Constructing maps by clustering trajectories

Jacobs, R.

Award date:
2016
Constructing Maps by Clustering Trajectories

Master’s Thesis

Roel Jacobs

Supervisor:
dr. K.A. Buchin

Committee Members:
dr. K.A. Buchin
dr. R.M. Pires da Silva Castro
dr. M.A. Westenberg

Eindhoven, 31 August 2016
Abstract

We propose a new approach for constructing the underlying map from trajectory data. This algorithm is based on the idea that road segments can be identified as stable subtrajectory clusters in the data. For this, we consider how subtrajectory clusters evolve for varying distance values, and choose stable values for these, in this way avoiding a global proximity parameter. Within trajectory clusters, we choose representatives, which are combined to form the map. We experimentally evaluate our algorithm on hiking and vehicle tracking data. These experiments demonstrate that our approach can naturally deal with different road widths, and differences in density of the data. It can also to an extent separate roads that run close to each other and can deal with outliers in the data, two issues that are notoriously difficult in road network reconstruction.
Contents

1 Introduction 1

2 Preliminaries 5
   2.1 Trajectories 5
   2.2 Bundles 5
   2.3 Distance 6
      2.3.1 Computing the Distance 7
   2.4 Finding Bundles 9

3 Finding Bundles 13
   3.1 Relevant Bundles 13
      3.1.1 Maximal Bundles 13
      3.1.2 Large Bundles 15
      3.1.3 Stable Bundles 15
   3.2 Algorithms for Generating Bundles 18
      3.2.1 Preprocessing 18
      3.2.2 Fixed Spatial Proximity 21
      3.2.3 Fixed Spatial Proximity and Fixed Size 22
      3.2.4 Determining Evolution of the Bundles 27
   3.3 Further Improvements 27
      3.3.1 Dealing with Parallel Roads 28
      3.3.2 Dealing with Outliers and Sparse Data 29
      3.3.3 Reducing the Number of Iterations 30

4 Constructing the Map 31
   4.1 Greedy Approach 31
      4.1.1 Adding an Edge 31
      4.1.2 Updating the Bundles 32
   4.2 Partially Overlapping Bundles 34
      4.2.1 Alternatives 40
1. Introduction

Street maps and travel networks play an important role in many applications. We can use maps to determine our own location or to navigate an area. Maps are also used in many tools such as for route planning, which allow us to determine a route between two locations. Traditionally, creating these maps is quite labor-intensive and requires a lot of manual work. This includes extensive and expensive field surveying, and manual processing of all the data. This has as a result that street maps typically cover only commercially interesting areas, and they tend to be updated slowly.

Advances in technology such as the global positioning system (GPS) have made it feasible to track the location of a large number of moving entities. These technologies are also being more frequently used, such as in small GPS tracking devices, or embedded into smart phones. As a result, vast amounts of trajectory data exist today. In a large portion of this trajectory data, the moving entity is restricted to move on an underlying network. Consider, for example, a car driving on roads, or a person walking along hiking trails. It is therefore an interesting problem to see how we can use this data to help constructing maps, like in Figure 1.1. This would allow us to get more reliable and recent maps for less commercially interesting areas. Also, we would be able to generate maps for other settings, such as hiking maps, or biking routes.

Volunteered geographic information efforts such as OpenStreetMap make use of free sources, including GPS trajectory data, to construct maps that can complement proprietary and commercial map data. This is, however, still a manual process done by many volunteers, and the quality of the data varies world wide. A large body of recent research aims to reconstruct this underlying network from the trajectory data automatically, in order to aid in the construction of maps; see [3, 4] for surveys. This has turned out to be a challenging task. We here highlight some of these challenges.

One challenge is that we have to deal with noise in the data. Not all tracking devices are as accurate as others, and even though the accuracy has improved over the years, it is by no means perfect. Also, GPS signals may be disturbed due to some external interference. Noise may also be caused by low sampling rates of the tracking devices. If the device only measures the location, say, once a minute, and the movement is assumed to be linear between measurements, while the real movement is over a curvy road, then we may get distorted results.

![Figure 1.1: Using trajectory data to construct an underlying map](image-url)
1. INTRODUCTION

Another challenge is that there is great diversity in the tracking data due to the differences in devices and in the applications they are used for. We may have devices with different sampling frequencies, different accuracies, some only move at slow speeds and cover a small area, whereas others may move at high speeds covering large distances. Also, some areas contain a lot of trajectories with many measurements, whereas in other areas the available data is very limited.

One of the main problems in map construction is to determine to which candidate edge, road or path in the network, a subtrajectory corresponds to. Many of the proposed approaches deal with this by introducing a global spatial parameter modeling the road width; when the subtrajectory is within some distance \( \epsilon \) of the candidate edge \( e \), it is assigned to \( e \). The problem is, however, that in real world data sets there is no global value \( \epsilon \) that accurately captures the road width of all roads in the network. For some road segments we may need a much larger distance threshold than for others. For example, if the road or path is in a canyon, and GPS reception is poor, the trajectories will deviate more from the actual underlying road segment, or simply if a highway is much wider than a back-alley in the center of a city. Even if, after much manual parameter tweaking, the algorithm can be configured to produce a reasonably accurate map on a given data set, the same parameter values often do not give the best map on a different data set. Hence, more manual work is again required.

We propose a new approach for map construction with the aim to alleviate this problem. In particular, our approach does not require a global parameter to model the road width. Instead, we find the desired road width for each edge in the network from the trajectory data itself. More specifically, our algorithm consists of two phases, which are illustrated in Figure 1.2.

In the first phase, the algorithm clusters the input (sub)trajectories into what we call bundles, groups of entities (trajectories) that are close together for a sufficiently long time. Each such bundle may have a different size, length, and spatial “proximity”. The algorithm finds appropriate values for these parameters by considering when the size and length of the bundle change significantly as a function of the spatial parameter \( \epsilon \). In the second phase, the algorithm constructs the actual map (road network) from the bundles. We use a relatively simple greedy approach that constructs the map, starting from the largest bundle.

Our main contribution is the clustering of the trajectories into bundles and automatically selecting relevant bundles. This allows our algorithm to successfully reconstruct network edges of various widths and under noise for hiking and vehicle tracking data. The algorithm also succeeds in separating close, parallel roads under moderate noise. Since it is based on subtrajectory clustering rather than point clustering, it can handle data of varying sampling rates.

Figure 1.2: The two phases of the algorithm. The first phase creates bundles from a set of trajectories, and the second phase creates a map from these bundles.
Related Work

Several different approaches have been proposed in the literature for constructing street maps from trajectory data, see Ahmed et al. [3,4] for an overview. These approaches can be roughly divided into four categories.

Point clustering algorithms [17,21,31] consider the input as a set of points and then cluster these points in different ways to obtain intersections or street segments. This includes algorithms that cluster points based on neighborhood complexes [1,11,13,20]. The algorithm by Edelkamp and Schrödl [17], for example, uses a clustering technique based on the $k$-means method. It first selects an initial set of sample points from the trajectories, and associates both a heading and location with these points. These points act as the initial cluster centers, which are then refined by iteratively adding additional sample points that are close to the clusters. Each step the cluster centers are updated based on the new points, by averaging the position and the heading. After refining, the cluster centers become the nodes of the map, which are connected based on the trajectories passing through the clusters.

Density-based algorithms [8,12,14,24,27,28] first compute a density function over the set of input points, and then extract a road map from this density. For example, Davies et al. [14] use image processing techniques to construct a map. The map area is covered by a rectangular grid, and for each cell a density value is computed. This value is based on the total length of the edges passing through the cell. This grid can be interpreted as a gray scale image. It then tries to find the contours in this image, which are then used to reconstruct the roads. The recent algorithm by Wang et al. [30] utilizes discrete Morse theory to construct a high quality map. These density-based algorithms, however, generally do not fully exploit the continuous traversal information contained in the input trajectories.

Incremental track insertion algorithms [5,10,25,26] insert the trajectories one at a time into an initially empty map, often making use of map-matching ideas. In the algorithm by Ahmed and Wenk [5], each trajectory is added in two steps to an initially empty map. First the trajectory is matched against the partially constructed map to see which parts of the trajectory are already represented. Then in the second step the unmatched parts are added to the graph by introducing additional vertices and edges. In these algorithms, the order in which the trajectories are added may affect the resulting map.

Intersection linking algorithms [19,23] detect the intersection nodes first and then focus on connecting the intersections with edges. Karagiorgou and Pfoser’s algorithm [23] focuses on identifying intersection nodes by detecting changes in movement (speed and direction) and clustering the resulting locations. Trajectory portions between locations in corresponding intersection nodes are then bundled together to form edges.

Organization

This document is organized as follows. In Chapter 2 we introduce some preliminary concepts we use throughout the document, which are mainly related to trajectories and bundles. Chapter 3 describes the first phase of our approach, which is finding relevant bundles. We define here when we consider bundles relevant, and how we can obtain these. This chapter is followed by Chapter 4 which describes the second phase of constructing the map out of a set
of bundles. This is a greedy approach that adds edges to a map based on the bundles one by one. In Chapter 5 we describe and discuss the experimental evaluation of the approach on hiking and urban vehicle data. We construct maps for different datasets, and discuss different properties of these maps. Finally, we describe our conclusions and directions for future work in Chapter 6.
2. Preliminaries

In this paper we present an approach for the map construction problem. This problem can be formulated as follows. Let $\mathcal{T}$ be a set of trajectories, that is, sequences of time-stamped locations. We assume that the entities that generated these trajectories moved on a travel network, represented by a geometric graph $G$. Given the trajectories $\mathcal{T}$, we wish to reconstruct the part of $G$ that the trajectories in $\mathcal{T}$ used.

An important part of our map construction approach is finding subtrajectory clusters, which we refer to as bundles, for a given set of trajectories. In this chapter we introduce and describe important concepts relating to trajectories, bundles, and finding bundles. These definitions are used throughout the rest of this document.

2.1 Trajectories

Our approach uses GPS trajectories, which are sequences of GPS measurements. Such a measurement most often consists of a latitude, longitude, elevation, a time stamp and possibly some additional information such as the number of satellites used. For our approach we do not use the elevation as it is often an attribute with low precision and we do not use the time stamp.

Therefore we interpret a GPS trajectory $T$ with $n$ measurements as a sequence of $n$ points $\langle p_1, \ldots, p_n \rangle$, where $p_i \in \mathbb{R}^2$. We represent the trajectories as polygonal curves, meaning that $T : [1, n] \to \mathbb{R}^2$, with $T(i) = p_i$ for $i \in \{1, \ldots, n\}$ and $T(i + t) = (1 - t)p_i + tp_{i+1}$ for $i \in \{1, \ldots, n - 1\}$ and $t \in [0, 1]$. A subtrajectory of a trajectory $T$ is $T$ restricted to an interval $[a, b]$ with $1 \leq a \leq b \leq n$. We denote the subtrajectory of $T$ restricted to $[a, b]$ as $T[a, b]$. We do not force $a$ and $b$ to be integers, hence the endpoints of $T[a, b]$ may not correspond to any point $p_i$. An example can be seen in Figure 2.1.

![Figure 2.1: Subtrajectories of a trajectory $T$ with four points. In (a) the endpoints of the interval are integers, which makes the subtrajectory start and end at a point of $T$. In (b) the endpoints are not integers, so the subtrajectory starts and ends on edges of $T$.](image)

2.2 Bundles

The first phase of our approach consists of clustering (sub)trajectories. We call these clusters bundles. A set $B$ is a bundle on a set of trajectories $\mathcal{T}$ if it is a set of similar subtrajectories from $\mathcal{T}$. We consider bundles parameterized by their size, their length and their spatial proximity. For the definition of a bundle, we use the definition presented by Buchin et al. [9].

Definition 2.1 (Bundle). Let $T$ be a set of trajectories. A $(k, \ell, \varepsilon)$-bundle on $T$, is a set with at least $k$ subtrajectories from $T$, of which the longest subtrajectory has length $\ell$ and the largest pairwise distance between the subtrajectories is at most $\varepsilon$. Subtrajectories from the same trajectory should not overlap in time.

The different parameters of a bundle are depicted in Figure 2.2. For a bundle $B$, we define the size of $B$, written as $|B|$, as the number of subtrajectories in $B$. We define length of $B$, written as $\text{len}(B)$, as the length of the longest subtrajectory in $B$. Note that the definition of a bundle does not explicitly state which distance measure should be used, however this is usually (a variant of) the Fréchet distance. The distance measure we use is covered in Section 2.3.

![Figure 2.2: The parameters of a bundle of size $k = 5$](image)

2.3 Distance

As a distance measure for computing bundles we use a variation of the Fréchet distance [6]. The Fréchet distance is a distance measure for continuous curves and shapes. It takes into account both the location and the ordering of the points on the curve. Intuitively, it can be imagined as a dog and its owner walking constrained by two separate paths. Both the dog and owner may only walk forward, but they may change speed at any time. The shortest leash length required to connect the owner and the dog is the Fréchet distance. For completeness we state the formal definition here.

Definition 2.2 (Fréchet distance). Let $P = [p, p'] \to \mathbb{R}^2$ and $Q = [q, q'] \to \mathbb{R}^2$ be two curves. The Fréchet distance between $P$ and $Q$, denoted $\delta_F(P, Q)$, is defined as

$$\delta_F(P, Q) = \inf_{\alpha:[0,1] \to P, \beta:[0,1] \to Q} \max_{t \in [0,1]} d(P(\alpha(t)), Q(\beta(t)))$$

where $d(\cdot, \cdot)$ is the Euclidean distance between two points, and $\alpha, \beta$ range over all continuous non-decreasing reparametrizations, with $\alpha(0) = p$, $\alpha(1) = p'$, $\beta(0) = q$ and $\beta(1) = q'$

For polygonal curves, a natural variation of the Fréchet distance is the discrete Fréchet distance [18], which only considers distances between vertices. A problem with using the discrete Fréchet distance is that it may lead to distances that are far off if the vertices of two trajectories are misaligned. Consider for example Figure 2.3. The trajectories are quite close to each other, however, because the vertices are far away from each other, the discrete Fréchet distance will be relatively large.

