Institute of Formal Methods in Computer Science

University of Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Masterarbeit Nr. 101

Incremental Weak Fréchet Map-Matching

Jonas Tangermann

Course of Study:	Softwaretechnik
Examiner:	Prof. Dr. Stefan Funke
Supervisor:	Dipl.-Inf. Martin Seybold
Commenced:	May 15, 2016
Completed:	November 14, 2016
CR-Classification:	E.1, F.2.2, G.2.2

Abstract

Many geographic information systems share the common task of mapping positional data given as trajectories onto a given map, the so-called map-matching. One category of approaches to solve this problem are geometric map-matching algorithms utilizing the Fréchet metric. To find the optimal match, these approaches either use parametric search and depth-first search or a modified version of Dijkstra's algorithm.

In this thesis a weak Fréchet based map-matching algorithm for undirected graphs is developed, which avoids the above-mentioned techniques with the objective of computation speed improvements. Furthermore, this algorithm is designed to support parallelization through its incremental nature.

The proposed approach is evaluated by comparing theoretical complexities as well as experimental evaluation against the Dijkstra basted approach of Wenk et al.[WSP06]. The evaluation, however, only yielded indications for positive results under heavy use of parallelization.

Zusammenfassung

Viele geografische Informationssysteme haben die Aufgabe Positionsdaten in Form von Verläufen auf eine gegebene Karte abzubilden, das sogenannte Map-Matching. Eine Kategorie von Ansätzen zur Lösung dieser Problemstellung sind geometrische Algorithmen, die sich der Fréchet Metrik bedienen. Diese Verfahren nutzen entweder eine Kombination von Parameter- und Tiefensuche oder eine modifizierte Variante des Dijkstra-Algorithmus, um den besten Match zu finden.

In dieser Arbeit wird ein Algorithmus auf Basis der Weak Fréchet Metrik entwickelt, welcher diese Techniken umgeht, um eine Beschleunigung der Match-Berechnung zu erreichen. Der vorgestellten Algorithmus ermöglicht eine einfache Parallelisierung aufgrund seines inkrementellen Vorgehens.

Die Evaluierung des vorgestellten Algorithmus wurde auf Basis der theoretischen Laufzeiten sowie einer experimentellen Gegenüberstellung des Ansatzes von Wenk et al.[WSP06] vollzogen, zeigte jedoch nur bei starker Parallelisierung positive Ergebnisse.

Contents

1 Introduction

This chapter aims to give an overview of the subject and the context of this thesis. In section 1.1 the objective of this thesis is explained and supported by scientific and personal motivations. Eventually, the overall structure of this paper will be described in section 1.2.

As devices and applications capturing positional data like smart phones, car navigation systems or GPS trackers for sports and health applications are widespread nowadays, the usage of positional data has become important in order to maintain a high quality standard for services and applications like routing applications, fleet management systems or traffic estimation and control [GDM+12; QON07]. Many of them use some kind of map and GPS as their source of positional data. However, the combination of positional and map data is not as straight forward as it may seem. All positional data that was created by some kind of measurement, for example through GPS, contains some deviation. In addition, maps also contain deviation. Nevertheless, for most applications it is crucial to combine these to datasets correctly. For example, imagine a map with two parallel streets and a sequence of positional measurements of a car that are located somewhere between these streets. For a navigation-system it is important to know the correct street the car is driving on in order to generate the correct driving instructions.

The task of projecting positional measurements onto a given map is called map-matching problem. The goal of map-matching algorithms is, given a sequence of positional measurements, to compute a path within the given map such that it is as close as possible to the unknown real path. This task can be tackled in a variety of ways, however, this thesis focuses on geometric approaches using the Fréchet metric. In contrast to other approaches, which embed the usage of other information, such as the type of movement or velocity, the algorithms of interest only consider geometric features regarding to the combination of map and positional measurements.

The Fréchet metric can be used to quantify the distance between two curves. In this thesis the weak Fréchet metric will be used to measure how well a path in the map fits the given positional measurement sequence.

1.1 Motivation

The goal of this thesis is to develop a weak Fréchet distance based algorithm, which solves the map-matching problem in an incremental fashion, such that it undercuts the running times of known approaches. The incremental property of the algorithm should be established by a segment after segment construction of the match output as well as through parallel computation of the main data structure.

As mentioned above, map-matching algorithms are employed in a variety of use cases. For applications that rely on geometrical matching approaches a speed improvement would be desirable. Furthermore, the computation hardware has developed towards more parallelization, hence, a parallel approach could be beneficial. However, the subject of this thesis is not only interesting due to its applications, but also because of its questions on weak Fréchet computation.

