith iteration, it computes for the subtrajectory $\langle p_1, \ldots, p_i \rangle$ of T and each matching candidate $v \in V_i$ the objective value $\mathcal{E}_i^v$ of a solution $\langle v_1, \ldots, v_i \rangle$ that minimizes $\mathcal{E}(\langle p_1, \ldots, p_i \rangle, \langle v_1, \ldots, v_i \rangle)$ under the restriction that $v_i = v$. This computation is done as follows. For $i = 1$ and any node $v \in V_1$, $\mathcal{E}_1^v$ is simply the state-based energy for v, i.e., $\mathcal{E}_1^v = \alpha_c \cdot \| p_1 - v \|^2$. For $i > 1$, a dummy node s_i and a directed edge $s_i u$ for each $u \in V_{i-1}$ whose weight we set as $\mathcal{E}_{i-1}^u$ are introduced. With this, for any node $v \in V_i$, $\mathcal{E}_i^v$ corresponds to the weight of a minimum-weight s_i-v-path in the augmented graph, plus the state-based energy for v. Thus, for s_i and every node in V_i, a minimum-weight path needs to be found. All these paths can be found with one single-source shortest path query with source s_i and, thus, with a single execution of Dijkstra's algorithm.

3.3.2 Experimental Results

The experimental region is the area around Lake Constance, Germany. For this region, data is extracted from OSM to build a transport network feasible for cyclists and pedestrians. In total, a graph with 931,698 nodes and 2,013,590 directed edges is extracted. The open spaces used for tessellation are identified by extracting polygons with special tags. Spaces with unrestricted movement (e.g., parks) and *obstacles* (e.g., buildings) are identified by lists of tags representing these categories, respectively. Following these tags, 6827 polygons representing open spaces are extracted. Including tessellation edges, the final graph for matching consists of 1,148,213 nodes and 3,345,426 directed edges.

The quality of the approach is analyzed using 58 cycling or walking trajectories. Off-road sections were annotated manually to get a ground truth. The length of the trajectories varies from 300 meters to 41.3 kilometers, and overall, they contain 66 annotated off-road segments.

The energy model parameters were set to $a_u = 10, a_t = 1.1$, and $a_c = 0.07$. Using these values, the precision and recall for off-road section identification were both close to 1.0. A sensitivity study revealed that similar results are achieved for quite large parameter ranges. Furthermore, movement shapes are far better preserved with the presented approach compared to using map matching only on the transport network. This leads to the conclusion that the combined graph consisting of the transport network plus tessellation nodes and edges is only about 50% larger than the road network, but enables a faithful representation of restricted and unrestricted movement at the same time. This then allows applying storage and indexing methods that demand the movement to happen in an underlying network to be also applicable to semi-restricted trajectories.

3.4 Indexing and Querying of Massive Trajectory Sets

Storing and indexing huge trajectory sets in an efficient manner is the very basis for large-scale analysis and mining tasks. An important goal is to compress the input to deal with millions or even billions of trajectories (which individually might consist of hundreds or even thousands of time-stamped positions). But at the same time, we want to be able to retrieve specific subsets of the data (e.g., all trajectories intersecting a given spatiotemporal range) without the necessity to decompress huge parts of the data.

In this section, we present a novel index structure called PATHFINDER that allows to answer range queries on map-matched trajectory sets on an unprecedented scale. Current approaches for the efficient retrieval of trajectory data make use of different dedicated data structures for the two main tasks, compression and indexing. In contrast, our approach elegantly uses an augmented version of the so-called contraction hierarchy (CH) data structure (Geisberger et al. 2012) for both of these tasks. CH is typically used to accelerate route planning queries, but has also proved successful in other settings like continuous map simplification (Funke et al. 2017). This saves space and makes our algorithms relatively simple without the need of too many auxiliary data structures. Only this slenderness allows for scalability to continent-sized road networks and huge trajectory sets. Other existing indexing schemes only work on small network sizes (e.g., PRESS (Song et al. 2014), network of Singapore; PARINET (Sandu Popa et al. 2011), cities of Stockton and Oldenburg; TED (Yang et al. 2018), cities of Singapore and Beijing) or moderate sizes (e.g., SPNET (Krogh et al. 2016): network of Denmark with 800k vertices). Our approach efficiently deals with networks containing millions of nodes and edges.

3.4.1 Methodology

Throughout this section, we assume to be given a trajectory collection $\mathcal{T}$ that already was map matched to an underlying directed graph $G(V, E)$ with V embedded in $\mathbb{R}^2$. We also assume the edges in the graphs to have non-negative costs $c : E \to R^+$. Accordingly, each element in $t \in \mathcal{T}$ is a path $\pi = v_0 v_1 \dots v_k$ in G annotated with timestamps $\tau_0, \tau_1, \dots, \tau_k$.

The goal is to construct an index for $\mathcal{T}$ which allows to efficiently answer queries of the form $[x_l, x_u] \times [y_l, y_u] \times [\tau_l, \tau_u]$ that aim to identify all trajectories which in the time interval $[\tau_l, \tau_u]$ traverse the rectangular region $[x_l, x_u] \times [y_l, y_u]$. In the literature, this kind of query is often named *window* query (Yang et al. 2018) or *range* query (Krogh et al. 2016), where the formal definitions may differ in detail; also see Zheng (2015).

Contraction Hierarchies Our algorithms heavily rely on the *contraction hierarchy* (CH) (Geisberger et al. 2012) data structure, which was originally developed to

speed up shortest path queries. A nice property of CH is that, as a by-product, it also constructs compressed representations of shortest paths.

The CH augments a given weighted graph $G(V, E, c)$ with *shortcuts* and *node levels*. The elementary operation to construct *shortcuts* is the so-called node contraction, which removes a node v and all of its adjacent edges from the graph. To maintain the shortest path distances in the graph, a shortcut $s = (u, w)$ is created between two adjacent nodes u, w of v if the only shortest path from u to w is the path uvw. We define the cost of the shortcut to simply be the sum of the costs of the replaced edges, i.e., $c(s) = c(uv) + c(vw)$. The construction of the CH is the successive contraction of all $v \in V$ in some order; this order defines the *level* $l(v)$ of a node v. The order in which nodes are contracted strongly influences the resulting speed-up for shortest path queries, and hence, many ordering heuristics exist. In our work, we choose the probably most popular heuristic: nodes with low *edge difference*, which is the difference between the number of added shortcut edges and the number of removed edges when contracting a node, are contracted first. We also allow the simultaneous contraction of non-adjacent nodes. As a result, the maximum level of even a continent-sized road network like the one of Europe never exceeds a few hundred in practice. The final CH data structure is defined as $G(V, E^+, c, l)$ where E^+ is the union of E and all shortcuts created.

We also define the nesting depth $nd(e)$ of an edge $e = (v, w)$. If e is an original edge, then $nd(e) = 0$. Otherwise, e is a shortcut replacing edges e_1, e_2, and we define its nesting depth $nd(e) := \max\{nd(e_1), nd(e_2)\} + 1$. Clearly, the nesting depth is upper bounded by the maximum level of a node in the network.

Compression Given a pre-computed CH graph, we construct a CH representation for each trajectory $t \in \mathcal{T}$, that is, we transform the path $\pi = e_0 e_1 \dots e_{k-1}$ with $e_i \in E$ in the original graph into a path $\pi' = e'_0 e'_1 e'_2 \dots e'_{k'-1}$ with $e'_i \in E^+$ in the CH graph.

Our algorithm to compute a CH representation is quite simple: We repeatedly check if there is a shortcut bridging two neighboring edges e_i and e_{i+1}. If so, we substitute them with the shortcut. We do this until there are no more such shortcuts. See Fig. 3.5 for an example.

Note that uniqueness of the CH representation can be proven, and therefore, it does not matter in which order neighboring edges are replaced by shortcuts. The running time of that algorithm is linear in the number of edges. Note that by switching to the CH representation, we can achieve a considerable compression rate in case the trajectory is composed of few shortest paths.