Something similar happens if one trajectory contains more vertices than another trajectory, as in Figure 2.4. This could be due to differences in the sampling rates of the trajectories.
Because one trajectory has more vertices, some of the vertices of the top trajectory have a large distance to any of the vertices of the bottom trajectory. Hence we would like to still be able to use the distances from the vertices to the edges, yet we would like to keep the freedom of the discrete Fréchet distance that we do not force monotonicity within the edges. Therefore we use a variation of the Fréchet distance that only requires monotonicity for the vertices. Using the man-dog analogy, both are allowed to backtrack, but only within the edges. We refer to this as the vertex-monotone Fréchet distance, which was introduced in [29].

**Definition 2.3** (Vertex-monotone Fréchet distance). Let \( P = [1, n] \to \mathbb{R}^2 \) and \( Q = [1, m] \to \mathbb{R}^2 \) be two polygonal curves with vertices \( \langle p_1, ..., p_n \rangle \) and \( \langle q_1, ..., q_m \rangle \) respectively, such that \( P(1) = p_1, P(n) = p_n, Q(1) = q_1 \) and \( Q(m) = q_m \). The vertex-monotone Fréchet distance between \( P \) and \( Q \), denoted \( \delta_{vmF}(P,Q) \), is defined as

\[
\delta_{vmF}(P, Q) = \inf_{\alpha : [0,1] \to [1, n], \beta : [0,1] \to [1, m]} \max_{t \in [0,1]} d(P(\alpha(t)), Q(\beta(t)))
\]

where \( d(\cdot, \cdot) \) is the Euclidean distance between two points, and \( \alpha, \beta \) range over all continuous reparametrizations, such that

1. For \( s, t \in [0,1], s \leq t \Rightarrow (\alpha(s) \leq \alpha(t) \lor |\alpha(s)| = |\alpha(t)|) \)
2. For \( s, t \in [0,1], s \leq t \Rightarrow (\beta(s) \leq \beta(t) \lor |\beta(s)| = |\beta(t)|) \)
3. \( \alpha(0) = 1 \) and \( \beta(0) \leq 2 \), or \( \beta(0) = 1 \) and \( \alpha(0) \leq 2 \)
4. \( \alpha(1) = n \) and \( \beta(1) \geq m - 1 \), or \( \beta(1) = m \) and \( \alpha(1) \geq n - 1 \)

Conditions 1 and 2 ensure that we have monotonicity for the vertices, but not necessarily for the edges. Note that when \( |\alpha(s)| = |\alpha(t)| \) then we have two points on the same segment, and hence the order does not matter. Conditions 3 and 4 enforce that at the start respectively the end positions only one of the trajectories has to be at a vertex, the other may be on the first, respectively last, edge.

### 2.3.1 Computing the Distance

To compute the vertex-monotone Fréchet distance, we need a geometric data structure called the free-space diagram, introduced by Alt and Godau [6]. This diagram shows for two polygonal curves \( P \) and \( Q \) which pair of points have Euclidean distance of at most \( \varepsilon \) between them, for some \( \varepsilon \), as can be seen in Figure 2.5.
Figure 2.5: Free space diagram (left) for two curves (right)

**Definition 2.4** (Free-space diagram). Let $P$ and $Q$ be two polygonal curves with respectively $n$ and $m$ vertices. The *free-space diagram* between $P$ and $Q$ is the set

$$F_\varepsilon(P,Q) = \{(s,t) \in [1,n] \times [1,m] \mid d(P(s),Q(t)) \leq \varepsilon\}$$

where $d(\cdot,\cdot)$ is the Euclidean distance between two points.

For the regular Fréchet distance, Alt and Godau [6] showed that the Fréchet between two curves $P$ and $Q$ is at most $\varepsilon$ if and only if there is a monotone path in $F_\varepsilon(P,Q)$ from $(0,0)$ to $(n,m)$. So we can answer the decision question “Is the Fréchet distance between $P$ and $Q$ at most $\varepsilon$” by finding such a path.

The vertex-monotone Fréchet distance can be computed in a way similar to that of the discrete Fréchet distance. For the discrete Fréchet distance we only consider the grid points of the free-space diagram, which are labeled free or not. A monotone path is now a sequence of grid points from $(0,0)$ to $(n,m)$, where a point $(i,j)$ is followed by either $(i,j+1), (i+1,j)$ or $(i+1,j+1)$ [18].

Now for the vertex-monotone Fréchet distance, instead of considering the grid points, we consider the grid edges of the free space diagram. Note that such an edge represents the distances from a vertex on one curve to the points on an edge of the other curve. A horizontal grid edge of the shape $((i,j),(i+1,j))$ is labeled free if the shortest distance between the $i$th edge of the first curve and the $j$th vertex of the other curve is at most $\varepsilon$. Similarly, a vertical grid edge of the shape $((i,j),(i,j+1))$ is labeled free if the shortest distance between the $i$th vertex of the first curve and the $j$th edge of the other curve is at most $\varepsilon$. A vertex-monotone path is now a sequence of grid edges such that

- it starts at either $((0,0),(1,0))$ or $((0,0),(0,1))$
- it ends at either $((n-1,m),(n,m))$ or $((n,m-1),(n,m))$
- each edge $((i,j),(i+1,j))$ or $((i,j),(i,j+1))$ is followed by either $((i,j+1),(i+1,j+1))$ or $((i+1,j),(i+1,j+1))$
Similar to the regular Fréchet distance, the vertex-monotone Fréchet distance is at most $\varepsilon$ if and only if there is a vertex-monotone path in the free space. Figure 2.6 shows valid paths for the curves in Figure 2.5.

![Discrete Fréchet distance](a) Discrete Fréchet distance ![Vertex-monotone Fréchet distance](b) Vertex-monotone Fréchet distance

**Figure 2.6: Finding paths in the free-space diagram**

### 2.4 Finding Bundles

In our approach we make use of a modified version of the algorithm for finding bundles by Buchin et al. \[9\]. We give a short description of their algorithm using the discrete Fréchet distance for completeness here. The description of the algorithm is originally for finding bundles for a single trajectory $T$ with itself, but it can easily be adapted to find bundles for a set of trajectories $\mathcal{T}$ by simply joining all trajectories into one long trajectory.

Given a trajectory $T$ with $n$ vertices, a fixed size $k$ and a fixed distance $\varepsilon$, the algorithm tries to find a bundle maximizing the length $\ell$. Note that this algorithm is a 2-distance approximation, meaning that if the optimal solution is a $(k, \ell^*, \varepsilon)$-bundle, it will find a $(k, \ell', 2\varepsilon)$-bundle, with $\ell' \geq \ell^*$. It finds bundles by looking for so-called *cluster curves*.

Consider the free-space diagram $F = F_\varepsilon(T, T)$. Let $s, t \in \{1, \ldots, n\}$ be two points in time for $T$, and let $l_s$ and $l_t$ be the two vertical lines in $F$ corresponding to them. For simplicity we assume that the $x$- and $y$-axis of the diagram goes from 1 to $n$.

**Definition 2.5.** We say that there are $k$ cluster curves between $l_s$ and $l_t$ in $F$ if and only if there are $k$ non-identical curves in $F$ starting at $l_s$ and ending at $l_t$, such that:

- each curve is monotonically increasing in both coordinates
- The $y$-coordinates of two curves overlap in at most a point.

An example of cluster curves can be seen in Figure 2.7. Buchin et al. \[9\] showed that if these $k$ cluster-curves exist, then there exists a $(k, \ell, 2\varepsilon)$-bundle, containing $T[s, t]$. This bundle is
Figure 2.7: Example of four cluster curves between \(l_s\) and \(l_t\). The curves all lie in the free space. Note that the bottom two curves have overlapping \(y\)-coordinates in just a point.

simply the set of \(k\) trajectories corresponding to the cluster-curves. We refer to \(T[s, t]\) as the reference trajectory of the bundle.

To find the curves a sweep-line algorithm is used. Note that \(F\) consists of at at most \(n^2\) grid points which lie in free space. We sweep two lines \(l_s\) and \(l_t\), with \(x\)-coordinates \(s\) and \(t\), over \(F\). The lines represent the start and end points of the reference trajectory \(T[s, t]\). Initially \(l_s\) and \(l_t\) are at the leftmost position in \(F\), and we sweep both lines in discrete steps over \(F\).

The algorithm proceeds as follows. First \(l_t\) is moved to the right until there are less than \(k\) cluster curves between \(l_s\) and \(l_t\). Then \(l_s\) is moved until there are again at least \(k\) cluster curves, then \(l_t\) is moved again. This continues until \(t = n\). During the sweep, the longest bundle is maintained.

To check the number of cluster curves between \(l_s\) and \(l_t\) a special data structure is used. The free space between \(l_s\) and \(l_t\) is stored as a directed, labeled graph on the vertices of the free space. An edge is stored from vertex \(p\) to \(p'\), if both are in the free space of the same cell, and \(p'\) is to the left or below \(p\). The edge is labeled with the smallest \(x\)-coordinate reachable by a path along this edge. An example of such a graph can be seen in Figure 2.8.

The graph is updated whenever a sweep-line is moved. When \(l_t\) is moved new column of the free space is computed at the right of the graph, and when \(l_s\) is moved the leftmost column is deleted.

To find \(k\) cluster curves, the graph is searched greedily. We start at the topmost vertex on \(l_t\) which has an outgoing edge labeled less or equal than \(s\), say \((t, y_t)\). In each vertex we follow the topmost edge which again has a label less or equal than \(s\). This ends on a vertex \((s, y_s)\) on \(l_s\). We now have a cluster curve from \((s, y_s)\) to \((t, y_t)\), which corresponds to the subtrajectory \(T[y_s, y_t]\). We continue with the topmost vertex on \(l_t\) at height at most \(y_s\) which as before has an outgoing edge labeled less or equal than \(s\). We stop when no move valid vertices on \(l_t\) can be found, and we can check if we have found at least \(k\) cluster curves. If this is the case we have found a new \((k, \ell, 2\varepsilon)\)-bundle, for some length \(\ell\). When we find a horizontal path in the free space, that is, a path from \((t, y)\) on \(l_t\) to \((s, y)\) on \(l_s\), we continue with the topmost vertex.
Figure 2.8: Example a labeled graph representing the free space. All paths can reach a vertex with an $x$-coordinate of 1, hence all all edges are labeled with 1.

on $l_t$ at height at most $y - 1$.

In our approach we use the vertex-monotone Fréchet distance instead of the discrete one. The modified version of this algorithm is described in Section 3.2.3.
3. Finding Bundles

The first phase of our approach consists of finding “relevant” bundles. These bundles are used in the second phase to construct the map. In this chapter we define which bundles are “relevant”, and how we obtain these bundles from a set of input trajectories.

3.1 Relevant Bundles

Recall that a \((k, \ell, \varepsilon)\)-bundle is a cluster of size \(k\), length \(\ell\), and distance \(\varepsilon\) of similar subtrajectories. Clearly, every subset of trajectories will form a \((k, \ell, \varepsilon)\)-bundle, for some, or even several, combinations of parameter values \(k, \ell,\) and \(\varepsilon\). Not all of these bundles are useful for us to construct a map. Recall that we would like bundles to represent road segments. If we consider Figure 3.1, we can see many possible bundles for the given trajectories, however one of them seems to represent the road properly.

Figure 3.1: Many possible bundles for a set of three trajectories. Both the red bundles and the blue one are valid bundles for the set of trajectories, however only the blue one seems to represent the road correctly.

Therefore, instead of considering all possible bundles, we want to consider only “relevant” bundles. The properties that we want from our bundles are that they are maximal, stable and large.

3.1.1 Maximal Bundles

Consider \(k\) vertically aligned horizontal line segments of length \(\ell\), spaced \(\varepsilon\) apart. It is easy to see that any subset of the trajectories (line segments) of size \(k'\) forms a \((k', \ell', \varepsilon)\)-bundle, for any length \(\ell' \leq \ell\). This is similar to the situation we saw in Figure 3.1 if we assume the distance between the topmost and bottommost trajectory is at at most \(\varepsilon\). However, in this case the only relevant bundle seems to be the one consisting of all \(k\) trajectories, and length \(\ell\). To capture this inclusion, we introduce the notion of maximal bundles.

We can define a relation between bundles, where one bundle is contained in another bundle. We say that a bundle is a subbundle of another bundle. On one hand, a subbundle is simply a subset of a bundle. However, we allow the subbundle to be of arbitrarily smaller length. For example, the blue bundle in Figure 3.2 is a subbundle of the red bundle.
3. FINDING BUNDLES

Figure 3.2: The red bundle covers the blue bundle, hence the blue bundle is a subbundle of the red one.

**Definition 3.1** (Subbundle). Let $T$ be a set of trajectories. Let $B_1, B_2$ be two bundles on $T$ with $|B_1| \leq |B_2|$. We say $B_1$ is a subbundle of $B_2$, denoted $B_1 \subseteq B_2$, if for every trajectory $T_1 \in B_1$ there is a trajectory $T_2 \in B_2$, such that $T_1$ is a subtrajectory of $T_2$. Conversely, we say that $B_2$ dominates $B_1$, denoted $B_2 \supseteq B_1$.

It can happen that $B_1$ is not a subbundle of $B_2$ by just a small margin, like in Figure 3.3. Such a small margin may not be relevant, as it is often caused by noise. Hence we consider $B_1$ as an “approximate” subbundle of $B_2$. For this we introduce an error-margin $\lambda$. We use it to define approximate subtrajectories that allow us to define approximate subbundles.

Figure 3.3: The red bundle does not cover the blue bundle, but just by a small margin.

**Definition 3.2** ($\lambda$-subtrajectory). Let $T_1$ and $T_2$ be two subtrajectories of the same trajectory. For $\lambda \geq 0$, we call $T_1$ a $\lambda$-subtrajectory of $T_2$, if

$$\text{len}(T_1) \leq \text{len}(T_1 \cap T_2) + \lambda$$

Hence only a section of $T_1$ of length $\lambda$ is not part of $T_2$. We extend this notion to $\lambda$-subbundle and $\lambda$-dominates.