1.2 Structure of Work

First of all, the preliminaries are explained in chapter 2. Section 2.1 introduces the definition of the map-matching problem, which is to be solved by the proposed algorithm. The Fréchet distance is the core metric used by the developed algorithm and is defined in section 2.2. Mathematical models for the representation of the earth's surface as well as coordinate systems within these models are given in section 2.3. The spherical model is discussed in more detail, as it is the model of choice for this thesis. Additionally, some important calculations within this model, such as distance calculation, are explained. All important data structures that are used by the proposed algorithm are explained and discussed in section 2.4. The preliminaries chapter ends with a short introduction to the OpenStreetMap project and map data, which is later used in the evaluation process.

In chapter 3 related work is mentioned starting with the explanation of some map matching approaches using completely different techniques compared to those used by the approach suggested/used in this thesis. For example, the interactive voting based approach by Yuan et al. [YZZ+10]. The second part of the chapter is dedicated to the known approaches that are conceptionally close to the algorithm proposed in this paper. The usage of the Fréchet distance is one main common factor of these approaches.

Chapter 4 finally describes the algorithm developed in this thesis. This chapter is structured along the algorithm's main stages. These stages are map extraction 4.1, free space graph computation 4.2, and match search 4.3. The map extraction section explains why one could not use the whole map graph within the later stages and how to extract the needed subgraph. The free space graph computation section describes how the main structure of the algorithm is created by combining the map subgraph and a given position trace. The final step that extracts the algorithm's result out of the previously created structure is explained in the match search section. An overview on the algorithm's complexity is given in section 4.4.

The evaluation procedure and its results are described in chapter 5. The algorithm is examined according to performance and match quality. Later on, an experimental comparison is done in order to classify the quality and performance. This thesis ends with a conclusion and possible future work on this topic.

2 Preliminaries

The preliminaries chapter introduces subjects and definitions, which are imported for the understanding of the algorithm developed in this thesis. Furthermore alternative models and data structures that could have been used are described and discussed. First of section 2.1 explains and defines the map-matching problem, followed by the definition and computation strategies of the Fréchet distance. The mathematical base model for all geometrical operations needed is introduced in section 2.3. Some of the most important data structures used in the proposed algorithms are explained in section 2.4. Finally the OpenStreetMap project is named in section 2.5, it is later used for evaluation purposes.

2.1 Map-Matching

Map-matching or map adaptation describes the projection of movement data into a given way network. This projection is considered to be non-trivial due to errors in the movement data.

Movement data is mostly defined as a sequence of position measurements. A position measurement itself is defined as a tuple (la, lo) consisting of the components latitude and longitude. The tracked object's movement is then approximated by a polygonal curve, which is created by using linear interpolation on those point measurements. These curves are often called trajectories. Each measurement can be augmented with additional information such as the current time or error radius [EFH+11].

The error radius can be adjusted according to the used measurement technology. The most commonly used position measurement technology is the Global Positioning Systems (GPS). Such measurements are distorted by two types of errors. The first error source is the measurement technology itself. Every technology comes with a certain accuracy and, therefore, a deviation in its measurements. The deviation of GPS positions can be described as a bivariate normal distribution in three dimensional space. This distribution is assumed to be Gaussian, see figure 2.1. Within this model the accuracy of GPS measurements can be described in terms of standard

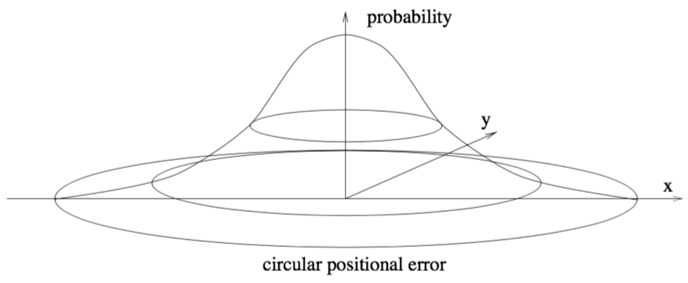

Figure 2.1: Measurement derivation of GPS [PJ99]

deviation.[LRT15] Another reason for inaccurate measurements has its origin in the used sampling rate, which is the time interval between consecutive measurements. In [PJ99] Pfoser and Jensen showed that the sum of measurement and sampling error can be seen as a lenticular probability distribution.

The way network is formally defined as graph $G_w = (V, E)$ with $V \subseteq \{(la, lo) | la \text{ latitude}, lo \text{ longitude}\}$ and $E \subseteq \binom{V}{2}$. Ways of any kind are represented by paths $v_1, v_2, \ldots, v_n$ with $\{v_i, v_{i+1}\} \in E$ were $0 \leq i < n$. The application domain may require the use of a directed graph, for example if a system considers one-way streets.