Spatial Indexing and Retrieval The general idea is to associate a trajectory with all edges of its *compressed representation* in E^+. Only due to that compression, it becomes feasible to store a huge number of trajectories within the index. Answering a spatial query then boils down to finding all edges of the CH for which a corresponding path in the original graph intersects the query rectangle. Typically, an additional query data structure would be used for that purpose. Yet, we show how to utilize the CH itself as a geometric query data structure. This requires two central definitions:

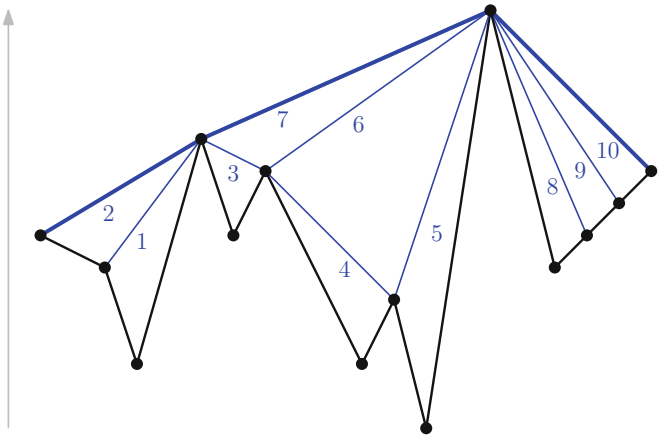

Fig. 3.5 Original path (black, 13 edges) and derivation of its CH representation (bold blue, 3 edges) via repeated shortcut substitution (in order according to the numbers). y-coordinate corresponds to CH level

- With $PB(e)$, we denote the *path box* of an edge e. It is defined as the bounding box for the path that e represents in the original graph G in case $e \in E^+$ is a shortcut or simply the bounding box for the edge e if $e \in E$.
- We define the *downgraph box* $DB(v)$ of a node v as the bounding box of all nodes that are reachable from v on a down-path (only visiting nodes of decreasing CH level), ignoring the orientation of the edges.

Both $PB(e)$ and $DB(v)$ can be computed for all nodes and edges in linear time via a bottom-up traversal of the CH in a preprocessing step and *independent* of the trajectory set to be indexed.

For a spatial-only window query with query rectangle Q, we start traversing the CH level by level in a top-down fashion, first inspecting all nodes which do not have a higher-level neighbor (note that there can be several of them in case the graph is not a single connected component). We can check in constant time for the intersection of the query rectangle and the downgraph box of a node, only continuing with children of nodes with non-empty intersection. We call the set of nodes with non-empty intersection V_I. The set of candidate edges E_O are then all edges adjacent to a node in V_I.

Having retrieved the candidate edges, we have to filter out edges e for which only the path box $PB(e)$ intersects but not the path represented by e. For this case, we recursively unpack e to decide whether e (and the trajectories associated with e) has to be reported. As soon as one child edge reports a non-empty intersection in the recursion, the search can stop, and e must be reported. We call the set of edges that results from this step E_r. Our final query result is all the trajectories which are referenced at least once by the retrieved edges, i.e., those $t \in \bigcup_{e \in E_r} \mathcal{T}_e$.

Temporal Indexing Timestamps of a trajectory t are annotations to its nodes. In the CH representation of t, we omit nodes via shortcuts, hence losing some temporal information. Yet, PATHFINDER will always answer queries conservatively, i.e., returning a superset of the exact result set.

Like the spatial bounding boxes $PB(e)$, we store time intervals to keep track of the earliest and latest trajectory passing over an edge. Similar to $DB(v)$, we compute minimal time intervals containing all time intervals associated with edges on a down-path from v. This allows us to efficiently answer queries which specify a time interval $[\tau_l, \tau_u]$. Like the spatial bounding boxes, we use these time intervals to prune tree branches when they do not intersect the time interval of the query.

An edge is associated with a set of trajectories, each of which we could check for the time when the respective trajectory traverses the edge. A more efficient approach is to store for all trajectories traversing an edge their time intervals in a so-called interval tree (Berg et al. 2008). This allows us to efficiently retrieve the associated trajectories matching the time interval constraint of the query for a given edge. An interval tree storing ℓ intervals has space complexity $O(\ell)$, can be constructed in $O(\ell \log \ell)$, and can retrieve all intervals intersecting a given query interval in time $O(\log \ell + o)$ where o is the output size.

3.4.2 Experimental Results

As input graph data, we extracted the road and path network of Germany from OpenStreetMap. The network has about 57.4 million nodes and 121.7 million edges. CH preprocessing took only a few minutes and roughly doubled the number of edges.

For evaluation, we considered all trajectories within Germany from the OpenStreetMap collection, only dropping low-quality ones (e.g., due to extreme outliers, non-monotonous timestamps, etc.). As a result, we obtained 350 million GPS measurements which were matched to the Germany network with the map matcher from Seybold (2017) to get a dataset with 372,534 trajectories which we call $\mathcal{T}_{\text{ger,real}}$.

Our experiments reveal that on average, a trajectory from our dataset can be represented by 11 shortest paths in the CH graph. While the original edge representation of $\mathcal{T}_{\text{ger,real}}$ consists of 121.8 million edges (992 MB on disk), the CH representation only requires 13.8 million edges (112 MB on disk). The actual compression for the 372k trajectories took 42 seconds, that is, around 0.1ms per trajectory.

To test window query efficiency, we used rectangles of different sizes and different scales of time intervals. Across all specified queries, PATHFINDER was orders of magnitude faster than its competitors, including the naive linear scan baseline but also an inverted index that also benefits from the computed CH graph. Indeed, queries only take a few microseconds per trajectory in the output set.

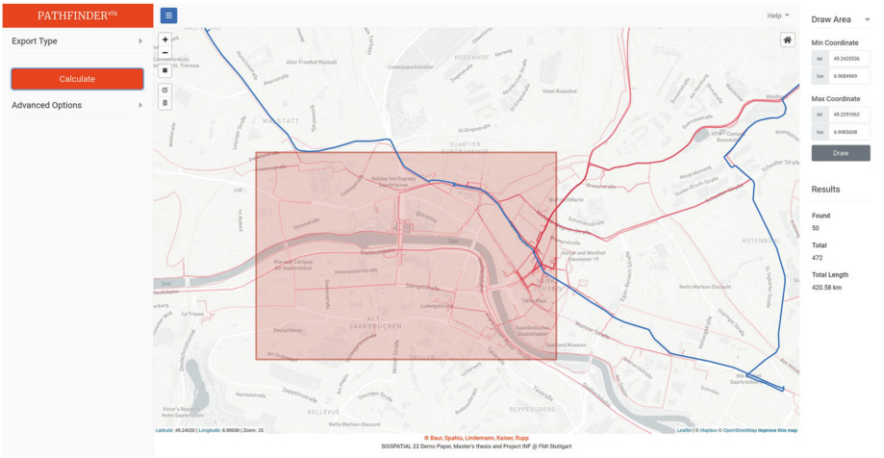

Fig. 3.6 Visualization of trajectory data as developed in Rupp et al. (2022)

Furthermore, the CH-based representation also allows reporting aggregated results easily, which is especially useful for the efficient visualization of large result sets; see, for example, Fig. 3.6, which is a screenshot of our visualization tool presented in Rupp et al. (2022).

3.5 Preference-Based Trajectory Clustering

It is well observable in practice that drivers' preferences are not homogeneous. If we have two alternative paths π_1, π_2 between a given source-target pair, characterized by 3 costs/metrics (travel time, distance, and ascent along the route) each, e.g., $c(\pi_1) = (27\,\text{min}, 12\,\text{km}, 150\,\text{m})$ and $c(\pi_2) = (19\,\text{min}, 18\,\text{km}, 50\,\text{m})$, there are most likely people who prefer π_1 over π_2 and vice versa. The most common model to formalize these preferences assumes a linear dependency on the metrics. This allows for *personalized route planning*, where a routing query consists of not only source and destination but also a weighting of the metrics in the network.