**Definition 3.3** ($\lambda$-subbundle). Let $B_1, B_2$ be two bundles with with $|B_1| \leq |B_2|$. For $\lambda \geq 0$, we say $B_1$ is a $\lambda$-subbundle of $B_2$, denoted $B_1 \subseteq_\lambda B_2$, if for all trajectories $T_1 \in B_1$ there is a trajectory $T_2 \in B_2$ such that $T_1$ is a $\lambda$-subtrajectory of $T_2$. Conversely, we say $B_2$ $\lambda$-dominates $B_1$, denoted $B_2 \supseteq_\lambda B_1$.

Note that $\lambda$ governs the distance between the end-points of trajectories involved. A natural choice would then be to consider $\lambda$ a function of the proximity parameter $\varepsilon$. We set $\lambda = c_\lambda \varepsilon$ for some small constant $c_\lambda$. We choose $c_\lambda = 2$ or $c_\lambda = 3$, to essentially allow an error of $\varepsilon$ on both sides of the bundle, possibly with some additional margin.
Now let $\mathcal{T}$ be a set of trajectories and $B$ a set of bundles on $\mathcal{T}$. The desired property is that we would like every bundle $B \in B$ to be $\lambda$-maximal, meaning that for $B$ there is no other bundle $B' \in B$ such that $B \sqsubseteq_\lambda B'$.

### 3.1.2 Large Bundles

Another property is that we would like bundles to be large enough. The idea is that we would like to have enough evidence for an edge in the network, so we should have sufficiently many “witnesses”. This in particular removes outliers. Hence we require each bundle to have at least $k_{\min}$ trajectories, for some constant $k_{\min}$. Additionally, we require the shortest trajectory in the bundle to have a minimum length of $c_{\text{int}} \varepsilon$, for some small constant $c_{\text{int}}$. This is to prevent bundles representing intersections to end up being relevant. At intersections, we usually have a bundle of large size, as many trajectories meet, but the length of the bundle is proportional to $\varepsilon$, see Figure 3.4. Since we would like bundles to represent roads and not intersections, we eliminate these bundles.

![Figure 3.4: Bundle at an intersection.](image)

### 3.1.3 Stable Bundles

We only consider bundles that are relatively stable with respect to differences in $\varepsilon$. That is, we have to increase $\varepsilon$ by a large amount before new trajectories are added to the bundle, or before the length increases significantly. Note that bundles are monotone in $\varepsilon$, and thus any (sub)trajectory that is present for parameter value $\varepsilon$ will also be present for parameter value $\varepsilon' \geq \varepsilon$. The motivation behind using stability as a measure of relevance, is that we expect that stable bundles are “complete”, meaning that no additional trajectories should be added to them unless we drastically change scale. With respect to map construction, such stable bundles may represent roads, for the current scale. These bundles should be more important than bundles that are “incomplete” and collect additional trajectories if we make small changes to $\varepsilon$. Note that each bundle may have its own value(s) of $\varepsilon$ for which it is stable, and thus each road may have its own road width.

When we increase $\varepsilon$ and inspect the maximal bundles that we have, we can see that some bundles appear for the first time (born), some bundles continue to exist and may increase in length, and some bundles disappear as they merge with another bundle (it becomes a
subbundle of another bundle). This means we can investigate the evolution of bundles with respect to $\varepsilon$ as a tool to determine stability. Consider for example Figure 3.5. We have three trajectories and different maximal bundles for these trajectories for increasing $\varepsilon$.

(a) At $\varepsilon_0$ we have one bundle for every trajectory.
(b) At $\varepsilon_1$, the red and green bundles merge into one new orange bundle. This is a new bundle as the size of the bundle increases.
(c) At $\varepsilon_2$, we get a new purple bundle
(d) At $\varepsilon_3$, we have a new turquoise bundle. At the same time, the purple bundle merges into it.
(e) At $\varepsilon_4$, the turquoise bundle only increases in length. Hence there are no new bundles, and none of the bundles merge
(f) At $\varepsilon_5$, both the orange and the blue bundle merge into the turquoise one and the turquoise one continues to exist.

Figure 3.5: Evolution of bundles for three trajectories. We consider the bundles for increasing values of $\varepsilon$, where $\varepsilon_i < \varepsilon_{i+1}$.

We can visualize the evolution in a diagram like in Figure 3.6. On the vertical axis we have increasing $\varepsilon$. The horizontal axis is only for display purposes. Points represent the moments when a certain bundle appears for the first time, the lines represents when a bundle is alive. For example, the diagram in Figure 3.6 tells us we have a red and a green bundle that live from $\varepsilon_0$ to $\varepsilon_1$, and merge with the orange bundle.

Fix a set of trajectories $S$. We now have that the trajectories in $S$ form a maximal bundle for possibly several disconnected intervals $I_1, I_2, \ldots$. We consider the bundles $B_1, B_2, \ldots$ for
3. FINDING BUNDLES

Each interval $I_i = [\varepsilon_b, \varepsilon_d]$ starts at a value $\varepsilon_b$ at which $B_i$ is born as a maximal bundle, and ends at a value $\varepsilon_d$ at which $B_i$ becomes a subbundle of another bundle (and hence dies as a maximal bundle). We refer to the length of the interval as the \textit{lifespan} of $B_i$, and we define $(\varepsilon_d - \varepsilon_b)/\varepsilon_b = \varepsilon_d/\varepsilon_b - 1$ as the \textit{relative lifespan} of $B_i$. We make use of the relative lifespan, since we for instance expect a bundle that lives between $\varepsilon = 5$ and $\varepsilon = 20$ to be more relevant than one that lives between $\varepsilon = 50$ and $\varepsilon = 65$. We consider each maximal bundle (interval) that has both a large relative lifespan, meaning it is larger than some constant $L_r$, and a large absolute lifespan, meaning it is larger than some constant $L_a$, to be stable: we have to increase $\varepsilon$ significantly before the set of trajectories in the bundle changes.

What remains to be determined is the value of $\varepsilon$ associated with each maximal bundle. This then also fixes the exact length of the bundle. For a bundle $B$, consider its length as a function of $\varepsilon$. More formally, let $\text{len}_B(\varepsilon) = \ell$ if and only if $\ell > 0$ is the largest value for which the entities in $B$ form a $([B], \ell, \varepsilon)$-bundle. See Figure 3.7 for an example. One of the observations is that the length of a bundle may jump a couple of times, for example in Figure 3.7 it jumps between $\varepsilon = 5$ and $\varepsilon = 15$, and it jumps between $\varepsilon = 25$ and $\varepsilon = 30$. This is caused by errors in the trajectories, causing one or more “bottlenecks” that have to be crossed. The length of the bundle is more or less stable until it hits the “bottleneck” distance, after which the length makes a big jump, see Figure 3.8.

To select the value of $\varepsilon$ associated with $B$, we select the starting point of the first interval of length $C$ in which the function is contained in a horizontal slab of height $HC$, for constants $H, C$. That is, the smallest value $\varepsilon$ such that $\text{len}_B(\varepsilon + C) - \text{len}_B(\varepsilon) \leq HC$. This ensures that the bundle is also stable in length, and we do not pick $\varepsilon$ too large to prevent drastic changes in scale. In addition we only pick values in the interval $[0, D]$ for some large $D$, also to prevent drastic changes in scale.

Figure 3.6: Diagram displaying evolution. Points indicate the point where a bundle first appears. Lines indicate for which values of $\varepsilon$ the bundle exists. The number next to the lines indicate the size of the corresponding bundle.
3. FINDING BUNDLES

Figure 3.7: The length \( \text{len}_B \) of a bundle as a function of \( \varepsilon \).

Figure 3.8: Trajectories forming a bundle with a bottleneck

3.2 Algorithms for Generating Bundles

We have now defined which bundles we consider relevant, however, we still have to obtain these bundles. One crucial part is that we need to compute bundles for different values of the proximity parameter \( \varepsilon \), as we need to determine the evolution, and specifically the lifespans and the \( \text{len}_B \) function, of the bundles. Therefore we discretize the range of \( \varepsilon \)-values by picking equally spaced values \( \varepsilon_1, \varepsilon_2, \ldots \) up to some upper-bound \( \varepsilon_{\text{max}} \). We use equally spaced values for two reasons. Many of the computations needed to determine the relevance of bundles involve some \( \varepsilon \) value. It is convenient if all values that we work with are multiples of the same constant \( \Delta \varepsilon \), also when we compare values. Secondly, discretization of the range of \( \varepsilon \)-values introduces an error in the proximity parameter, which is bounded by a constant when we use equally spaced values. For each such \( \varepsilon_i \), we compute a set of maximal bundles \( B_i \), making use of a modified version of the algorithm presented by Buchin et al. [9], which we describe in Section 3.2.3 After that we relate the bundles for different \( \varepsilon \)-values to determine the lifespans of the bundles.

3.2.1 Preprocessing

Before running the main algorithm we preprocess the trajectories, which includes simplifying and segmenting the trajectories. We use segmentation to handle bundles with multiple subtrajectories from the same trajectory, and simplification to speed up the algorithm.
3. FINDING BUNDLES

Simplification

The running time of the algorithm is highly dependent on the number of points in the trajectories. A way to improve the performance is to simplify the trajectories. Simplification aims at reducing the number of points on a trajectory without losing too much information. This is done by constructing a new trajectory with fewer points that approximates the original trajectory. For example, if we have many points along a nearly straight path, we can approximate the trajectory with a single line segment from the first to the last point. Additionally, simplification can help us to remove some noise from trajectories where an object remains at one location for a longer period of time. In such cases we usually get a cluster of points near each other, where we would like to have just a single point, see Figure 3.9. We use Douglas-Peucker simplification [15] to simplify the input trajectories, as it is fast in practice and easy to implement. Alternative approaches could be the algorithm by Agarwal et al. [2], or by Imai and Iri [22].

Figure 3.9: Example where an object remains at one location for a longer period of time. Because of errors in GPS tracking we may get a cluster of many input points next to each other

Segmentation

A natural choice seems to be to allow multiple subtrajectories from the same trajectory to be in the same bundle, as this allows trajectories to loop or go back and forth as in Figure 3.10. We can then consider parts that are traversed multiple times as larger in size (and hence more important).

However, allowing multiple subtrajectories from the same trajectory can lead to potential problems as discussed below. Hence we disallow this, and instead segment trajectories, meaning that we “cut” the trajectories in smaller pieces. We do this in such a way that the pieces from the same bundle are separated. Note that we do not take direction into account within a bundle, that is, subtrajectories may be traversed in either direction.

Figure 3.10: Multiple subtrajectories from the same trajectory
First we briefly describe the problems that arise when allowing multiple subtrajectories per bundle. For large enough $\varepsilon$, we may end up with artificial bundles where small parts from the same trajectory are picked to form a bundle, whereas these parts are not alongside each other, as in Figure 3.11. One could try to prevent this by requiring some $\varepsilon$-proportional gap between subtrajectories of the same trajectory within one bundle. However, this would artificially shorten bundles, which would lead to incorrect merging of bundles, as can be seen in Figure 3.12.

![Figure 3.11: Bundle of size 4 where the subtrajectories are not alongside each other.](image)

**Figure 3.11**: Bundle of size 4 where the subtrajectories are not alongside each other.

Furthermore, when allowing only one subtrajectory of each trajectory per bundle, there is a clear maximal bundle, consisting of all trajectories for a very large $\varepsilon$. Allowing multiple subtrajectories per bundle leads to problems when merging bundles, as there will be bundles that are larger than the number of input trajectories.

The reason we apply segmentation is because we would still like to be able to represent the fact that a trajectory traverses a path multiple times. We can represent each time the path is traversed with a separate segment. Additionally, if we would not apply segmentation at all, we would have to pick a subtrajectory if there are multiple options available, and it is not always clear which choice is best. This choice does not necessarily have to be fixed for all values for $\varepsilon$. This is illustrated in Figure 3.13.

![Figure 3.12: Problem when adding an $\varepsilon$-proportional gap between subtrajectories. One bundle is artificially shortened, causing a bundle to reappear.](image)

**Figure 3.12**: Problem when adding an $\varepsilon$-proportional gap between subtrajectories. One bundle is artificially shortened, causing a bundle to reappear.

Furthermore, when allowing only one subtrajectory of each trajectory per bundle, there is a clear maximal bundle, consisting of all trajectories for a very large $\varepsilon$. Allowing multiple subtrajectories per bundle leads to problems when merging bundles, as there will be bundles that are larger than the number of input trajectories.

The reason we apply segmentation is because we would still like to be able to represent the fact that a trajectory traverses a path multiple times. We can represent each time the path is traversed with a separate segment. Additionally, if we would not apply segmentation at all, we would have to pick a subtrajectory if there are multiple options available, and it is not always clear which choice is best. This choice does not necessarily have to be fixed for all values for $\varepsilon$. This is illustrated in Figure 3.13.

![Figure 3.13: Different choices for selecting a single subtrajectory.](image)

**Figure 3.13**: Different choices for selecting a single subtrajectory.

We segment the trajectories as follows. For each trajectory, we use the bundle-algorithm as described in Section 3.2.3 to find size-2 bundles with itself (and its reverse), for a fixed distance $\varepsilon_{seg}$.

Then for each bundle we check if the subtrajectories are already on different segments, otherwise we split the trajectory in the “gap” between the two subtrajectories. Ideally, we would
like to split into as few segments as possible. To achieve this, we heuristically split at the vertex in the middle of a gap. An example of this can be seen in Figure 3.14.

To avoid segmenting the trajectory too much, we require the bundles to be sufficiently long. That is, the length of the bundle should be at least $2\varepsilon_{seg}$ before it is considered for segmentation. This avoids segmentation in trajectories where there is a little bit of noise.