Definition 2.1 (Map-Matching Problem). *Let $G_w = (V, E)$ the way graph and $p = p_1, p_2, \ldots, p_n$ a sequence of position measurements be given. Furthermore, let w_r be the actually traversed way while recording p in G_w. The map-matching problem can then be stated as finding a way w_m in G_w such that w_m is as close to w_r as possible.*

An abstract map-matching example can be seen in figure 2.2. The picture is composed out of three differing lines or segments. The first type defines the way-graph or road-map and is illustrated by segments with dotted segment ends. Secondly, the trajectory to be matched is coloured red, has square ends, and is named p. The actual path that was travelled while recording this trajectory is indicated by dotted segments, named w_r. A possible match w_m produced by a map-matching algorithm might be the track shown as green line segments. In this example the map-matching algorithm is quiet close to the correct path but fails to find the proper end-segment. A categorization and description of such algorithms is given in chapter 3.

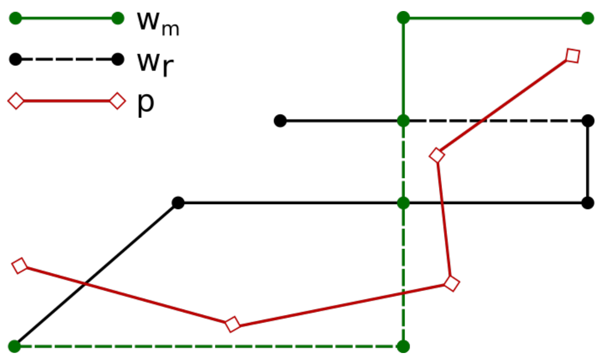

Figure 2.2: Example match

2.2 Fréchet Distance

This section introduces the Fréchet and weak Fréchet distance for two polygonal curves, which was originally invented by Fréchet in 1906 [Fré06]. Furthermore, the calculation of these distance metrics will be discussed. In this context the definition of free space and the free space diagram of two curves will be given [AG95].

The most common illustration of the Fréchet distance borrows the picture of a dog owner walking his dog. The dog owner as well as the dog want to follow an individual path while doing there walk, but these two paths could be restricted by the used leash. The Fréchet distance can now be described as the length of the shortest leash, so that both individuals are able to walk along their paths without ever going backwards. By interpreting the paths as curves the Fréchet distance is the length of that leash.

Definition 2.2 (Fréchet distance)**.** *Given two polygonal curves $f : [0, n] \to \mathbb{R}^2$ and $g : [0, n'] \to \mathbb{R}^2$ with $n, n' \in \mathbb{N}$ number of segments as well as two continues monotone increasing parametrisations α and β with $\alpha(0) = \beta(0) = 0 \wedge \alpha(1) = n \wedge \beta(1) = n'$ then the Fréchet distance can be defined as follows:*

$$\delta_F(f, g) := \inf_{\substack{\alpha:[0,1]\to[0,n] \\ \beta:[0,1]\to[0,n']}} \max_{t\in[0,1]} ||f(\alpha(t)) - g(\beta(t))|| \tag{2.1}$$

To define the weak Fréchet distance $\tilde{\delta}_F(f, g)$ the monotonic property of the parametrisations is omitted. For the intuitive description this would result in both the dog owner and the dog to be able to walk backwards on their paths.

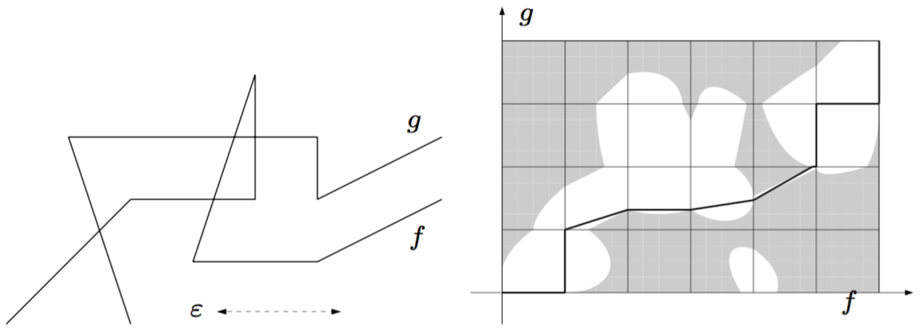

Figure 2.3: Two polygonal curves anyd their free space regarding ε. [BPSW05]

The calculation of the Fréchet distance can be achieved by using the free space concept. The free space of two curves is defined as the set of all pairs of points belonging to these curves such that their distance is less or equal to some given ε. The whole free space F_ε can be viewed as the union of all segment-segment free spaces.