The larger a set of paths is, though, the less likely it is that a single preference/weighting exists which explains all paths, i.e., for which all paths are optimal. One might, for example, think of different driving styles/preferences when commuting versus leisure trips through the countryside. So, a natural question to ask is as follows: what is the minimum number of preferences necessary to explain a set of given paths in a road network with multiple metrics on the edges? This can also be interpreted as a trajectory clustering task, where routes are to be classified according to their purpose. In our example, one might be able to differentiate between commute and leisure. Or in another setting, where routes of different

drivers are analyzed, one might be able to cluster them into speeders and cruisers depending on the routes they prefer.

A use case from recent research of our methods is presented in Chap. 11. In a field study, cyclists were equipped with sensors tracking their exposure to environmental stressors. The study's aim is to evaluate to what degree cyclists are willing to change their routing behavior to decrease their exposure to these stressors. The algorithms presented in the following section can be used to aid this evaluation.

3.5.1 A Linear Preference Model

The following section is based on a *linear* preference model, i.e., for a given directed graph $G(V, E)$, for every edge $e \in E$, a d-dimensional cost vector $c(e) \in \mathbb{R}^d$ is given, where $c_1(e), c_2(e), \ldots, c_d(e) \geq 0$ correspond to non-negative quantities like travel time, distance, non-negative ascent, etc., which are to be minimized. A path $\pi = e_1 e_2 \ldots e_k$ in the network then has an associated cost vector $c(\pi) := \sum_{i=1}^{k} c(e_i)$.

A preference to distinguish between different alternative paths is specified by a vector $\alpha \in [0, 1]^d$, $\sum \alpha_i = 1$. For example, $\alpha^T = (0.4, 0.5, 0.1)$ might express that the respective driver does not care much about ascents along the route, but considers travel time and distance similarly important. Alternative paths π_1 and π_2 are compared by evaluating the respective scalar products of the cost vectors of the path and the preference, i.e., $c(\pi_1)^T \cdot \alpha$ and $c(\pi_2)^T \cdot \alpha$. Smaller scalar values in the linear model correspond to a preferred alternative. An st-path π (which is a path with source s and target t) is optimal for a fixed preference α if no other st-path π' exists with $c(\pi')^T \cdot \alpha < c(\pi)^T \cdot \alpha$. Such a path is referred to as *α-optimal*.

From a practical point of view, it is very unintuitive (or rather almost impossible) for a user to actually express their driving preferences as such a vector α, even if they are aware of the units of the cost vectors on the edges. Hence, it would be very desirable to be able to *infer* their preferences from paths they like or which they have traveled before. In Funke et al. (2016), a technique for preference inference is proposed, which essentially instruments linear programming to determine an α for which a given path π is α-optimal or certify that none exists. Given an st-path π in a road network, in principle, their proposed LP has non-negative variables $\alpha_1, \ldots, \alpha_d$ and one constraint for each st-path π' which states that for the α we are after, π should not be preferred over π'". So the LP looks as follows:

$$
\begin{aligned}
&\max && \alpha_1 && \\
&\forall st\text{-paths } \pi' : && \alpha^T (c(\pi) - c(\pi')) \leq 0 && \textit{optimality constraints} \\
& && \alpha_i \geq 0 && \textit{non-negativity constraints} \\
& && \textstyle\sum \alpha_i = 1 && \textit{scaling constraint}
\end{aligned}
$$

As such, this linear program is of little use, since typically, there is an exponential number of st-paths, so just writing down the complete LP seems infeasible. Fortunately, due to the equivalence of optimization and separation, it suffices to have an algorithm at hand which—for a given α—decides in polynomial time whether all constraints are fulfilled or, if not, provides a violated constraint (such an algorithm is called a *separation oracle*). In our case, this is very straightforward: we simply compute (using, e.g., Dijkstra's algorithm) the optimum st-path for a given α. If the respective path has better cost (w.r.t. α) than π, we add the respective constraint and resolve the augmented LP for a new α; otherwise, we have found the desired α. Via the ellipsoid method, this approach has polynomial running time; in practice, the dual simplex algorithm has proven to be very efficient.

3.5.2 Driving Preferences and Route Compression

The method to infer the preference vector α from a trajectory presented in Sect. 3.5.1 requires that the given trajectory is optimal for at least one preference value α. In practice, for many trajectories, this is not the case. One prominent example is round trips, which are often recorded by recreational cyclists to share with their community. The corresponding trajectories end at the same location as they started. As such, the optimal route for any preference value with respect to the linear model is to not leave the starting point at all, thus having a cost of zero. Note, however, that this problem of not being optimal for any preference value also occurs for many one-way trips. In this section, we review a method by Forsch et al. (2022) that allows us to infer the routing preferences from these trajectories as well. The approach is loosely based on the minimum description length principle (Rissanen 1978), which states that the less information is needed to describe a dataset, the better it is represented. Applied to trajectories, the input trajectory, represented as a path in the road network, is segmented into as few as possible subpaths, such that each of these subpaths is α-optimal. In the following, a compression-based algorithm for the bicriteria case, i.e., when considering two cost functions, is presented. The algorithm is evaluated on cycling trajectories by inferring the importance of signposted cycling routes. Additionally, the trajectories are clustered using the algorithm's results.

3.5.2.1 Problem Formulation

In the bicriterial case, the trade-off between two cost functions c_0 and c_1 for routing is considered. The linear preference model as described in Sect. 3.5.1 simplifies to $c_\alpha = \alpha_0 \cdot c_0 + \alpha_1 \cdot c_1$, which can be rewritten to:

$$c_\alpha = (1 - \alpha_1) \cdot c_0 + \alpha_1 \cdot c_1.$$

Thus, only a single preference value α_1 is present, which is simply denoted with α in the remainder of this section. For each path, the α interval for which the path is α-optimal is of interest. This interval is called the *optimality range* of the given path. The search for the optimality range is formalized in the following problem:

> Given a graph G with two edge cost functions c_0 and c_1, and a path π, find $\mathcal{I}_{\text{opt}} = \{\alpha \in [0, 1] \mid \pi \text{ is } \alpha\text{-optimal}\}$.

Many paths are not optimal for any α, and as such, their optimality range will be empty. For these paths, a minimal milestone segmentation and the corresponding preference value are searched. A *milestone segmentation* is a decomposition of a path π into the minimum number of subpaths $\{\pi_1, \ldots, \pi_h\}$, such that every subpath π_i with $i \in \{1, \ldots, h\}$ is α-optimal for the given α. Thus, the following optimization problem is solved:

> Given a graph G with two edge cost functions c_0 and c_1 and a path π, find $\alpha \in [0, 1]$ and a milestone segmentation of π with respect to α that is as small as possible. That is, there is no $\tilde{\alpha}$ that yields a milestone segmentation of π with a smaller number of subpaths.

The splitting points between the subpaths are called *milestones*. A path π in G can be fully reconstructed given its starting point and endpoint, α, and the corresponding milestones. By minimizing the number of milestones, we achieve the largest compression of the input data over all α values and thus, according to the minimum description length principle, found the preference value that best describes the input path.

3.5.2.2 Solving Milestone Segmentation

Using a separation oracle as described in Sect. 3.5.1, the optimality range of every (sub)path of π is computed in $O(\text{SPQ} \cdot \log(Mn)$ time, where SPQ is the running time of a shortest path query in G, n is the number of vertices in G, and M is the maximum edge cost among all edges regarding c_0 and c_1. For a more detailed description of the algorithm, we refer to Forsch et al. (2022).