A related problem is detecting turning points in trajectory data as suggested by Beckmann et al. [7]. Their approach is more specific for turning points, that is, points $p$ where a subtrajectory up to $p$ is similar to the reverse of a subtrajectory starting at $p$ are detected. Beckmann et al. also use the Fréchet distance, hence their algorithm looks for straight paths ending on the diagonal of the free space. We, on the other hand, look for arbitrary monotone paths.

### 3.2.2 Fixed Spatial Proximity

Buchin et al. [9] present an algorithm to find bundles for fixed spatial proximity and fixed size, maximizing length. We, however, only have a fixed spatial proximity, and would like to explore the full range of bundle sizes. Therefore, we fix a bundle size $k$, and use a modified version the bundling algorithm (which is discussed in Section 3.2.3) to find all maximal length bundles. Note that $k$ is the minimum size of a bundle, so it is possible we also find bundles that have larger size. We start with $k = 1$ and increase it by one until no more bundles are found. We can skip values of $k$ if the smallest bundle we find has size $k' > k$. We can then use $k'$ in the next iteration, as for all values in $[k,k']$ we will find the same set of bundles.

Once we have all bundles for all sizes, we remove all bundles that are $\lambda$-dominated by another bundle, to ensure that the bundles are maximal with respect to the other bundles in the set. An overview of the algorithm can be found in Algorithm [3.2.3]. Note that because we do not allow multiple subtrajectories from the same trajectory in a bundle, we could replace the stopping condition in Line 8 by $\mathcal{M} = \emptyset \lor k = |\mathcal{T}|$, as we cannot have a bundle larger than the total number of trajectories.

Since the $\lambda$-subbundle (and hence $\lambda$-dominated) relation is not transitive, the order in which subbundles are removed matters, as can be seen in Figure 3.15. We therefore consider the bundles in order of decreasing size (and decreasing length in case of ties). This ensures that we remove subbundles only if they are a subbundle of a bundle that is selected in the set.
Algorithm 1: GenerateMaximalBundles(\(\mathcal{T}, \varepsilon, \lambda\))

\textbf{input}: Set \(\mathcal{T} = \{T_1, \ldots, T_n\}\) of trajectories. Proximity parameter \(\varepsilon\). Error \(\lambda\).

\textbf{output}: Set of maximal bundles \(\mathcal{B}\).

1. \(k \leftarrow 1\)
2. \(\mathcal{B} \leftarrow \emptyset\)
3. \(\mathcal{M} \leftarrow \emptyset\)
4. repeat
5. \(\mathcal{M} \leftarrow \text{Set of maximal-length } (k, \ell, \varepsilon)\)-bundles on \(\mathcal{T}\)
6. \(\mathcal{B} \leftarrow \mathcal{B} \cup \mathcal{M}\)
7. \(k \leftarrow \max\{k + 1, \min_{B \in \mathcal{M}} |B|\}\)
8. until \(\mathcal{M} = \emptyset\)
9. foreach \(B_1 \in \mathcal{B}\) in order of decreasing size do
10. foreach \(B_2 \in \mathcal{B}\) and \(B_1 \neq B_2\) do
11. if \(B_1 \sqsubseteq_\lambda B_2\) then
12. \(\mathcal{B} \leftarrow \mathcal{B} \setminus \{B_2\}\)
13. return \(\mathcal{B}\)

![Figure 3.15: Three bundles with \(|B_1| < |B_2| < |B_3|\) and \(B_1 \sqsubseteq_\lambda B_2, B_2 \sqsubseteq_\lambda B_3,\) but \(B_1 \not\sqsubseteq_\lambda B_3\). If the bundles are treated in the order \(B_1, B_2, B_3\), then both \(B_1\) and \(B_2\) are removed, and only \(B_3\) remains. This is because treating \(B_2\) causes \(B_1\) to be removed, after which \(B_3\) causes \(B_2\) to be removed. If the bundles are treated in the order \(B_3, B_2, B_1\), however, only \(B_2\) is removed. This is because \(B_3\) removes \(B_2\), meaning \(B_2\) is not treated and \(B_1\) remains.]

3.2.3 Fixed Spatial Proximity and Fixed Size

For fixed size \(k\) and spatial proximity \(\varepsilon\), we use a modified version of the bundling algorithm from [9]. These modifications include the following. First of all, we use the vertex-monotone Fréchet distance instead of the discrete Fréchet distance. Secondly, the original algorithm only reports a single maximum length bundle, we make sure we report all bundles of maximal length. Finally, we make sure we select clusters for which the subtrajectories are reasonably long.

The original algorithm also only finds bundles for a single trajectory with itself, but to solve this we concatenate all input trajectories in \(\mathcal{T}\) into one long trajectory \(T^*\), as suggested in [9].
We then find bundles for $T^*$ and make sure that the subtrajectories we select start and end on the same input trajectory. Additionally, we ignore the directionality of the trajectories, and additionally concatenate all trajectories in $T$ in reverse to $T^*$. We do make sure that we only pick a subtrajectory in one direction.

Using the Vertex-Monotone Fréchet Distance

The original algorithm uses the discrete Fréchet distance when constructing and using the labeled graph, see Section 2.4. For clarity we will refer to the vertices and edges of the labeled graph as labeled vertices and labeled edges respectively.

We now build the labeled graph on the grid edges of the free-space diagram instead of the grid vertices, as can be seen in Figure 3.16. More specifically, every labeled vertex is associated with a grid edge of the free-space diagram, and the labeled edges connect these labeled vertices. In the following we will use “grid edge” and “labeled vertex that is associated with the grid edge” interchangeably.

A labeled edge may only go from an edge at the top or right of a grid cell to a grid edge that is the bottom edge or the left edge of a grid cell. The edge label corresponds to the $x$-coordinate of the lowest/leftmost endpoint of the leftmost reachable grid edge.

![Figure 3.16: Labeled graph on the grid edges of the free-space diagram](image)

As in the original algorithm, the labeled graph is updated every time one of the sweep lines is moved, by adding or removing a column. Before, we had one column for each $x$-coordinate of a grid point of the free-space diagram. Since we now use the grid edges, we have a difference between horizontal grid edges and vertical ones. Therefore, we associate each $x$-coordinate, except for the leftmost one, with two adjacent columns, one for the horizontal grid edges ending at a grid point with that $x$-coordinate, and one for the vertical grid edges starting at a grid point with that $x$-coordinate. See Figure 3.17 for a side-by-side comparison. Whenever we now add or remove a column for a given $x$-coordinate, we add or remove two columns, first the one for the horizontal edges, followed by the one for the vertical edges.

We enforce that the endpoints of the reference trajectory are close enough to the other cluster curves. This means that when searching the labeled graph, we start and end on a vertical grid edge, assuming the reference trajectory is represented by the horizontal axis.

We postprocess the subtrajectories found to avoid having trajectories that are much longer than the reference trajectory, as in Figure 3.18. We check with an $\varepsilon$-disk around the endpoints of the reference trajectory, and remove the parts of the first and last edge of the cluster curves.
Finding Maximal Length Bundles

The original algorithm sweeps two vertical lines over the free space diagram, and keeps track of the bundle whose reference trajectory (“cluster center”) is the longest (measured in number of data points). We modify the algorithm such that it finds all bundles with maximal length, meaning that there is no bundle with larger length that dominates the bundle found. We keep track of the bundle found in every step, and simply report the previous bundle every time the leftmost sweep line is moved. The reason for this is that the left sweep line is only moved
if no bundle can be found for the current reference trajectory, and by moving the sweep line the reference trajectory reduces in length. Therefore, in the previous step, we found a bundle that was locally maximal with respect to the length (of the reference trajectory).

**Finding Long Trajectories**

In the original algorithm, when traversing the labeled graph, from each vertex always the topmost outgoing edge with a correct label is chosen. This ensures that as many cluster curves are found as possible. However, this often results in many small length subtrajectories, next to a longer reference trajectory. We would prefer to have the other subtrajectories in the cluster to be reasonably long as well. Essentially, we would want the minimal length in the bundle to be reasonably large.

Therefore, instead of picking the topmost outgoing labeled edge, we pick the bottommost labeled edge. This ensures the cluster curve found spans the largest interval possible. We have to keep in mind is that we should not search too far down, as the cluster curves that we find must be valid. As indicated, we concatenate all input trajectories in one long trajectory $T^\ast$. When searching for cluster curves, we have to ensure that the curves found start and end on the same trajectory. Implicitly we have a separation between the trajectories that make up $T^\ast$. If we search downwards, however, we risk crossing these imaginary boundaries more easily than if we would always pick the topmost labeled edge.

In order to avoid this situation, we explicitly keep track of the boundaries between the trajectories, and make sure that the labeled graph does not cross the boundaries when constructing it. The free-space diagram essentially is subdivided into different sub grids for which a graph is generated separately, see Figure 3.20. In this way, the longest possible cluster curve we are able to find is a complete input trajectory, not more.

![Figure 3.20: Labeled graph with boundaries. The graph is constructed such that the labeled edged do not cross these boundaries.](image)

Another risk with always choosing the bottom outgoing edge comes from the fact that we use the vertex-monotone Fréchet distance. While traversing the graph, it is possible that we end up at a horizontal grid edge at the bottom of the free-space diagram, or just before a
3. FINDING BUNDLES

boundary between two trajectories, resulting in a dead end. Recall that we would like to end the search through the labeled graph at a vertical grid edge. If we are about to reach the desired $x$-coordinate, we may have two outgoing labeled edges, one going to the desired vertical grid edge, and one going to a horizontal grid edge. Since we always pick the bottom-most edge, we will end up at the horizontal grid edge, which may be a dead end. This is illustrated in Figure 3.21. In this case it would have been desirable to pick the topmost edge.

Figure 3.21: By picking the bottom most outgoing labeled edge, at the end we end up at the bottom of the free-space diagram, giving a dead end. If we would have picked the topmost labeled edge, we would have reached the desired destination.

To solve this we make sure that these dead ends are not reachable. This can be achieved during the construction of the labeled graph. Whenever we try to add a labeled edge to a horizontal grid edge, we check if this horizontal grid edge has any outgoing edges. If this is not the case, we do not add the edge. Note that this effect may propagate through the graph, as can be seen in Figure 3.22. In this way we cannot end up in a dead end, and any path always ends on a vertical grid-edge.

Figure 3.22: Paths leading to a dead end. The red edges are not added to the graph, such that the paths only end on vertical grid edges.

Additionally, we may have situation where we have multiple candidate cluster curves from a single input trajectory. In such case we always prefer the longest subtrajectory, also if the subtrajectories are in opposite directions.
3.2.4 Determining Evolution of the Bundles

Once we have the sets of maximal bundles $B_1, B_2, \ldots$ for $\varepsilon_1, \varepsilon_2, \ldots$, we still have to link bundles for different $\varepsilon_i$ to each other. Only then we are able to determine in which interval a bundle is alive, and from that derive its lifespan.

The approach to determine the evolution of the bundles consists of computing the following three sets for each $\varepsilon_i$.

1. The set $B_{\text{cont},i}$, which consists of the bundles in $B_i$ that are a continuation of a bundle in $B_{i-1}$, meaning it keeps the bundle alive. A bundle $B \in B_i$ is a continuation of a bundle $B' \in B_{i-1}$ if $B \supseteq_\lambda B'$ and $|B| = |B'|$. Therefore both $B$ and $B'$ consist of the same entities, and only differ in length. Note that $B_{\text{cont},1} = \emptyset$.

2. The set $B_{\text{birth},i}$ which consists of the bundles that are born at $\varepsilon_i$. These include all bundles in $B_i$ that do not qualify for the previous condition. Therefore, we have $B_i = B_{\text{cont},i} \cup B_{\text{birth},i}$ and $B_{\text{cont},i} \cap B_{\text{birth},i} = \emptyset$.

3. The set $B_{\text{merge},i}$, which consists of the bundles in $B_{i-1}$ that are now merged with a bundle in $B_i$. We say $B' \in B_{i-1}$ merges with a bundle $B \in B_i$ if $B \supseteq_\lambda B'$, and there is no bundle $B'' \in B_i$ that is a continuation of $B'$. We additionally store the pair $(B', B)$, to track with which bundles other bundles merged. Note that $B_{\text{merge},1} = \emptyset$.

Preferably, we would like $B_{i-1} = B_{\text{merge},i} \cup B_{\text{cont},i}$, but this cannot always be guaranteed. Rarely it may happen that we have a bundle $B \in B_{i-1}$ such that $B \notin B_{\text{merge},i}$ and $B \notin B_{\text{cont},i}$. The reason for this is related to the problems segmentation tries to solve. The segmentation is not perfect, so we may occasionally miss a trajectory that should have been segmented. If we refer back to Figure 3.13, we may actually pick one segment for small $\varepsilon$, and the other segment for larger $\varepsilon$. Because the two segments do not overlap, the bundles we get in both situations are not $\lambda$-subbundles of each other, and hence do not appear in either $B_{\text{merge},i}$ or $B_{\text{cont},i}$. In this case we discard bundle $B$ in any further computations.

Fix a set of trajectories $B$. We can now determine the intervals $I_1, I_2, \ldots$ with $I_i = [\varepsilon_b, \varepsilon_d]$, for which the trajectories in $B$ form maximal bundle as follows:

- $\varepsilon_b = \varepsilon_x$, such that $B \in B_{\text{birth},x}$
- $\varepsilon_d = \varepsilon_y$, such that $y$ is the smallest $x < y$ such that $B \in B_{\text{merge},y}$

3.3 Further Improvements

Here we describe some possible improvements to finding relevant bundles, which may improve the map we obtain in the end.
3. FINDING BUNDLES

3.3.1 Dealing with Parallel Roads

A challenge for many map construction algorithms, including this one, is when we have trajectories over two very close parallel roads. Especially in the presence of noise, it becomes very difficult to distinguish two narrow parallel roads from one wide road, as can be seen in Figure 3.23. In both cases we are likely to get a bundle containing all trajectories that is maximal and stable, giving just a single edge in the map. We would like to combine the information of different bundles to distinguish the two cases.