Retrieving the milestone segmentation with the smallest number of subpaths is done by first computing a set that contains a milestone segmentation of π for every α and then selecting the optimal milestone segmentation. Existing works in the field of trajectory segmentation use *start-stop matrices* (Alewijnse et al. 2014; Aronov et al. 2016) to evaluate the segmentation criterion. In our case, α-optimality is the segmentation criterion, and the segmentation is not solved for a single α, but the α value minimizing the number of subpaths over all $\alpha \in [0, 1]$ is searched. Therefore, the whole optimality interval for the corresponding subpath is stored in the start-stop matrix $\mathcal{M}$. This allows us to use the same matrix for all α values. Figure 3.7 shows the (Boolean) start-stop matrix for a single α, retrieved by applying the expression $\alpha \in \mathcal{M}[i, j]$ to all elements of the matrix. For a path π consisting of k vertices, a $(k \times k)$-matrix $\mathcal{M}$ of subintervals of $[0, 1]$ is considered. The entry $\mathcal{M}[i, j]$ in row i and column j corresponds to the optimality range of the subpath of π starting at its

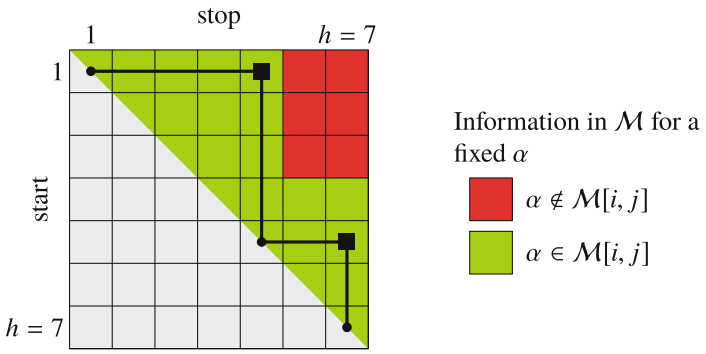

Fig. 3.7 Depiction of a start-stop matrix of a path consisting of $h = 7$ vertices for a fixed $\alpha \in [0, 1]$. The black line represents a minimum segmentation of the path into two subpaths (v_1, v_5, v_7)

ith vertex and ending at its jth vertex. Since we are only interested in subpaths with the same direction as π, we focus on the upper triangle matrix with $i \leq j$.

Due to the optimal substructure of shortest paths (Cormen et al. 2009), for $i < k < j$, both $\mathcal{M}[i, j] \subseteq \mathcal{M}[i, k]$ and $\mathcal{M}[i, j] \subseteq \mathcal{M}[k, j]$ hold. This results in the structure visible in Fig. 3.7, where no red cell is below or left of a green cell. The start-stop matrix is therefore computed starting in the lower-left corner, and each row is filled starting from its main diagonal. That way, the search space for $\mathcal{M}[i, j]$ is limited to the intersection of the already computed values of $\mathcal{M}[i + 1, j]$ and $\mathcal{M}[i, j - 1]$. This is especially useful if one of these intervals is empty, meaning the current interval is empty as well and computation in this row can be stopped.

As a consequence of the substructure optimality explained above, once the start-stop matrix is set up, it is easy to find a solution to the segmentation problem for a fixed α. According to Buchin et al. (2011), for example, an exact solution to this problem can be found with a greedy approach in $O(h)$ time; see Fig. 3.7.

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

3.5.2.3 Experimental Results

The experiments are set up for an examination of the extent to which cyclists stick to official bicycle routes. For this, the following edge costs are defined:

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

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

Hence, in the cost function, edge costs c_0 and c_1 are used that, apart from an edge's geodesic length d, depend only on whether the edge in question is part of an official bicycle route. Using this definition, a minimal milestone segmentation for $\alpha = 0.5$ suggests that minimizing geodesic length outweighs the interest in official cycle paths. A value of $\alpha < 0.5$ indicates avoidance of official cycle paths, whereas $\alpha > 0.5$ indicates the opposite, i.e., preference.

For the analysis, a total of 1016 cycling trajectories from the user-driven platform GPSies[1] are used. The map-matching algorithm described in Sect. 3.3 (without the extended road network) is used to generate paths in the road network, resulting in 2750 paths.[2]

First, the results of the approach on a single trajectory are presented. Figure 3.8 shows the path resulting from map matching the trajectory, as well as a minimal milestone decomposition for $\alpha = 0.55$. In this example, a minimal milestone segmentation is found for $\alpha \in [0.510, 0.645]$, resulting in the milestones a, b, and c. For $\alpha = 0.5$, i.e., the geometric shortest path, a milestone segmentation takes an additional milestone between a and b. For this specific trajectory, the minimal milestone segmentation is found for α values greater than 0.5. This relates to a preference toward officially signposted cycle routes. The increase in milestones for $\alpha > 0.645$ indicates the upper boundary on the detour the cyclist is willing to take to stick to cycle ways.

Based on the compression algorithm, the set of trajectories is divided into three groups `PRO`, `CON`, and `INDIF`. The group `PRO` comprises trajectories for which a milestone segmentation of minimal size is only found for $\alpha > 0.5$. Such a result is interpreted that way that the trajectory was planned in favor of official cycle routes. Conversely, it is considered as avoidance of official cycle routes if milestone segmentations of minimal size exist only for $\alpha < 0.5$. The group `CON` represents these trajectories. Considering trajectories lacking this clarity, no strict categorization into one of the two groups is done. Instead, these trajectories form a third group, `INDIF`. The results of this classification are displayed in Fig. 3.9. The

[1] www.gpsies.com, [no longer available], downloaded May 2018.

[2] Trajectories are split by the algorithm at unmatched segments.

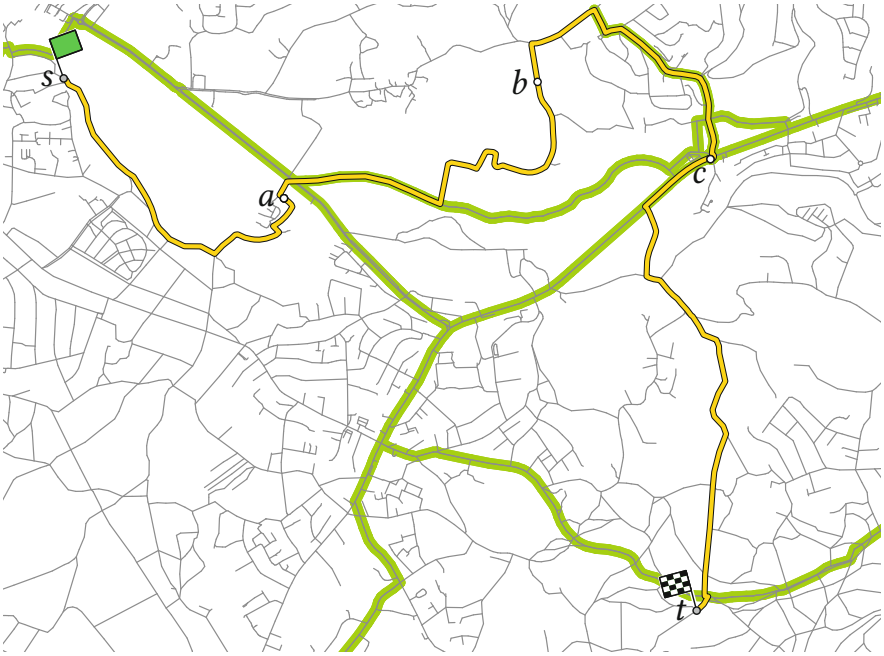

Fig. 3.8 The user's path has an optimal milestone segmentation into four optimal subpaths (s, a, b, c, t) presented by white nodes. Officially signposted paths are highlighted in green. Disregarding minor shiftings of a and/or b, this is a valid optimal segmentation for each $\alpha \in [0.510, 0.645]$. For $\alpha = 0.5$, i.e., the geometric shortest path, a milestone segmentation takes an additional milestone between a and b

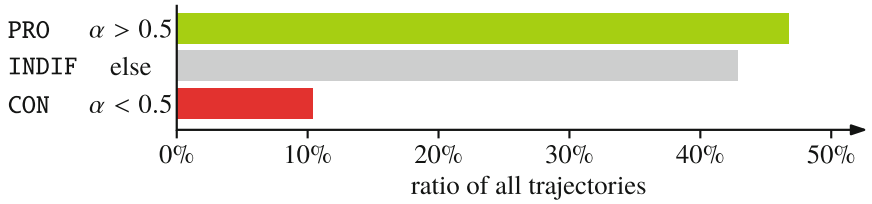

Fig. 3.9 Size of the categories PRO (green, 998 paths), CON (red, 230 paths), and INDIF (gray, 1522 paths)

group PRO being over four times larger than the group CON is a first indicator that cyclists prefer official cycle routes over other roads and paths. The group CON is the smallest group with a share of 8% of all paths. One assumption for this result is that this group mainly consists of road cyclists who prefer using roads over cycle ways. In future research, this could be verified by analyzing a dataset of annotated trajectories.

Overall, more than 50% of the paths are segmented into five α-optimal subpaths or less, resulting in significant compression of the data.

3.5.3 Minimum Geometric Hitting Set

The approach described in Sect. 3.5.1 can easily be extended to decide for a *set* of paths (with different source-target pairs) whether there exists a single preference α for which they are optimal (i.e., which explains this route choice). It does not work, though, if different routes for the same source-target pair are part of the input or simply no single preference can explain all chosen routes. The latter seems quite plausible when considering that one would probably prefer other road types on a leisure trip on the weekend versus the regular commute trip during the week. So, the following optimization problem is quite natural to consider:

> Given a set of trajectories T in a multiweighted graph, determine a set A of preferences of minimal cardinality, such that each $\pi \in T$ is optimal with respect to at least one $\alpha \in A$.

We call this problem *preference-based trajectory clustering* (PTC).

For a concrete problem instance from the real world, one might hope that each preference in the set A then corresponds to a driving style, like speeder or cruiser. Note that while a real-world trajectory often is not optimal for a single α, studies like in Barth et al. (2020) show that it can typically be decomposed into very few optimal subtrajectories if multiple metrics are available.

In Funke et al. (2016), a sweep algorithm is introduced that computes an approximate solution of PTC. It is, however, relatively easy to come up with examples where the result of this sweep algorithm is by a factor of $\Omega(|T|)$ worse than the optimal solution. In Barth et al. (2021), we aim at improving this result by finding practical ways to solve PTC optimally. Our (surprisingly efficient) strategy is to explicitly compute for each trajectory π in T the polyhedron of preferences for which π is optimal and to translate PTC into a geometric hitting set problem.

Fortunately, the formulation as a linear program as described in 3.5.1 already provides a way to compute these polyhedra. The constraints in the above LP exactly characterize the possible values of α for which one path π is optimal. These values are the intersection of half-spaces described by the optimality constraints and the non-negativity constraints of the LP. We call this (convex) intersection *preference polyhedron*.

Using the preference polyhedra, we are armed to rephrase our original problem as a *geometric hitting set (GHS)* problem. In an instance of GHS, we typically have geometric objects (possibly overlapping) in space, and the goal is to find a set of points (a *hitting set*) of minimal cardinality, such that each of the objects contains at least one point of the hitting set. Figure 3.10 shows an example of how preference polyhedra of different optimal paths could look like in the case of three metrics. In terms of GHS, our PTC problem is equivalent to finding a hitting set for the preference polyhedra of minimum cardinality, and the "hitters" correspond to respective preferences. In Fig. 3.10, we have depicted two feasible hitting sets (white squares and black circles) for this instance. Both solutions are minimal in that no hitter can be removed without breaking feasibility. However, the white squares

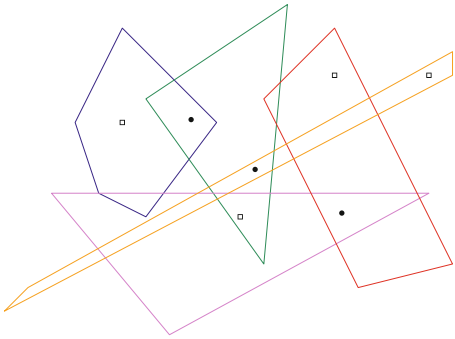

Fig. 3.10 Example of a geometric hitting set problem as it may occur in the context of PTC. Two feasible hitting sets are shown (white squares and black circles)

(in contrast to the black circles) do not describe a minimum solution, as one can hit all polyhedra with fewer points.

While the GHS problem allows picking arbitrary points as hitters, it is not hard to see that it suffices to restrict to vertices of the polyhedra and intersection points between the polyhedra boundaries or more precisely vertices in the arrangement of feasibility polyhedra.

The GHS instance is then formed in a straightforward manner by having all the hitting set candidates as ground set and subsets according to containment in respective preference polyhedra. For an exact solution, we can formulate the problem as an integer linear program (ILP). Let $\alpha^{(1)}, \alpha^{(2)}, \ldots, \alpha^{(l)}$ be the hitting set candidates and $\mathcal{U} := \{P_1, P_2, \ldots, P_k\}$ be the set of preference polyhedra. We create a variable $X_i \in 0, 1$ indicating whether $\alpha^{(i)}$ is picked as a hitter and use the following ILP formulation:

$$\min \sum_i X_i$$

$$\forall\, P \in \mathcal{U} : \sum_{\alpha^{(i)} \in P} X_i \geq 1$$

$$\forall\, i : X_i \in \{0,\ 1\}$$

While solving ILPs is known to be NP-hard, it is often feasible to solve ILPs derived from real-world problem instances, even of non-homeopathic size.

3.5.3.1 Theoretical and Practical Challenges

In theory, the complexity of a single preference polyhedron could be almost arbitrarily high. In this case, computing hitting sets of such complex polyhedra is extremely expensive. To guarantee bounded complexity, we propose to "sandwich" the preference polyhedron between approximating inner and outer polyhedra of bounded complexity.

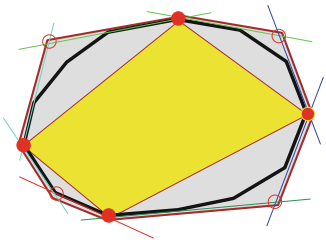

Fig. 3.11 Inner (yellow) and outer approximation (gray) of the preference polyhedron (black)

For d metrics, our preference polyhedron lives in $d-1$ dimensions, so we uniformly ϵ-sample the unit $(d-2)$-sphere using $O((1/\epsilon)^{d-2})$ samples. Each of the samples gives rise to an objective function vector for our linear program; we solve each such LP instance to optimality. This determines $O((1/\epsilon)^{d-2})$ extreme points of the polyhedron in equally distributed directions. Obviously, the convex hull of these extreme points is *contained within* and with decreasing ϵ converges toward the preference polyhedron. Guarantees for the convergence in terms of ϵ have been proven before, but are not necessary for our (practical) purposes. We call the convex hull of these extreme points the *inner approximation* of the preference polyhedron.

What is interesting in our context is the fact that each extreme point is defined by $d-1$ half-spaces. So we can also consider the set of half-spaces that define the computed extreme points and compute their intersection. Clearly, this half-space intersection *contains* the preference polyhedron. We call this the *outer approximation* of the preference polyhedron.

Let us illustrate our approach for a graph with $d=3$ metrics, so the preference polyhedron lives in the two-dimensional plane; see the black polygon/polyhedron in Fig. 3.11. Note that we do not have an explicit representation of this polyhedron, but can only probe it via LP optimization calls. To obtain inner and outer approximation, we determine the extreme points of this implicitly (via the LP) given polyhedron, by using objective functions $\max \alpha_1$, $\max \alpha_2$, $\min \alpha_1$, $\min \alpha_2$. We obtain the four solid red extreme points. Their convex hull (in yellow) constitutes the inner approximation of the preference polyhedron. Each of the extreme points is defined by two constraints (half-planes supporting the two adjacent edges of the extreme points of the preference polyhedron). In Fig. 3.11, these are the light green, blue, dark green, and cyan pairs of constraints. The half-space intersection of these constraints form the *outer approximation* in gray.

3.5.3.2 Experimental Results

For our experiments, we extracted a weighted graph from OpenStreetMap of the German state of Baden-Württemberg with the cost types *distance, travel time for cars*, and *travel time for trucks* containing about 4M nodes and 9M edges. A set of 50 preferences were chosen u.a.r. per instance and created a random source target

Table 3.1 Instance generation and solving for various polyhedra approximations. Car graph with 1000 paths. Time in seconds

Algo.	Polyh. Time	Arr. Time	ILP Sol.	ILP Time
Inner-16	11.3	4.8	45	15.5
Inner-64	26.9	4.9	39	1.8
Exact	6.5	5.2	36	0.7
Outer-64	26.9	5.1	36	0.8
Outer-16	11.3	5.2	36	1.8

optimal for those preferences. Our implementation was evaluated on 24-core Xeon E5-2630 running Ubuntu Linux 20.04.