(a) Parallel roads, no noise    (b) Parallel roads, with noise

(c) Wide road

Figure 3.23: Trajectories on parallel roads versus wide roads.

One observation is that for the case of parallel roads we are very likely to have next to the large bundle also two smaller relevant bundles representing the individual paths. We can use these bundles to check if the large bundle should really represent two parallel roads instead of just a wide one. In Figure 3.24, for instance, we have two relevant bundles $B_1, B_2$ representing the parallel paths, and a relevant bundle $B_3$ containing (parts of) all trajectories of the two parallel paths. We can partition $B_3$ in two bundles $B'_1, B'_2$ such that $B'_1 \subseteq B_1, B'_2 \subseteq B_2$, and $|B'_1| = |B_1|$ and $|B'_2| = |B_2|$. Detecting this situation would allow us to discard $B_3$ as relevant, such that we get parallel paths in the final map.

Figure 3.24: Bundles for two parallel roads can be used to partition a larger bundle.

One important property of $B_1, B_2$ is that they should really represent parallel roads, meaning that they should be “properly” separated, as can be seen in Figure 3.25. If we look at the distance between trajectories in $B_1$ and $B_2$, we expect that within the bundles, the distances between the trajectories are very small. On the other hand, if we compare the distances between trajectories, where one comes from $B_1$ and the other from $B_2$, then these distances should be much larger. In the case of two bundles on a wide road, we expect both the distances within a bundle and the distances between bundles to be similar. We can use this observation to define when the bundles are properly separated.
Figure 3.25: Properly separated bundles (left) versus not properly separated bundles (right).

**Definition 3.4** (Properly separated bundles). Let $B_1, B_2$ be two bundles with $|B_1|, |B_2| > 1$. Let $D$ be the smallest pairwise distance between two trajectories $T \neq T'$, where either $T, T' \in B_1$ or $T, T' \in B_2$. Let $D'$ be the smallest pairwise distance between two trajectories $T \neq T'$, where $T \in B_1$ and $T' \in B_2$.

We say $B_1$ and $B_2$ are properly separated if $D' \geq c_{\text{sep}} D$ for some constant $c_{\text{sep}}$.

To accommodate for noise, we do not require $B_3$ to be completely partitioned, but only some smaller subset $B_3' \subseteq B_3$, such that $|B_3'| \geq \gamma |B_3|$ for some $0 \leq \gamma \leq 1$. In this way $B_1$ and $B_2$ do not have to include noisy trajectories, and we can still identify the presence of two parallel roads. Also, to ensure we do not compare a very small to a very large bundle, the sizes of $B_1$ and $B_2$ may differ by at most a factor two.

To summarize the approach:

1. We select a set of relevant bundles $B$ as before.

2. For each bundle $B \in B$, we check if there are bundles $B_1, B_2$ such that
   
   (a) $B_1$ and $B_2$ are properly separated
   
   (b) $\frac{1}{2} \leq |B_1|/|B_2| \leq 2$

   (c) $B_1$ and $B_2$ can be used to partition a subset $B' \subseteq B$, such that $|B'| \geq \gamma |B|$ for some $0 \leq \gamma \leq 1$. This means $B'$ can be partitioned in two bundles $B_1', B_2'$ such that $B_1' \subseteq B_1, B_2' \subseteq B_2$, and $|B_1'| = |B_1|$ and $|B_2'| = |B_2|$.

3. If such bundles can be found, we remove $B$ from $B$.

### 3.3.2 Dealing with Outliers and Sparse Data

Currently we deal with outliers by requiring relevant bundles to have a minimum size $k_{\text{min}}$. This works particularly well for dense data, as a bundle representing a real road is expected to contain many trajectories, whereas a bundle containing one or more outliers is expected to be considerably smaller in size. This approach can become problematic if the data is very sparse. There may be a few very infrequently visited paths in the travel network, so in our dataset there may be certain paths only one or two trajectories travel over. If we then simply filter based on the size of the bundles, then bundles representing these paths will be removed, and hence they will not end up in the map that is generated. The current approach does not distinguish between these paths and outliers.

We can observe a distinction between most outlier trajectories and most trajectories over unpopular paths, see Figure 3.26. Outliers are very likely to merge with a nearby bundle that represents the road that the outlier was supposed to represent, although this will be for relatively large $\varepsilon$. Trajectories over unpopular paths, on the other hand may eventually merge
with some bundle, but only for even much larger $\varepsilon$, as there may not be any other bundle nearby. We can use this observation to try to distinguish between outliers and sparse data.

Instead of always removing bundles with size less than $k_{\text{min}}$, we can introduce an additional constant $\varepsilon_{\text{size}}$. If a bundle merges with another bundle after $\varepsilon_{\text{size}}$, we drop the size requirement on the bundle when checking for relevance (the maximality and stability constraint remain unchanged). In this way, we can still eliminate outliers that merge before $\varepsilon_{\text{size}}$, but include roads for which there is almost no data that merge after $\varepsilon_{\text{size}}$.

Figure 3.26: Outlier trajectory (red) and a trajectory on a path for which there is very few data available (blue).

3.3.3 Reducing the Number of Iterations

The algorithm performs many iterations of the sweep-line algorithm to find bundles. This has two causes. First, it iterates over all discrete $\varepsilon$-values. Secondly, for each $\varepsilon$ it iterates over all sizes. We can govern the number of $\varepsilon$-steps by changing the distance between two values, and by changing $\varepsilon_{\text{max}}$. This does not influence the number of iterations for all sizes, which is potentially the number of input trajectories.

Instead of always incrementing $k$ by one in Line 7 of Algorithm 1, we could increment using larger values. Alternatively, we could increase $k$ exponentially, for example using powers of two. This can greatly reduce the number of iterations required. One problem is that we may not find all bundles we are interested in, as we do not check certain sizes. This does not have to be a problem. Even if we can only find a bundle containing a subset of the trajectories that move over a certain path, this path can still be represented using this bundle. We may, however, get multiple smaller bundles on one path as the large bundle covering all trajectories is not found.
4. Constructing the Map

Once we have found all relevant bundles we use a simple greedy algorithm to construct the map. The main underlying idea for the algorithm is that each bundle represents a path between two locations in the network. Therefore we represent each bundle by a path/edge, and build the map by adding these one by one. In this chapter we describe the steps in more detail, along with how we solve issues that may arise.

4.1 Greedy Approach

Given a set $B$ of (relevant) bundles on a set of trajectories $T$, we would like to construct a geometric graph $G$ representing the part of the travel network that the trajectories in $T$ used.

We start by associating each bundle $B \in B$ with two points $p_{\text{start}}(B)$ and $p_{\text{end}}(B)$, which represent the start and end location of the path in the graph that the bundle should represent. Additionally, we associate a subtrajectory $\text{rep}(B) \in B$ which acts as the geometric representation of the path. The bundling algorithm from [9] naturally gives such a representative. It is the subtrajectory acting as the “cluster center”, that is, it is the reference subtrajectory that was used in the bundling algorithm to discover $B$. For $p_{\text{start}}(B)$ and $p_{\text{end}}(B)$ we initially pick the start and the end point of $\text{rep}(B)$ respectively. This is illustrated in Figure 4.1.

![Figure 4.1: Bundles with associated endpoints (boxes) and representative (dashed trajectory).](image)

We initially start with an empty map $G$, and incrementally add edges to $G$ based on the bundles in $B$. Note that bundles will be overlapping, in particular larger bundles overlapping longer ones. Hence we consider the bundles in order of decreasing size, and add them to the map one by one. For each bundle $B$ we perform two steps: adding an edge to the graph and updating the bundles

4.1.1 Adding an Edge

We first create two vertices $v_1$ and $v_2$, having the coordinates of $p_{\text{start}}(B)$ and $p_{\text{end}}(B)$ respectively. We add these vertices to $G$, if they do not yet exist. Then we add an edge between $v_1$ and $v_2$ with $\text{rep}(B)$ as representation. See Figure 4.2 for an example.
Due to the updates to the bundles, which is discussed in Section 4.1.2, it is possible that \( p_{\text{start}}(B) \) and \( p_{\text{end}}(B) \) of a bundle \( B \) no longer coincide with the endpoints of the representative. This will then lead to a disconnected parts of the graph, as in Figure 4.3. Therefore, when adding edges to the graph, we connect the start of the representative to \( p_{\text{start}}(B) \) and the end of the representative to \( p_{\text{end}}(B) \) using straight line segments.

4.1.2 Updating the Bundles

After adding the edge for \( B \), we update all other bundles in \( B \), by removing the (sub)trajectories that make up \( B \). Note that this also updates the representatives of the bundles. If this disconnects the trajectories in one of these bundles, we split it in two separate bundles, and we update the endpoints of all remaining bundles appropriately. This step is illustrated in Figure 4.4.

We next describe the update in more detail. Let \( B' \) be a bundle that is updated by \( B \), meaning that the subtrajectories in \( B' \) have parts in common with the subtrajectories in \( B \). We initialize two empty sets \( B_1 \) and \( B_2 \), which hold the new bundle(s). For each trajectory \( T' \in B' \), we find a trajectory \( T \in B \) such that they overlap, that is \( T \cap T' \neq \emptyset \). We then perform the following checks and updates:

- If \( T \cap T' \) includes the first point of \( T' \), then we add \( T' \setminus T \) to \( B_2 \), see Figure 4.5a. If \( T = \text{rep}(B) \), then we set \( \text{rep}(B_2) \) to be \( T' \).
4. CONSTRUCTING THE MAP

Figure 4.4: Updated bundles. Some bundles are split, and the endpoints of the bundles have been updated.

- If $T \cap T'$ includes the last point of $T'$, then we add $T' \setminus T$ to $B_1$, see Figure 4.5b. If $T = \text{rep}(B)$, then we set $\text{rep}(B_1)$ to be $T'$.

- If $T \cap T'$ includes neither the first or start point of $T'$, then $T' \setminus T$ falls apart into two smaller disconnected trajectories $T_1$ and $T_2$, see Figure 4.5c. We add $T_1$ to $B_1$ and $T_2$ to $B_2$. If $T = \text{rep}(B)$, then we set $\text{rep}(B_1)$ to be $T_1$ and $\text{rep}(B_2)$ to be $T_2$.

![Diagram](image)

(a) $T' \setminus T$ added to $B_2$  
(b) $T' \setminus T$ added to $B_1$  
(c) $T_1$ added to $B_1$, $T_2$ added to $B_2$

Figure 4.5: Updating trajectories of a bundle

Now if $B_1$ is not empty, we set $p_{\text{start}}(B_1)$ to $p_{\text{start}}(B')$. Also, we set $p_{\text{end}}(B_1)$ to be the point in $\{p_{\text{start}}(B), p_{\text{end}}(B)\}$ closest to the last point in $\text{rep}(B_1)$. If $\text{rep}(B_1)$ is undefined, we set it to an arbitrary trajectory in $B_1$.

We do something similar if $B_2$ if it is not empty. We set $p_{\text{start}}(B_2)$ to be the point in $\{p_{\text{start}}(B), p_{\text{end}}(B)\}$ closest to the first point in $\text{rep}(B_2)$. If $\text{rep}(B_2)$ is undefined, we set it to an arbitrary trajectory in $B_2$. Secondly, we set $p_{\text{end}}(B_2)$ to $p_{\text{end}}(B')$.

We finally remove $B'$ from $\mathcal{B}$ and add $B_i$ (i.e., $\{1, 2\}$) if they are non-empty, and $p_{\text{start}}(B_i) \neq p_{\text{end}}(B_i)$. The last condition is to prevent self-loops in the map.
4. CONSTRUCTING THE MAP

4.2 Partially Overlapping Bundles

One issue with the approach described in Section 4.1 occurs when we encounter partially overlapping bundles. Two bundles \( B, B' \) are partially overlapping if there is a trajectory \( T \in B \) that overlaps none of the trajectories in \( B' \), and there is a trajectory \( T' \in B' \) that overlaps none of the trajectories in \( B \). This is illustrated in Figure 4.6. This situation may not be very common, however we need to deal with it whenever we encounter it.

Figure 4.6: Partially overlapping bundles. The purple bundle does not properly overlap the red bundle at the top.

We take the following approach. Let \( B \) be a bundle for which we have just added an edge to the graph, and let \( B' \) be a bundle that is updated. Let \( N \) be the set of subtrajectories \( T \in B' \) for which there is no subtrajectory in \( B \) that overlaps \( T \). After we have added all subtrajectories to the bundles \( B_1 \) and \( B_2 \), but before we select the start and endpoints associated with \( B_1 \) and \( B_2 \), we add the trajectories in \( N \) to \( B_1 \) and \( B_2 \) if they are not empty. Also, if \( N \) includes \( \text{rep}(B) \), we set \( \text{rep}(B_1) \) and \( \text{rep}(B_2) \) to \( \text{rep}(B) \). Hence we keep the trajectories in \( N \) in the bundles and simply do not remove anything from them.

Partial overlapping bundles may however occasionally lead to an additional edge in the map. Consider the situation in Figure 4.7. The red and blue bundle partially overlap each other. First the red bundle is processed. Note that it only overlaps three of the trajectories of the blue bundle, however, it overlaps the trajectories entirely. This means that both \( B_1 \) and \( B_2 \) will be empty, and the blue bundle simply disappears (recall that the trajectories in \( N \) are only added if they are not empty). Hence there is no bundle anymore overlapping the green bundle, which will add an additional edge at that location, which may be undesirable in certain situations.

Figure 4.7: Partially overlapping in such a way that it causes an extra edge to appear.
4. CONSTRUCTING THE MAP

4.2.1 Alternatives

Here we briefly discuss some possible alternative strategies for dealing with partially overlapping bundles, and issues that could arise when using them.

Discarding the Bundle being Processed

One way to deal with partially overlapping bundles is to simply discard the bundle that is being added to the graph, as it is the main cause of the problem. In Figure 4.6 this would be the purple bundle. Let $B$ be the bundle that is being processed. Consider a situation, however, where $B$ contains many trajectories of different bundles, and just misses a single subtrajectory of a bundle $B'$, as in Figure 4.8. By discarding $B$ we may lose the information that there is a (big) road and we get multiple edges between two locations in the graph.

Discarding the Bundle being Updated

Another way is to discard the bundle $B'$ that is being updated by some bundle $B$ that was just added to the graph. In Figure 4.6 this would be the red bundle. Now consider a situation where $B'$ consists of many trajectories, and $B$ just overlaps one of them, as in Figure 4.9. By discarding $B'$ we may lose an important road, as there is evidence for many trajectories being close to each other.
Splitting the Bundle being Updated

Instead of discarding any of the bundles, we could modify the bundles in such a way that the partial overlap disappears. Say we have a bundle $B'$ that is being updated by some bundle $B$ that was just added to the graph. In Figure 4.6 this would be the red bundle. We could split $B'$ into two new bundles $B''$ and $B'''$, where $B''$ contains all trajectory that $B$ overlaps, and $B'''$ all trajectories that $B$ does not overlap. Then we only update $B''$ and keep $B'''$ as it is. The problem with this approach is that we introduce an edge to the graph for every bundle. So when splitting the bundle, we essentially add an additional edge. This will in the end lead to multiple edges between two locations where just one is desired.

We can see that our solution and the different alternatives have some advantages and disadvantages. The main advantage of our approach is that it does not remove edges from the map, but it may potentially add a few. We expect the situation of partially overlapping bundles to occur rarely anyway, so it may not be of crucial importance with approach is chosen.
5. Experimental Evaluation

In this chapter, we present an experimental evaluation of our algorithm. We base our evaluation on two recent cross-comparisons of map construction algorithms \[3,16\]. The main difference between both comparisons is that the first one is based on urban vehicle data, while the second one is on hiking data.

The implementations of the algorithms is done in Java. The machine used to run the algorithms has a 2.0 GHz quad-core processor and 8GB of RAM. The algorithms are used without any of the improvements of Section \[3.3\] unless stated otherwise. Also, all distance values in this chapter, in particular the values for the different $\varepsilon$-parameters, are in meters, unless stated otherwise.

5.1 Hiking Networks

5.1.1 Setup

For the hiking data we use the datasets from Duran et al. \[16\], which consists of trajectories from popular geographic areas in Catalonia. We perform two different experiments using this data.

In the first experiment we make use of the cases identified by Duran et al. \[16\] where many map construction algorithms show artifacts. We take small samples of the datasets that highlight these difficult cases, and evaluate how our approach performs on these cases. We also use these samples to investigate the influence of some of the parameters on the maps that are generated. The samples range between 30 and 60 trajectories and between 600 and 10,000 vertices. An overview of the samples is given in Table 5.1, and an image of the datasets in Figure 5.1.

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
<th>Figure</th>
</tr>
</thead>
<tbody>
<tr>
<td>bifurcation</td>
<td>Trajectories at a bifurcation.</td>
<td>5.1a</td>
</tr>
<tr>
<td>close-cross</td>
<td>Trajectories over paths that are relatively close to each other and some cross each other.</td>
<td>5.1b</td>
</tr>
<tr>
<td>curvy</td>
<td>Trajectories over a curvy path.</td>
<td>5.1c</td>
</tr>
<tr>
<td>high-curvature</td>
<td>Trajectories over a curvy path with a turn with very high curvature.</td>
<td>5.1d</td>
</tr>
<tr>
<td>outliers</td>
<td>Set of trajectories with many outliers.</td>
<td>5.1e</td>
</tr>
<tr>
<td>parallel-middle</td>
<td>Trajectories over two parallel roads, and noisy trajectories appearing in between the two trajectories.</td>
<td>5.1f</td>
</tr>
<tr>
<td>parallel</td>
<td>Trajectories over two parallel roads.</td>
<td>5.1g</td>
</tr>
<tr>
<td>sampling</td>
<td>Trajectories with irregular sampling rates</td>
<td>5.1h</td>
</tr>
<tr>
<td>wide</td>
<td>Wide spread trajectories over a path.</td>
<td>5.1i</td>
</tr>
</tbody>
</table>

Table 5.1: Descriptions of the small “Difficult cases” datasets
5. EXPERIMENTAL EVALUATION

In the second experiment we run the algorithm on one of the full datasets, specifically Aiguamolls del Baix Empordà (Aiguamolls), to see how well the algorithms can create full hiking maps. An interesting property of this dataset is that it includes part of a beach on which people can walk freely and are not restricted to a particular path. We highlight interesting features of the map generated.

Some properties of the datasets are summarized in Table 5.2. The parameters used to run the algorithms are summarized in Table 5.3. Note that for the small difficult cases we have used varying values for the simplification parameter, depending on how complex the data was.

5.1.2 Difficult Cases

In the following paragraphs, we briefly discuss the main issues reported by Duran et al. [16] about the behavior of previous map construction algorithms for hiking data, and how our
5. EXPERIMENTAL EVALUATION

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Num. Trajectories</th>
<th>Num. Vertices</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>original</td>
<td>preprocessed</td>
</tr>
<tr>
<td>bifurcation</td>
<td>52</td>
<td>58</td>
</tr>
<tr>
<td>close-cross</td>
<td>37</td>
<td>10</td>
</tr>
<tr>
<td>curvy</td>
<td>51</td>
<td>96</td>
</tr>
<tr>
<td>high-curvature</td>
<td>30</td>
<td>69</td>
</tr>
<tr>
<td>outliers</td>
<td>52</td>
<td>83</td>
</tr>
<tr>
<td>parallel-middle</td>
<td>31</td>
<td>34</td>
</tr>
<tr>
<td>parallel</td>
<td>59</td>
<td>66</td>
</tr>
<tr>
<td>sampling</td>
<td>30</td>
<td>34</td>
</tr>
<tr>
<td>wide</td>
<td>34</td>
<td>83</td>
</tr>
<tr>
<td>Aiguamolls</td>
<td>101</td>
<td>149</td>
</tr>
</tbody>
</table>

Table 5.2: Properties of the hiking datasets

<table>
<thead>
<tr>
<th>Situation</th>
<th>Preprocessing</th>
<th>Finding bundles</th>
<th>Relevant bundles</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>δ</td>
<td>ε_{seg}</td>
<td>c_λ</td>
</tr>
<tr>
<td>Difficult cases</td>
<td>0-5 20</td>
<td>2 5</td>
<td>100</td>
</tr>
<tr>
<td>Full dataset</td>
<td>10 20</td>
<td>2 5</td>
<td>200</td>
</tr>
</tbody>
</table>

Table 5.3: Parameter values used for the hiking data. For the preprocessing, δ is the error used in Douglas-Peucker simplification and ε_{seg} is the segmentation-distance. For bundling c_λ is the scaling factor between λ and ε, Δε is the step size when discretizing the range of ε-values, ε_{max} is the upper bound of the range of ε-values considered. Parameters C, H, D L_a, L_r, c_{int}, k_{min} are used to determine relevant bundles.

algorithm deals with these issues based on the “difficult cases”-samples.

**Noisy trajectories** The ability to deal with noise is foremost among the issues that a map construction algorithm must handle well. A difficult situation occurs when multiple trajectories follow the same path, but the error causes the trajectories to spread around the actual path, being relatively far from each other; see Figure 5.2. For our algorithm, widespread trajectories are not a problem if we allow the proximity parameter for a bundle ε to be large enough to cover them all. This is illustrated by the result on the wide-dataset; see Figure 5.3.

The situation becomes more difficult if the data contains many outliers, as in the outliers-dataset. By increasing the size-threshold k_{min} we can filter these outliers from the set of bundles; see Figure 5.4. The downside of this is that this also removes any bundles at locations where we have very few data available, leading to missing paths in the map.

**Zig-zags and bifurcations** Zig-zagging paths are one of the most challenging situations in hiking data, as witnessed for all algorithms tested by Duran et al. [16]. This is to a large extent due to the fact that proximity thresholds do not work well in zig-zags. Most algorithms
5. EXPERIMENTAL EVALUATION

Figure 5.2: Artifacts in previous algorithm due to noisy parallel trajectories. In this and following similar figures, the blue paths indicate input trajectories, and the red paths the map edges generated. Figure from [16].

Figure 5.3: Bundles (left) and map (right) generated for the wide-dataset, which is the same situation as in Figure 5.2. Edges are colored based on the size of the supporting bundle from blue (small) to red (large).

Figure 5.4: Maps generated for the outliers-dataset, with different thresholds on the sizes of bundles.

either tend to add a large number of shortcuts or to merge the paths close to the turns, as for example in Figure 5.3. Our algorithm is less sensitive to zig-zags than previous ones. We use bundles to reconstruct the different paths, and each bundle is represented by a part of an input trajectory. Therefore the geometry of a zig-zag or winding path is reconstructed reasonably well; see Figure 5.6 for the result on the windy-dataset. Similarly, many previous algorithms have trouble with bifurcations, where we would get artificial shortcuts; see Figure 5.7. Our algorithm has no problem in dealing with this situation, as can be seen in Figure 5.8. The only issue that may arise is that the point where the two roads fork may shift position slightly, as the bundle combining both paths may extend slightly into both arms of the fork. The extent
to which this happens depends on the bundle specific proximity parameter.

Figure 5.5: Previous algorithm producing shortcuts in a zig-zag. Figure from [16].

Figure 5.6: Bundles (left) and map (right) generated for the *curvy*-dataset, which is the same situation as in Figure 5.5. Edges are colored based on the size of the supporting bundle from blue (small) to red (large).

Figure 5.7: Previous algorithm producing shortcut at a bifurcation. Figure from [16].

**Irregular sampling rate** The heterogeneous hiking data sources used by Duran et al. [16] resulted in very irregular sampling rates, something that causes problems to many algorithms. One such problem is that two parallel trajectories, even at a short distance from each other, can be considered far apart based on vertex-based distances. This may give multiple edges in the map for one path, as can be seen in Figure 5.9. Our algorithm deals successfully with this issue by clustering subtrajectories using the vertex-monotone Fréchet distance. So, in this case, a single bundle is properly created, leading to a single path in the map, see Figure 5.10.
5. EXPERIMENTAL EVALUATION

Figure 5.8: Bundles (left) and map (right) generated for the bifurcation-dataset, which is the same situation as in Figure 5.7. Edges are colored based on the size of the supporting bundle from blue (small) to red (large).

Figure 5.9: Previous algorithm producing two edges where there should be one, because of trajectories with different sampling rates. Figure from [16].

Figure 5.10: Bundles (left) and map (right) for the sampling-dataset, where the left of the two bundles/edges correspond to the same situation as in Figure 5.9. Our algorithm correctly detects this situation as one edge, and additionally correctly handles a noisy trajectory and close but separate edges. Edges are colored based on the size of the supporting bundle from blue (small) to red (large).

Back-and-forth trajectories Trajectories that go back and forth are not unusual in hiking data (for example, an out-and-back hike to reach a landmark). However, they cause issues to
many algorithms (some even assume no trajectory does this, for example, [5]). Our algorithm segments the input trajectories to produce trajectories that do not go back and forth on the same path. This is a delicate operation, since a turn with very high curvature may look like such a path, especially when the paths move parallel for some distance after the turn. However, our algorithm can distinguish this, since paths connected by a turn often do not move parallel for very long. An example of this situation, correctly handled by our algorithm, is illustrated in Figure 5.12.

Figure 5.11: Previous algorithm collapsing neighboring parts of a winding path into one edge (the additional edges are because of the noise in the data). Figure from [16].

Figure 5.12: Bundles (left) and map (right) generated for the high-curvature-dataset, which is the same situation as in Figure 5.11. Edges are colored based on the size of the supporting bundle from blue (small) to red (large).

Missing trajectory parts The use of proximity thresholds has the risk of ignoring whole parts of trajectories when they are too close to others. In our algorithm, we make sure that all parts of all trajectories are at least in one bundle. The only way for a trajectory not to be fully represented in the map is when all bundles containing it are considered not relevant. This should only occur if there are very few trajectories at that location. So situations such as shown in Figure 5.13 are unlikely as long as there is enough support for the edge. (See Figure 5.14 for the results of our algorithm in the same spot).

Fragmentation Algorithms that require a minimum number of trajectories along each edge of the map have the risk of producing maps with missing edges or fragments, see Figure 5.15. In our algorithm, each bundle is represented by a single path, so as long as we have a single
5. EXPERIMENTAL EVALUATION

Figure 5.13: Previous algorithm missing vertical connections between trajectories. Figure from [16].

Figure 5.14: Bundles (left) and map (right) generated for the close-cross-dataset, which is the same situation shown in Figure 5.13. Edges are colored based on the size of the supporting bundle from blue (small) to red (large).

For a bundle covering all the relevant trajectories for a given path, we will get a single, connected path in the constructed map. It may, however, happen that there are multiple bundles on the same path, on both sides of some bottleneck on that path, something that could lead to two disconnected edges in the final map. Therefore it is important to ensure that the bundle-specific ε is large enough to cover these (small) bottlenecks.

Figure 5.15: Fragmentation in map by previous algorithm. Figure from [16].

**Parallel paths** Many algorithms have difficulties with trajectories over parallel paths, as it becomes hard to differentiate between a wide path and the two parallel paths. This especially
becomes difficult if the distance between the paths is less than the average GPS error, and when there is a lot of noise in the trajectories. For the case of the parallel-middle-dataset, our algorithm is unable to separate the paths. Because of the trajectories in the middle we get a single bundle containing all trajectories, instead of two separate bundles representing the two paths. Hence the two parallel paths collapse to a single path, see Figure 5.16. The main reason for this to happen is that the trajectories in the middle make the smaller bundles grow in size by changing $\varepsilon$ by just a small amount, meaning that they are not considered stable.