In Table 3.1, we see the results for a set of 1000 paths. Constructing all the preference polyhedra exactly costs 6.5 seconds and setting up the hitting set instance 5.2 seconds. Solving the hitting set ILP took 0.7 seconds and resulted in a hitting set of size 36. While in theory, the complexity of the preference polyhedra can be almost arbitrarily high, this is not the case in practice. Hence, exact preference polyhedra construction is also more efficient than our proposed approximation approaches (in the table the rows denoted by Inner-XX and Outer-XX), which also only can provide an approximation to the actual hitting set size.

3.6 Visualizing Routing Profiles

In the previous sections, we reviewed algorithms to infer the routing preferences of a user or a group of users from user-generated trajectories. When applying these methods to large trajectory datasets, the trajectories, and thus the users, can be clustered according to their routing behavior. Each cluster has a characteristic routing behavior, expressed as a vector $\alpha \in [0, 1]^d$ of weighting factors of the d different influence factors. On the one hand, these weighting factors are very important for improving route recommendation because they can directly be used to compute optimized future routes. On the other hand, the information on the weighting can be used for planning purposes, e.g., for planning new cycling infrastructure. For this use case, the mathematical representation of the routing preference is not useful, as it is hard to interpret. In this section, we review an alternative representation of routing preferences based on isochrone visualizations geared specifically at a fast and comprehensive understanding of the cyclist's needs. Isochrones are polygons that display the area that is reachable from a given starting point in a specific amount of time and are often used for network analysis, e.g., for public transportation (O'Sullivan et al. 2000) or the movement of electric cars (Baum et al. 2016). For most routing profiles, time is not the only influencing factor. Therefore, the definition of isochrones is slightly altered, and polygons that display areas reachable within a certain amount of *effort* a user is willing to spend instead of a fixed time are used. This effort is dependent on the specific routing preferences.

Even though these polygons do not display times anymore, they will be called isochrones in the following. The visual complexity of the isochrones is reduced by using schematized polygons where the polygon's outline is limited to horizontal, vertical, and diagonal segments. This property is called *octilinearity*. The isochrones created with the presented approach guarantee a formally correct classification of reachable and unreachable parts of the network.

3.6.1 Methodology

The methodology for computing schematic isochrones for specific routing profiles can be split up into two major components: the computation of reachability and the generation of the isochrones.

Computing Reachable Parts of the Road Network The first step for displaying the routing profiles is computing the reachability in the road network, modeled as a directed graph $\mathcal{R}$, for the given profile. Recall the routing model from Sect. 3.5.1, where the personalized cost is the linear combination of the inherent costs in the road network and the user's preference, expressed as α. Thus, given a specific preference, e.g., computed by the methods presented in Sect. 3.5.2 or Sect. 3.5.3, we can compute the personalized costs in our graph. The sum of the personalized costs of the edges along an optimal route from s to t is called the *effort* needed to get from s to t. Shortest path algorithms such as Dijkstra's algorithm (Dijkstra 1959) can be used to compute optimal routes from a given starting location to all other locations in the road network. Such an algorithm is used to compute all nodes in $\mathcal{R}$ that are reachable within a given maximum effort. This information is stored by coloring the reachable nodes blue and the unreachable nodes red. Further, the road network graph $\mathcal{R}$ is planarized to obtain a graph $\overline{\mathcal{R}}$ whose embedding is planar, that is, there are no crossings between two edges. This is done by replacing each edge crossing with an additional node and splitting the corresponding edges at these crossings.

Generating the Isochrones Given the node-colored road network, we proceed by generating the polygon that encloses all reachable nodes, the isochrones. For this, we first extend the node coloring of the graph to an edge coloring by performing the following steps. For each edge e that is incident to both a blue and a red node, we determine the location on e that is still reachable from s; see Fig. 3.12a. At this location, we subdivide e by a blue *dummy node*. A special case occurs when the sum of the remaining distances at two adjacent, reachable nodes u and v is smaller than the length of the edge $e = \{u, v\}$; see Fig. 3.12b. In this case, e is subdivided by two additional dummy nodes, with the middle part being colored red. Further, the coloring and the newly inserted nodes are transferred into the planarized graph $\overline{\mathcal{R}}$, additionally coloring the nodes that have been introduced to planarize $\mathcal{R}$; see Fig. 3.12c. Each such node becomes blue if it subdivides an edge e in $\mathcal{R}$ that is only incident to blue nodes; otherwise, it becomes red. Altogether, we obtain a coloring

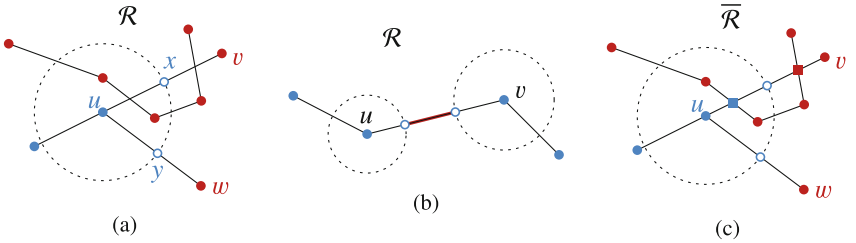

Fig. 3.12 The reachability is modeled by subdividing edges. (**a**) The node x (resp. y) subdivides the edge (u, v) (resp. (u, w)) at the last reachable position. (**b**) Special case: The nodes u and v are reachable from different directions. (**c**) The coloring of $\mathcal{R}$ is transferred to $\overline{\mathcal{R}}$, and the nodes that are introduced for the planarization (squares) are colored

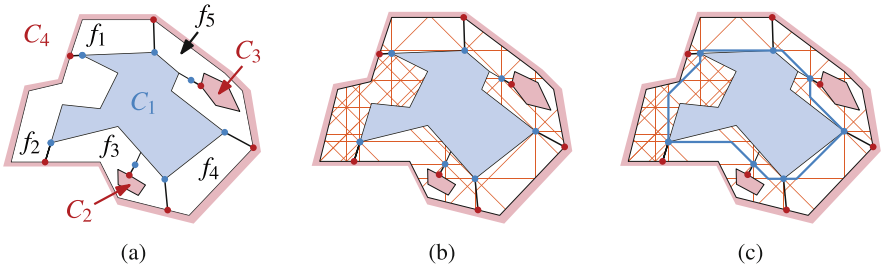

Fig. 3.13 Creating an octilinear polygon enclosing the component C_1 of all reachable nodes and edges. (**a**) The faces $f_1, \ldots, f_5$ surround the component C_1. (**b**) Each face is subdivided by an octilinear grid (Step 1). Furthermore, these grids are connected to one large grid G that is split by the port between f_1 and f_5 (Step 2). (**c**) An octilinear polygon is constructed by computing a bend-minimal path through G (Step 3)

of $\overline{\mathcal{R}}$, defining all nodes either blue or red. Due to the insertion of the dummy nodes, this node coloring induces an edge coloring: edges that are incident to two reachable nodes are also reachable, and all other edges are unreachable.

Given the colored planar graph $\overline{\mathcal{R}}$, we first observe that $\overline{\mathcal{R}}$ has faces that have both red and blue edges. An edge that is incident to both a blue and a red node is called a *gate*. Further, the reachable node of a gate is denoted its *port*. Removing the gates from $\overline{\mathcal{R}}$ decomposes the graph into a set of components $C_1, \ldots, C_\ell$ such that component C_1 is blue and all other components are red. These components are called the *colored components* of $\overline{\mathcal{R}}$. Figure 3.13a shows the gates and the colored components for an example graph.

Given the colored components, we are looking for a single octilinear polygon such that C_1 lies inside and $C_2, \ldots, C_\ell$ lie outside of the polygon. Our method consists of three steps, displayed in Fig. 3.13. In the first step, we create for each pair v_i, v_{i+1} of consecutive ports an octilinear grid G_i contained in f_i. In the second step, we fuse these grids to one large grid G. In the final step, we use G to determine an octilinear polygon by finding an optimal path through G.