If there are no/fewer trajectories in the middle of the two paths, as in the parallel-dataset, reconstructing the parallel becomes more feasible. If we simply select the bundles using the normal criteria, we get one large bundle combining the parallel paths, just as for the parallel-middle-dataset, as can be seen in Figure 5.17. Since in this case the trajectories on the paths are better separated, we use the strategy as described in Section 3.3.1. If we apply the strategy with $c_{\text{sep}} = 10$ and $\gamma = 0.75$, we can eliminate the large bundles. This allows us to properly reconstruct the parallel paths, as can be seen in Figure 5.18. A downside is that now an edge representing a small side-road becomes disconnected. This is because there are no bundles anymore overlapping the bundle that this edge represents, so we have no information to connect it to the rest of the graph. Because we removed the larger bundles, we are now left with smaller bundles whose proximity parameter is also small, resulting in that none of them is large enough to extend into the small side-road. This may not be too problematic, as the disconnected edge has very low support anyway.

Figure 5.16: Bundles (left) and map (right) generated for the parallel-middle-dataset. Edges are colored based on the size of the supporting bundle from blue (small) to red (large).

Figure 5.17: Bundles (left) and map (right) generated for the parallel-dataset. Edges are colored based on the size of the supporting bundle from blue (small) to red (large).
5. EXPERIMENTAL EVALUATION

Figure 5.18: Bundles (left) and map (right) generated for the parallel-dataset, if the large bundles are removed

5.1.3 Parameter Influence

We can use the small samples to check the influence of some of the parameters on the final result. For some parameters we already briefly showed their influence, for example \( k_{\text{min}} \) allows us to filter outliers. Here we discuss some more parameters.

First consider the parameters for finding bundles. The parameter \( \Delta \varepsilon \) determines the “resolution” in which we can inspect the properties of the bundles. In particular the lifespan and the proximity parameter of a bundle are multiples of \( \Delta \varepsilon \). Hence making this value smaller gives us better precision. It is important not to make this value too large, as that will cause bundles that should not live long have a long lifespan. For example, if we set \( \Delta \varepsilon = 20 \), even a bundle that would have a lifespan of 2 may get a lifespan of at least 20. For the outliers-dataset, if we increase \( \Delta \varepsilon \), we get a couple of small-sized bundles containing outliers that become relevant. This leads to noisy edges which have more support, as can be seen in Figure 5.19.

Figure 5.19: Map for the outliers-dataset, with \( \Delta \varepsilon = 20 \)

We can also vary \( c_\lambda \) to influence the error in the \( \sqsubseteq_\lambda \)-relation. Lower values for this parameter give us a lot more bundles, but they are generally longer, and higher values give us fewer bundles that are generally shorter. Figure 5.20 shows the bundles and the map for different \( c_\lambda \)-values on the sampling-dataset. Recall that for the result in Figure 5.10 we used \( c_\lambda = 2 \). We can see that when decreasing the value to 1, we get a long bundle with many trajectories covering both parallel paths, which causes the paths to collapse into one edge. This is because this bundle now lives longer than was the case for \( c_\lambda = 2 \); long enough to make it relevant. When increasing \( c_\lambda \) to 3 we have almost no change in the result when compared \( c_\lambda = 2 \). If we increase it all the way to 10, then we only get a very few short bundles, which cause an additional vertex to appear due to the way the bundles overlap.

Next we consider some of the parameters for finding relevant bundles. First consider \( L_a \) and \( L_r \), which determine which bundles are considered stable enough. Note that \( L_a \) only makes
5. EXPERIMENTAL EVALUATION

(a) $c_\lambda = 1$
(b) $c_\lambda = 3$
(c) $c_\lambda = 10$

Figure 5.20: Bundles and maps generated for the sampling-dataset, with different values for $c_\lambda$.

There is a difference in the selected bundles if we change it with multiples of $\Delta \varepsilon$, as the absolute lifespan of the bundles will be multiples of $\Delta \varepsilon$. For both parameters, lower values give us more bundles that are overall less stable, and higher values give us fewer bundles that are overall more stable. Additionally, high values for $L_r$ promote stable bundles for small $\varepsilon$. Figure 5.21 shows maps for the close-cross-dataset for different values of $L_a$. The results are very similar, only the maps for smaller values seem to contain a few more noisy features than the maps for larger values.

(a) $L_a = 5$
(b) $L_a = 10$
(c) $L_a = 15$
(d) $L_a = 20$

Figure 5.21: Maps generated for the close-cross-dataset, with different values for $L_a$.

For $L_r$ small changes in the value have a larger impact. Also, the value cannot be too large, otherwise we may miss a lot of bundles for larger $\varepsilon$, as can be seen in Figure 5.22. The wide-bundle that is supposed to represent the road is no longer present, and only a few small bundles with outliers are left.

Figure 5.22: Map for the outliers-dataset, with $L_r = 3$
Next we consider the influence of $c_{\text{int}}$, which helps removing intersections and very short bundles. Figure 5.23 shows maps for the close-cross-dataset for different values of $c_{\text{int}}$. We can observe that for low values we get some noisy features near locations where paths appear to intersect. We also get vertices at these locations, where we do not have them for larger values. These vertices may not be desirable. Even though the paths appear to cross, there is no evidence in the data it is really an intersection, as none of the trajectories take a turn. So we may have a situation where paths cross by means of a bridge or a tunnel. Also, for smaller values we get a lot more vertices on straight paths as we have a lot more short bundles. For larger values, the result does not seem to change much, except that some shorter bundles tend to disappear.

(a) $c_{\text{int}} = 0$

(b) $c_{\text{int}} = 2$

(c) $c_{\text{int}} = 4$

Figure 5.23: Maps generated for the close-cross-dataset, with different values for $c_{\text{int}}$.

For many of the parameters we can see we do not have to get the value very precise to give an acceptable result. Hence we can use a similar set of parameter values for different situation.

5.1.4 Full Datasets

Figure 5.24 presents the bundles and the resulting map of the Aiguamolls-dataset. Both are drawn on top of map-data from OpenStreetMap for comparison. The total process took about 3 hours to complete. In Tables 5.4 to 5.6 different numerical results are summarized. We can see that most input trajectories are represented in the map.

<table>
<thead>
<tr>
<th>Segmentation</th>
<th>Simplification</th>
<th>Bundling</th>
<th>Map</th>
</tr>
</thead>
<tbody>
<tr>
<td>230</td>
<td>6</td>
<td>11430093</td>
<td>80</td>
</tr>
</tbody>
</table>

Table 5.4: Running times (in milliseconds) for different parts of the map construction for the Aiguamolls-dataset.

<table>
<thead>
<tr>
<th>Generated</th>
<th>Relevant</th>
</tr>
</thead>
<tbody>
<tr>
<td>8615</td>
<td>517</td>
</tr>
</tbody>
</table>

Table 5.5: Number of bundles for the Aiguamolls-dataset.

By zooming into certain parts of the map as in Figure 5.25 we can see that the generated edges come close to the data from OpenStreetMap. Also, we can detect some noteworthy
5. EXPERIMENTAL EVALUATION

Figure 5.24: Trajectories and selected bundles (top) to construct the map (bottom), for the Aiguamolls dataset, on top of map data from OpenStreetMap.

<table>
<thead>
<tr>
<th>Vertices</th>
<th>Edges</th>
<th>Support</th>
</tr>
</thead>
<tbody>
<tr>
<td>223</td>
<td>456</td>
<td>Min 1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Max 22</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Avg 6.534</td>
</tr>
<tr>
<td></td>
<td></td>
<td>St. Dev 3.876</td>
</tr>
</tbody>
</table>

Table 5.6: Map properties for the maps generated for the Aiguamolls-dataset. The support is the size of the bundle supporting an edge.

features. First, by only looking for large bundles, we successfully eliminate outliers, such that they do not distort the map, as can be seen in Figure 5.24. We have a single erroneous trajectory crossing a river, which is undesirable to have represented in a map. Since this (part of the) trajectory is in a size-one bundle, it is not selected to construct the map. Also
5. EXPERIMENTAL EVALUATION

note that in this particular situation we have two roads relatively close alongside each other (about 50 m). These appear separately in the map, as the bundles for the individual roads are considered much more relevant than the large bundle covering both.

Figure 5.25: Zoomed in parts of the map for the Aiguamolls dataset, on top of map data from OpenStreetMap

Figure 5.26: Outlier trajectory (blue) crossing a river (left) is not represented in the map (right)

Because the dataset contains trajectories along a beach, it is interesting to see how these trajectories are represented in the map. It depends on the application of the map what representation of the beach is most desirable, but it at least seems undesirable to have it represented by many individual edges. Note that the OpenStreetMap map-data does not represent this beach by a road. In our case, the trajectories on the beach are represented by a single path, as can be seen in Figure 5.27. This is because we have wide bundles covering the full width of this beach. This shows that by not having a fixed proximity parameter we can adapt to different situations.

Another observation we can make is that the dataset appears to contain quite a few “lonely” trajectories, where we have just one or two trajectories on a path. These are not represented because of the $k_{\text{min}}$ threshold. We can use the improvement suggested in Section 3.3.2 to try to represent them anyway. This means we include the bundles that merge at or after $\varepsilon_{\text{size}}$ regardless of size. We choose $\varepsilon_{\text{size}} = 200(= \varepsilon_{\text{max}})$, meaning that we add bundles that do not merge at all for the considered range of $\varepsilon$-values, or that merge at the very end. Doing this increases the number of selected bundles to 1596. Figure 5.28 presents the map generated.
5. EXPERIMENTAL EVALUATION

Figure 5.27: Trajectories on a beach (left) are represented by a single path in the map (right)

Figure 5.28: Map for the Aiguamolls dataset when attempting to include sparse data. The map is displayed on top of map data from OpenStreetMap.

We can see that now a lot more trajectories are represented in the map, without including too much noise.

For this to work properly, it is important that $\varepsilon_{\text{size}}$ is not too small. If, for example, we would have picked $\varepsilon_{\text{size}} = 100$, then the outlier of Figure 5.26 is represented in the map, as can be seen in Figure 5.29. This also means $\varepsilon_{\text{max}}$ should be sufficiently large as $\varepsilon_{\text{size}} \leq \varepsilon_{\text{max}}$.

Finally, we can detect a part of the map where two parallel roads have collapsed into one, as can be seen in Figure 5.30. We can again attempt separating the bundle by using the improvement from Section 3.3.1. With $\varepsilon_{\text{sep}} = 10$ and $\gamma = 0.75$, we can eliminate the large bundle. In this way the parallel roads properly show up in the generated map (see Figure 5.31), and the rest of the map is not disturbed.
Figure 5.29: Bundles representing outliers are included and end up as an edge in the map if $\varepsilon_{\text{size}}$ is too small.

Figure 5.30: A bundle covering two parallel roads (left) causes the road to collapse to a single edge (right).

Figure 5.31: The bundle covering both parallel roads can be removed (left), such that they are properly represented (right).
5. EXPERIMENTAL EVALUATION

5.2 Urban Road Networks

5.2.1 Setup

For the vehicular data we use data sets from [3]. We use two types of datasets: small ones and large ones. For the small type we use Athens-small and a subset of 140 trajectories of Chicago. Our current implementation is too slow to process much larger data sets under normal circumstances. However, we stress that our current implementation should be regarded as a prototype, whose goal is to evaluate the quality of the results obtained by the bundling approach. For both datasets we have a ground-truth map available.

For the large type we use a subset of 30 trajectories from the Athens-large-dataset. To be able to generate a result within a reasonable amount of time we use the modification suggested in Section 3.3.3: we increase $k$ each iteration exponentially using powers of 2. We do not have a ground-truth map for this dataset.

Properties for the datasets can be found in Table 5.7. The parameter values used for preprocessing, finding the bundles and determining relevant bundles are listed in Table 5.8. We use a larger $\Delta \varepsilon$ for the large dataset to reduce the number of iterations. Also, because the Athens-large-dataset covers a relatively large area and since we take only a small subset, we set $k_{\min}$ slightly lower than for the smaller datasets.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Num. Trajectories</th>
<th>Num. Vertices</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>original</td>
<td>preprocessed</td>
</tr>
<tr>
<td>Athens-small</td>
<td>129</td>
<td>142</td>
</tr>
<tr>
<td>Chicago</td>
<td>140</td>
<td>156</td>
</tr>
<tr>
<td>Athens-large</td>
<td>30</td>
<td>287</td>
</tr>
</tbody>
</table>

Table 5.7: Properties of the vehicular datasets

<table>
<thead>
<tr>
<th>Situation</th>
<th>Preprocessing</th>
<th>Finding bundles</th>
<th>Relevant bundles</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$\delta$</td>
<td>$\varepsilon_{\text{seg}}$</td>
<td>$c_{\lambda}$</td>
</tr>
<tr>
<td>Small</td>
<td>10</td>
<td>20</td>
<td>2</td>
</tr>
<tr>
<td>Large</td>
<td>10</td>
<td>20</td>
<td>2</td>
</tr>
</tbody>
</table>

Table 5.8: Parameter values used for the vehicular data. For the preprocessing, $\delta$ is the error used in Douglas-Peucker simplification and $\varepsilon_{\text{seg}}$ is the segmentation-distance. For bundling $c_{\lambda}$ is the scaling factor between $\lambda$ and $\varepsilon$, $\Delta \varepsilon$ is the step size when discretizing the range of $\varepsilon$-values, $\varepsilon_{\text{max}}$ is the upper bound of the range of $\varepsilon$-values considered. Parameters $C$, $H$, $D$, $L_a$, $L_r$, $c_{\text{int}}$, $k_{\min}$ are used to determine relevant bundles.
5. EXPERIMENTAL EVALUATION

5.2.2 Results

Here we present the results for constructing the maps of the vehicular data. We discuss some observations we can make for the datasets and highlight and discuss some noteworthy features that can be found in the maps.

Figure 5.32 presents the bundles and the resulting map of the Athens-small-dataset. The total process took about 4 hours to complete. In Figure 5.33, we can see the bundles and the resulting map for the Chicago-dataset. The total process for this dataset took about 4.5 hours to complete. Finally, in Figure 5.34, we can see the bundles and the resulting map for the Athens-large-dataset. The total process for this dataset took about 6 hours to complete. We can see that most paths in the travel network captured by the trajectories appear to be represented, and for the small datasets we can see that the created edges do not deviate too much from the ground truth.