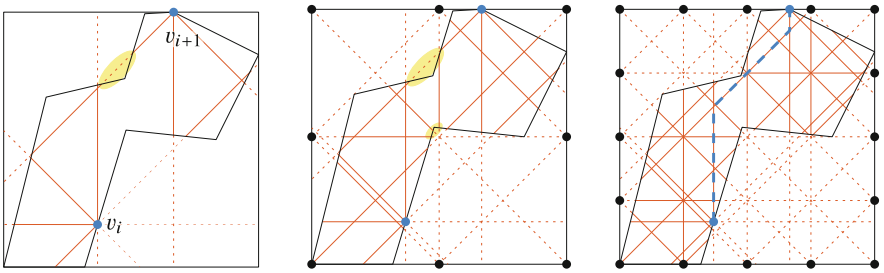

Fig. 3.14 The face f_2 of the example shown in Fig. 3.13 is subdivided by octilinear rays based on vertices on the bounding box. (**a**) Shooting octilinear rays from v_i and v_{i+1} does not yield a connected line arrangement (see yellow highlight). (**b**–**c**) The bounding box of f_1 is successively refined by vertices shooting octilinear rays until they connect v_i and v_{i+1}

Step 1 For each pair v_i, v_{i+1} of consecutive ports with $1 \leq i \leq k$, we first compute an octilinear grid G_i that is contained in f_i and connects v_i and v_{i+1}; see Fig. 3.14. To that end, we shoot from both ports octilinear rays and compute the line segment arrangement $\mathcal{L}$ of these rays restricted to the face f_i; see Fig. 3.14a. If $\mathcal{L}$ forms one component, we use it as grid G_i. Otherwise, we refine $\mathcal{L}$ as follows. We uniformly subdivide the bounding box of f_i by further nodes, from which we shoot additional octilinear rays; see Fig. 3.14b. We insert them into $\mathcal{L}$ restricting them to f_i. We call the number of nodes on the bounding box the *degree of refinement* d. We double the degree of refinement until $\mathcal{L}$ is connected or a threshold $d_{\max}$ is exceeded; see Fig. 3.14c. In the latter case, we also insert the boundary of f_i into $\mathcal{L}$ to guarantee that $\mathcal{L}$ is connected. Later on, when creating the octilinear polygon, we only use the boundary edges of f_i if necessary.

Step 2 In the following, each grid G_i is interpreted as a geometric graph, such that the grid points are the nodes of the graph and the segments connecting the grid points are the edges of the graph. These graphs are unioned into one large graph G. More precisely, G is the graph that contains all nodes and edges of the grids $G_1, \ldots, G_k$. In particular, each port v_i is represented by two nodes x_i and y_i in G such that x_i stems from G_{i-1} and y_i stems from G_i; for $i = 1$, we define $x_1 = v_k$. Two grids G_{i-1} and G_i in G are connected by introducing the directed edge (x_i, y_i) in G for $2 \leq i \leq k$.

Step 3 In the following, let $s := y_1$ and $t := x_1$. A path P from s to t in G is computed such that P has a minimum number of bends, i.e., there is no other path from s to t that has fewer bends. To that end, Dijkstra's algorithm on the linear dual graph of G is used, allowing us to penalize bends in the cost function. In case the choice of P is not unique because there are multiple paths with the same number of bends, the geometric length of the path is used as a tie-breaker, preferring paths that are shorter. As s and t have the same geometric location, the path P forms an octilinear polygon.

In some cases, C_1 contains one or multiple holes, i.e., one or more of the components $C_2, \ldots, C_\ell$ are enclosed by C_1. We deal with this in the following way. For each enclosed component C^*, the road network $\overline{\mathcal{R}}$ is recolored, such that C^* is blue and all other parts of the network are red. We then proceed by computing the enclosing polygon for C^* as outlined for C_1 above and subtracting the resulting polygon from the isochrone of C_1. After this step, the isochrone is guaranteed to contain all parts of the road network that are reachable and none of the parts that are unreachable. We call this the *separation property*.

3.6.2 Application

Recall the classification of cyclists in Sect. 3.5.2. The cyclists are clustered according to their usage of officially signposted cycle routes. Two different classes are introduced, PRO and CON, with preference values of $\alpha_{\texttt{PRO}} > 0.5$ and $\alpha_{\texttt{CON}} < 0.5$. In Fig. 3.15, these two classes are visualized by isochrones with $\alpha = 0.65$ for group PRO and $\alpha = 0.45$ for group CON. In Fig. 3.15a, the isochrones are stretched out very far in the east-west direction along the course of an official cycling route, reflecting the preference of a cyclist in group PRO for officially signposted cycle ways. Moreover, the isochrones are clinched in the north-south direction, revealing a deficit in the road network in this direction for this group of cyclists: while it is possible to get to these areas, the perceived distance will be much longer than in the east-west direction. Figure 3.15b highlights the routing profile for cyclists of the group CON. The isochrones for this routing profile are almost circular, stretching a similar distance in all directions. This is because in most cases, there is a non-signposted road very close to a signposted road, such that avoiding these signposted

(a) PRO ($\alpha = 0.65$) (b) CON ($\alpha = 0.45$)

Fig. 3.15 Two routing profiles visualized as isochrones. Roads (black) highlighted in blue are part of signposted cycling routes. Displayed are the areas that are reachable for a cyclist in group PRO (**a**) and CON (**b**) within four distinct values of maximum effort (encoded by different brightness values)

roads does not involve large detours and is often as easy as just switching to the road from the existing cycling lane. This routing profile is thus probably much less relevant for planning purposes.

3.7 Conclusion and Future Work

We have presented efficient algorithmic approaches to process and mine huge collections of trajectory datasets and demonstrated their usefulness on volunteered data. The next step is integrating our methods as well as other existing ones into an openly available trajectory processing pipeline, which will be flexibly adapted and augmented to cater for a specific application scenario by the user. This would allow others to easily access and benefit from our developed tools. For example, our anonymization as well as indexing and storing methods could be very useful for the VGI challenges discussed in Part III of this book. In the very next chapter, the focus will be on animal trajectories. Here, anonymization is less of an issue, but efficient mining and visualization tools are crucial. With the rich set of applications for trajectory mining occurring across all VGI domains, we see a clear demand for further development and improvements of algorithms to explore and learn from the information hidden in volunteered trajectory collections in the best possible way.

Acknowledgments This research was supported by the German Research Foundation DFG within Priority Research Program 1894 *Volunteered Geographic Information: Interpretation, Visualization, and Social Computing* (VGIscience). In this chapter, results from the projects VGI-Routing (424960421) and Trajectory 2 (314699172) are reviewed.

References

Abul O, Bonchi F, Nanni M (2008) Never walk alone: Uncertainty for anonymity in moving objects databases. In: 2008 IEEE 24th International Conference on Data Engineering, pp. 376–385. https://doi.org/10.1109/ICDE.2008.4497446

Alewijnse SPA, Buchin K, Buchin M, Kölzsch A, Kruckenberg H, Westenberg MA (2014) A framework for trajectory segmentation by stable criteria. In: Proc. 22nd ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems (SIGSPATIAL '14), Dallas Texas, 2014. Association for Computing Machinery, pp. 351–360. https://doi.org/10.1145/2666310.2666415

Aronov B, Driemel A, Kreveld MV, Löffler M, Staals F (2016) Segmentation of trajectories on nonmonotone criteria. ACM Trans Algorithms 12(2):1–28. https://doi.org/10.1145/2660772

Barth F, Funke S, Jepsen TS, Proissl C (2020) Scalable unsupervised multi-criteria trajectory segmentation and driving preference mining. In: BigSpatial@SIGSPATIAL. ACM, New York, pp 6:1–6:10. https://doi.org/10.1145/3423336.3429348

Barth F, Funke S, Proissl C (2021) Preference-based trajectory clustering—an application of geometric hitting sets. In: ISAAC, volume 212 of LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, pp 15:1–15:14. https://doi.org/10.4230/LIPIcs.ISAAC.2021.15