![Figure 5.32: Trajectories and selected bundles (left) to construct the map (right), for the Athens-small dataset, on top of the ground truth](image)

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Segmentation</th>
<th>Simplification</th>
<th>Bundling</th>
<th>Map</th>
</tr>
</thead>
<tbody>
<tr>
<td>Athens-small</td>
<td>87</td>
<td>&lt; 1</td>
<td>14 805 037</td>
<td>62</td>
</tr>
<tr>
<td>Chicago</td>
<td>313</td>
<td>&lt; 1</td>
<td>16 684 883</td>
<td>268</td>
</tr>
<tr>
<td>Athens-large</td>
<td>4922</td>
<td>&lt; 1</td>
<td>21 011 878</td>
<td>1620</td>
</tr>
</tbody>
</table>

Table 5.9: Running times (in milliseconds) for different parts of the map construction for the vehicular data.

In Tables 5.9 to 5.11 different numerical results for the datasets are summarized. We can make
some observations that hold for all datasets. First of all, Table 5.9 shows that the running time of the algorithm is largely dominated by finding bundles. This is not surprising, as the preprocessing steps and the map construction only have to be performed once, whereas finding
5. EXPERIMENTAL EVALUATION

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Generated</th>
<th>Relevant</th>
</tr>
</thead>
<tbody>
<tr>
<td>Athens-small</td>
<td>22 235</td>
<td>289</td>
</tr>
<tr>
<td>Chicago</td>
<td>19 450</td>
<td>456</td>
</tr>
<tr>
<td>Athens-large</td>
<td>33 284</td>
<td>4893</td>
</tr>
</tbody>
</table>

Table 5.10: Number of bundles for the vehicular data

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Vertices</th>
<th>Edges</th>
<th>Support</th>
<th>Min</th>
<th>Max</th>
<th>Avg</th>
<th>St. Dev</th>
</tr>
</thead>
<tbody>
<tr>
<td>Athens-small</td>
<td>109</td>
<td>234</td>
<td>1</td>
<td>73</td>
<td>10.205</td>
<td>12.678</td>
<td></td>
</tr>
<tr>
<td>Chicago</td>
<td>70</td>
<td>180</td>
<td>1</td>
<td>65</td>
<td>21.967</td>
<td>18.902</td>
<td></td>
</tr>
<tr>
<td>Athens-large</td>
<td>1381</td>
<td>3198</td>
<td>1</td>
<td>57</td>
<td>4.866</td>
<td>5.998</td>
<td></td>
</tr>
</tbody>
</table>

Table 5.11: Map properties for maps generated for the vehicular data. The support is the size of the bundle supporting an edge.

the bundles goes through many iterations. The number of iterations is highly dependent on the choice of $\Delta \varepsilon$ and $\varepsilon_{\text{max}}$, as these influence the number of discrete $\varepsilon$ values we compute bundles.

If we further look into the finding of bundles, then for each $\varepsilon$-value the algorithm performs three steps: generating the bundles for different sizes, removing all subbundles, and linking bundles to each other to determine the evolution. Figure 5.35 shows for each $\varepsilon$ the running times for these three steps. Also here we can see that finding the bundles in each step dominates the running time. This is because besides having to iterate for each discrete $\varepsilon$-value, we also iterate over all possible bundle-sizes. We see that the running time for the bundling mostly increases with $\varepsilon$, mainly because a larger proximity parameter means we can find larger bundles leading to more iterations.

![Figure 5.35: Running times for different aspects of finding bundles, for each $\varepsilon$-step](image)

We can additionally see a spike in the running times for removing subbundles and determining
evolution at around $\epsilon = 25$. This is because we have a much larger number of bundles found for this proximity value than for others, as can be seen in Figure 5.36. This is likely related to the fact it is large enough to have many bundles for many roads, but the value is too small to have many bundles dominating other bundles yet.

If we look at Table 5.10, we can see that of all maximal bundles, only a small portion is considered relevant. For the Athens-small-dataset, only about 1% of the bundles are selected, for the Chicago-dataset only about 2.5%, and for the Athens-large-dataset only about 15%. Also if we again consider Figure 5.36, we can see that only a small portion of all bundles generated is actually maximal. So even with a large number of possibilities for bundles available, we are able to largely reduce the possible options to a relatively small set of bundles that allows us to create a reasonable map.

Finally consider Table 5.11. We can see that we have quite a bit of variation in the support of the edges. On one hand this is good, because it shows our approach is able to deal with different densities in the data. On the other hand, we also have edges with a support of just 1. These edges may not be very interesting and may correspond to noise. We are able to prune the map based on the support to remove these edges if desired.

Next we will focus on each dataset in more detail.

**Athens-small**

Recall the map generated in Figure 5.32. If we zoom into certain parts of the map we can detect some interesting properties. First, the algorithm is able to properly deal with different densities in trajectories and road widths. In Figure 5.37 we have a large road with 59 trajectories, of which six branch off to a smaller road. Both the roads are properly represented in the map.

Secondly, by having a proximity parameter for each bundle, the algorithm is able to represent intersections even if the trajectories are more spread apart. An example of this is in Figure 5.38. By having appropriate values for the proximity parameter we have bundles for the different directions, which allows us to have the intersection represented. We can also see in Figure 5.38 that on the right we have one intersection where we may expect two. This is
5. EXPERIMENTAL EVALUATION

Figure 5.37: Roads with different densities in the trajectories (left) are properly represented in the map (right) because we have two large bundles close to each other representing two roads, and we have a few trajectories traveling in between. This causes the two intersections to collapse into a single one. Also, we do not have a path from the left side to the right side of the area, as there is no evidence for this in the trajectory data.

Figure 5.38: Properly identified intersection (right) even if the trajectories are spread apart (left).

Chicago

Recall the map generated in Figure 5.33. If we zoom into certain parts of the map we can again detect some interesting properties. First, we note that the algorithm is able to deal with trajectories that still stay relatively close to the actual road, but clearly show noisy parts. Because of the bundle-specific proximity parameter, these noisy trajectories do not disturb the represented edge. This is depicted in Figure 5.39.

In addition, even in the presence of considerable noise, we are able to identify important edges in the map. As can be seen in Figure 5.40 even if we have many noisy trajectories which result in some noisy edges we can still identify an important edge (purple in the figure). The purple edge represents about 30 trajectories, whereas the other (blue) edges represent just two or three trajectories. Thus this can be used to tell apart the most reliable edges.
5. EXPERIMENTAL EVALUATION

Figure 5.39: Trajectories which behave a bit more noisy than others (left) do not disturb the represented edge.

Figure 5.40: For roads with a lot of noisy trajectories (left) we can still identify the important edge (middle, in purple), in between noisy edges (middle, in blue). After filtering out edges with small supporting bundles only the correct edge remains (right).

Athens-large

Recall the map generated in Figure 5.34. Also recall that to generate the bundles for the map we used powers of 2 when iterating over the bundle sizes. If we look at areas where the data is less dense, meaning we only have a few trajectories over a path/road, we can see that the algorithm is quite successful in creating a map in which the trajectories are represented well. Figures 5.41 and 5.42 show examples of these. This is also the case if there is only a single road in the area on which there are many trajectories, as in Figure 5.42. Since we use powers of 2, we check multiple small bundle sizes. So if a road is only covered by few trajectories, we are likely to find a bundle covering all trajectories, such that the road is properly represented in the map.

The situation changes when the data in an area is much denser and when we have multiple roads with many trajectories, as in Figure 5.43. We can see that for the larger roads, we get multiple edges instead of just one. This is because we check fewer large bundle sizes, so we may not find a bundle capturing all desired trajectories. We do, however, find smaller bundles, which may partially overlap each other. So the road is represented in the generated map, only by more edges than desired.
5. EXPERIMENTAL EVALUATION

Figure 5.41: Area where the data is less dense (left) and the corresponding part of the map (right)

Figure 5.42: Area where the data is less dense (left) and the corresponding part of the map (right). We also have a single road here containing many trajectories

Figure 5.43: Area where the data is dense (left) and the corresponding part of the map (right)
5. EXPERIMENTAL EVALUATION

5.3 Challenges

Even though our algorithm is able to deal successfully with many situations in which previous algorithms perform poorly, some situations remain challenging. One of the challenges can be seen in Figure 5.40 where a lot of noise in the trajectories leads to additional edges. These additional edges can be filtered out by looking at the support for the edge, or we can try to distinguish outliers from sparse data using the improvement of Section 3.3.2. Additionally, we observe many edges in a small region where many trajectories end but at quite different positions, see Figure 5.44. In this case we cannot find a bundle that bundles the ends of these trajectories together well for small proximity. Instead, we get many smaller bundles grouping trajectories together that end at more or less the same location. Since we have an edge for each bundle, we get many edges instead of a single one.

![Figure 5.40](image1)

![Figure 5.44](image2)

We separately keep track between which two locations a path should appear, and its representation. It is possible that after multiple bundles are processed, the endpoints of the representative no longer coincide with the desired start and end points. We handle this by connecting the ends of the representative to the desired end points with straight edges. This may lead to some artifacts, such as Y-shapes at T-crossings, or Z/N-shapes where two paths are connected, see Figure 5.45.

![Figure 5.45](image3)

Figure 5.44: Many trajectories ending on the same road at different positions, giving multiple small bundles instead of a large one (left). This leads to many edges for each bundle (right).

Figure 5.45: Artifacts when connecting edges together: Z/N-Shape (left) and Y-Shape (right).
5. EXPERIMENTAL EVALUATION

Since we use the part of an input trajectory as a way to represent edges or paths on the map geometrically, any error that appears on that trajectory can also be seen back in the resulting map. If we look at Figure 5.46 we can see the that the reconstructed graph has a small dent, where we expect a straight path, simply because the representative has this dent. A solution would be to choose a better representative. Currently, we use the “cluster center” (the trajectory used as reference by the bundling algorithm). This is sufficient for our proof-of-concept implementation, but it would be interesting to see if a more clever representative such as an “average” or a “median” trajectory could be used.

Figure 5.46: Error in the representative of a bundle is reflected back in the map

A final challenge we have already briefly touched upon is the situation of two close parallel roads, especially in the presence of noise. As mentioned before it is challenging to differentiate between trajectories over two parallel roads and a road that is wide. In Section 5.1 we already saw that we can in some cases solve it by using the improvement from Section 3.3.1 by checking if a bundle can be separated into two. This, for example worked well for the parallel-dataset and for part of the Aiguamolls-dataset. It, however, does not work for the parallel-wide-dataset, where we have noisy trajectories in between two paths.
6. Conclusions

In this thesis, we presented a map construction algorithm based on subtrajectory clustering. The main focus of our work was on automatically selecting relevant bundles from a set of input trajectories that are useful for constructing maps. As demonstrated in the experimental evaluation, this successfully allows us to address key challenges in map construction. Specifically, by working with subtrajectories rather than points, the algorithm easily handles irregular sampling rates and manages to separate roads that are close in proximity, but separate. By looking for stable clusters, we avoid common problems of map construction algorithms that use a global spatial proximity parameter.

Our algorithm does not explicitly consider outliers, but successfully identifies outliers, since these result in bundles of small size. This allows to easily prune them. However, this pruning currently requires an additional threshold, namely the minimum size of a relevant bundle, which may also prime bundles for roads where we have only little data available. We suggested the improvement to drop the size requirement if a bundle is maximal beyond a certain value for the proximity parameter, which can in some cases successfully differentiate between outliers and roads with little available data. This does, however, require an additional global parameter. The open problem remains as to how (or if) one could automatically choose the size threshold for each bundle individually, by taking the neighborhood of the bundle into account. For instance, small bundles that are fairly close to a large bundle may be considered less relevant than a small bundle with no other bundles nearby.

Our algorithm introduces a considerable number of parameters for the different part of the algorithm, which can all be tweaked individually. Our experimental evaluation demonstrated, however, that we can use almost the same parameter values in many different situations to create maps of similar quality. We have constructed maps for small samples of difficult situations, full hiking datasets, and small and large vehicular datasets, which all have different properties. The parameters that were changed for the different situations were mostly related to speeding up the algorithm to accommodate for large sizes of the dataset, rather than directly influencing the bundles that are considered relevant.

In our current description and implementation of the algorithm, we naively generate many candidate bundles, by exploring many different values of the bundle parameters, and then throw away many of them. For our prototype implementation this worked well enough, as we could evaluate the quality of the bundling approach. However, for larger maps, we clearly need a faster algorithm. Hence, one of the main topics for future work would be to design and implement an algorithm that directly computes the set of maximal stable large bundles that we are interested in. Since our prototype implementation suggests that the bundling approach produces fairly good maps, this may lead to a very promising solution for the map construction problem.

Our algorithm stitches together relevant bundles to obtain the map. For this, we use a relatively simple greedy strategy. The portion of the map corresponding to a bundle is represented by one of the subtrajectories in the bundle. In most cases, this works well, also because the algorithm for subtrajectory clustering naturally computes a representative for each bundle. However, the representative sometimes itself contains some noise. It would
be interesting to compare different algorithms that compute a representative of a cluster in the setting of map construction. In the stitching process, we do not explicitly handle road crossings, resulting in small dents at T-crossings. Crossings in principle result in bundles of subtrajectories of large size but small length. Currently, we do not make use of these bundles and leave this to future research.

Finally, we mainly evaluated our algorithm using visual inspection of the constructed maps, sometimes with the help of a ground truth map. Ahmed et al. propose various quality measures specifically to evaluate map construction algorithms. Another topic for future work would be to evaluate this approach, or an improved version, by means of these quality measures.
Bibliography


[7] Lukas Beckmann, Benedikt Budig, Thomas C van Dijk, and Johannes Schamel. There and back again: Using Fréchet-distance diagrams to find trajectory turning points.


BIBLIOGRAPHY