Baum M, Bläsius T, Gemsa A, Rutter I, Wegner F (2016) Scalable exact visualization of isocontours in road networks via minimum-link paths. In: Sankowski P, Zaroliagis C (eds) 24th Annual European Symposium on Algorithms (ESA 2016), volume 57 of Leibniz International Proceedings in Informatics (LIPIcs), Dagstuhl, Germany, 2016. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, pp. 7:1–7:18. https://doi.org/10.4230/LIPIcs.ESA.2016.7

Behr T, van Dijk TC, Forsch A, Haunert J-H, Storandt S (2021) Map matching for semi-restricted trajectories. In: 11th International Conference on Geographic Information Science (GIScience 2021, online). https://doi.org/10.4230/LIPIcs.GIScience.2021.II.12

Berg MD, Cheong O, Kreveld MV, Overmars M (2008) Computational geometry: algorithms and applications, 3rd edn. Springer-Verlag TELOS. ISBN 3540779736, 9783540779735. https://doi.org/10.1007/978-3-540-77974-2

Boissonnat J-D, Devillers O, Teillaud M, Yvinec M (2000) Triangulations in CGAL. In: Proc. 16th Annual Symposium on Computational Geometry (SoCG '00), pp 11–18, 2000. https://doi.org/10.1145/336154.336165

Brauer A, Mäkinen V, Forsch A, Oksanen J, Haunert J-H (2022) My home is my secret: concealing sensitive locations by context-aware trajectory truncation. Int J Geograph Inform Sci 1–29. https://doi.org/10.1080/13658816.2022.2081694

Buchin M, Driemel A, Van Kreveld MJ, Sacristan V (2011) Segmenting trajectories: a framework and algorithms using spatiotemporal criteria. J Spatial Inform Sci 3(3):33–63. https://doi.org/10.5311/JOSIS.2011.3.66

Cormen TH, Leiserson CE, Rivest RL, Stein C (2009) Introduction to algorithms. MIT Press, Cambridge

Dai Y, Shao J, Wei C, Zhang D, Shen HT (2018) Personalized semantic trajectory privacy preservation through trajectory reconstruction. World Wide Web 21(4):875–914. https://doi.org/10.1007/s11280-017-0489-2

Dijkstra EW (1959) A note on two problems in connexion with graphs. Numerische Mathematik 1(1):269–271. https://doi.org/10.1007/bf01386390

Dong, Y, Pi D (2018) Novel privacy-preserving algorithm based on frequent path for trajectory data publishing. Knowl.-Based Syst. 148:55–65. https://doi.org/10.1016/j.knosys.2018.01.007

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

Forsch A, Oehrlein J, Niedermann B, Haunert J-H (2022) Inferring routing preferences of cyclists from user-generated trajectories using a compression criterion. J Spatial Inform Sci. (manuscript submitted on 22nd September 2022)

Funke S, Laue S, Storandt S (2016) Deducing individual driving preferences for user-aware navigation. In: Ravada S, Ali ME, Newsam SD, Renz M, Trajcevski G (eds) Proc. 24th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems (ACM SIGSPATIAL GIS '16), pp 14:1–14:9. https://doi.org/10.1145/2996913.2997004

Funke S, Schnelle N, Storandt S (2017) Uran: a unified data structure for rendering and navigation. In: Web and Wireless Geographical Information Systems. Springer, Cham, pp 66–82. https://doi.org/10.1007/978-3-319-55998-8_5

Funke S, Rupp T, Nusser A, Storandt S (2019) PATHFINDER: storage and indexing of massive trajectory sets. In: Proc. 16th International Symposium on Spatial and Temporal Databases (SSTD '19), pp 90–99. https://doi.org/10.1145/3340964.3340978

Geisberger R, Sanders P, Schultes D, Vetter C (2012) Exact routing in large road networks using contraction hierarchies. Transport Sci 46(3):388–404. https://doi.org/10.1287/trsc.1110.0401

Haunert J-H, Budig B (2012) An algorithm for map matching given incomplete road data. In: Proc. 20th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems (ACM SIGSPATIAL GIS '12), pp 510–513. https://doi.org/10.1145/2424321.2424402

Haunert J-H, Schmidt D, Schmidt M (2021) Anonymization via clustering of locations in road networks. In: 11th International Conference on Geographic Information Science (GIScience 2021)—Part II Short Paper Proceedings. https://doi.org/10.25436/E2CC7P

Huo Z, Meng X, Hu H, Huang Y (2012) You can walk alone: trajectory privacy-preserving through significant stays protection. In: International Conference on Database Systems for Advanced Applications. Springer, Berlin, pp 351–366. https://doi.org/10.1007/978-3-642-29038-1_26

Imielińska C, Kalantari B, Khachiyan L (1993) A greedy heuristic for a minimum-weight forest problem. Oper Res Lett 14(2):65–71. https://doi.org/10.1016/0167-6377(93)90097-Z

Koller H, Widhalm P, Dragaschnig M, Graser A (2015) Fast hidden Markov model map-matching for sparse and noisy trajectories. In: Proc. 18th IEEE International Conference on Intelligent Transportation Systems (ITSC '15). IEEE, pp 2557–2561. https://doi.org/10.1109/ITSC.2015.411

Krogh B, Jensen CS, Torp K (2016) Efficient in-memory indexing of network-constrained trajectories. In: Proc. 24th ACM SIGSPATIAL Int. Conf. on Advances in Geographic Information Systems. ACM, New York, pp 17:1–17:10. ISBN 978-1-4503-4589-7. https://doi.org/10.1145/2996913.2996972

Monreale A, Andrienko GL, Andrienko NV, Giannotti F, Pedreschi D, Rinzivillo S, Wrobel S (2010) Movement data anonymity through generalization. Trans Data Privacy 3(2):91–121.

Newson P, Krumm J (2009) Hidden markov map matching through noise and sparseness. In: Proceedings of the 17th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems (GIS '09). ACM, New York, pp 336–343. https://doi.org/10.1145/1653771.1653818

O'Sullivan D, Morrison A, Shearer J (2000) Using desktop GIS for the investigation of accessibility by public transport: an isochrone approach. Int J Geograph Inform Sci 14(1):85–104. https://doi.org/10.1080/136588100240976

Rissanen J (1978) Modeling by shortest data description. Automatica 14(5):465–471. https://doi.org/10.1016/0005-1098(78)90005-5

Rupp T, Baur L, Funke S (2022) PATHFINDER VIS (Demo Paper). In: SIGSPATIAL/GIS. ACM, New York, pp 11–14.

Sandu Popa I, Zeitouni K, Oria V, Barth D, Vial S (2011) Indexing in-network trajectory flows. J. Int J Very Large Data Bases 20(5):643–669. https://doi.org/10.1007/s00778-011-0236-8

Seybold MP (2017) Robust map matching for heterogeneous data via dominance decompositions. In: Proc. SIAM International Conference on Data Mining, pp 813–821. https://doi.org/10.1137/1.9781611974973.91

Song R, Sun W, Zheng B, Zheng Y (2014) Press: a novel framework of trajectory compression in road networks. Proc VLDB Endow 7(9):661–672. ISSN 2150-8097. https://doi.org/10.14778/2732939.2732940

Sweeney L (2002) k-anonymity: a model for protecting privacy. Int J Uncertainty Fuzziness Knowl.-Based Syst 10(05):557–570. https://doi.org/10.1142/S0218488502001648

Wang N, Kankanhalli MS (2020) Protecting sensitive place visits in privacy-preserving trajectory publishing. Comput Secur 97. https://doi.org/10.1016/j.cose.2020.101949

Yang X, Wang B, Yang K, Liu C, Zheng B (2018) A novel representation and compression for queries on trajectories in road networks. IEEE Trans Knowl Data Eng 30(4):613–629. https://doi.org/10.1109/TKDE.2017.2776927

Yarovoy R, Bonchi F, Lakshmanan LV, Wang WH (2009) Anonymizing moving objects: How to hide a mob in a crowd? In: Proceedings of the 12th International Conference on Extending Database Technology: Advances in Database Technology, pp 72–83. https://doi.org/10.1145/1516360.1516370

Zheng Y (2015) Trajectory data mining: an overview. ACM Trans Intell Syst Technol 6(3):29:1–29:41. ISSN 2157-6904. https://doi.org/10.1145/2743025