Definition 2.3 (Free space). *Let two polygonal curves f, g with $dom(f) = [0, n]$ and $dom(g) = [0, n']$ for $n, n' \in \mathbb{N}$ as well as a distance function d be given. Then the free spaces for some fixed ε is given by:*

$$\begin{aligned} F_\varepsilon &:= \{(x, y) \in [0, n] \times [0, n'] | d(f(x), g(y)) \leq \varepsilon\} \\ &= \bigcup_{\substack{0 \leq i < n \\ 0 \leq j < n'}} \{(x, y) \in [i, i+1] \times [j, j+1] | d(f(x), g(y)) \leq \varepsilon\} \end{aligned} \tag{2.2}$$

The following results achieved by Alt and Godau lead to their decision algorithm for $\delta_F(f, g) \leq \varepsilon$. Firstly, for two segments of polygonal curves f', g' it holds that $\delta_F(f', g')$ is convex. Secondly, for two polygonal curves f, g the following holds: $\delta_F(f, g) \leq \varepsilon$ is true if and only if there is a monotone curve from $(0, 0)$ to (n, n') in F_ε.

By calculating all segment transitions, it is possible to detect whether a monotone curve, as mentioned in the second result, exists in F_ε. If there are monotone increasing transitions for cells from $(0, 0)$ to (n, n') then the convex property of each segment-segment cell ensures that there is a monotone curve between these consecutive transitions. When these curves are composed, they form a monotone curve from $(0, 0)$ to (n, n'), which, hence, decides whether $\delta_F(f, g) \leq \varepsilon$ by directly inducing two parametrisations α and β. An example can be seen in figure 2.3. On the left side two polygonal curves are shown in combination with some value ε, the corresponding free space is given as free space diagram on the right side. The calculation of $\delta_F(f, g)$ can now be solved by using parameter search on the decision problem, which is possible because F_ε contains another $F_{\varepsilon'}$ if $\varepsilon' < \varepsilon$. The search space of critical ε values that must be included in the parameter search is of size $\mathcal{O}(n^2n' + nn'^2)$ and contains three types of ε values. Type one are the minimal ε values which add $(0,0)$ or (n,n') or both to the free space, type two are minimal ε values that open the free space between two adjacent free space cells, and type three are minimal ε values such that a new horizontal or vertical passage opens in the free space. This approach results in an asymptotic running time of $\mathcal{O}((n^2n' + nn'^2)\log(nn'))$, which is dominated by the need to sort the critical values. By processing critical ε of type tree separately and applying techniques presented by Cole one could achieve $\mathcal{O}(nn' \log nn')$ running time [Col87].[AG95]

By omitting the constraint of monotonicity, meaning the only requirement is connectivity between $(0,0)$ to (n,n') within the free space, this approach can be adapted to the weak Fréchet distance. Every test within the parametric search uses a depth-first search run to test for connectivity in the free space defined by the current ε. The critical ε values relevant to the weak Fréchet distance are those of type one and two. Type three is not relevant as such passages are already open in terms of weak Fréchet distance due to being non-monotonic. Hence, the amount of critical ε values reduces to $\mathcal{O}(nn')$. The running time for every depth-first search run is $\mathcal{O}(nn')$, thus calculating $\tilde{\delta}_F(f,g)$ takes $\mathcal{O}(nn' \log nn')$ time.[AG95; BPSW05]

2.3 Spherical Geometry and Coordinate Systems

This section will introduce geographic coordinate systems such as the latitude-longitude system and the cartesian ECEF (Earth-Centered Earth-Fixed) system. In addition the calculation of surface distances in spherical geometry will be explained. These calculations will later be used for various operations within the proposed algorithm.

A variety of mathematical models can be employed to represent the earth's shape. For instance, one can define a model in which the earth's surface is just a plane. Due to the object's size, those models can be accurate enough for some applications. There is the possibility of advancing the approximation by using a spherical model. However, the shape of the earth rather fits those of an ellipsoid, so that a carefully defined ellipsoid is currently the most accurate way to define a geographical model. As an example, the World Geodetic System (WGS) 84 defines such an ellipsoid. The algorithms proposed in this paper, however, involve relatively small fractions of the earth's surface, in the range of a few kilometers. Hence a trade-off between accuracy

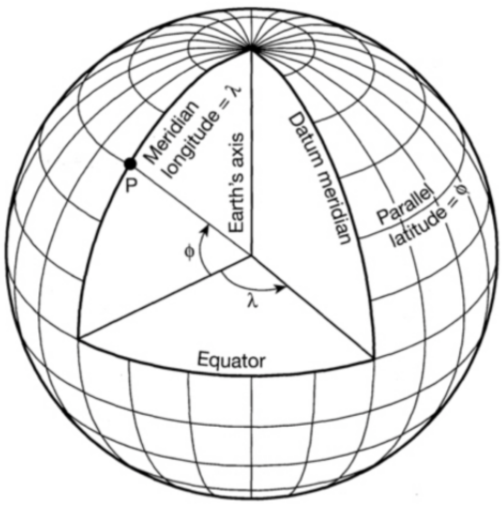

Figure 2.4: Latitude-Longitude Coordinate System [Jon14]

and complexity is picked by using the spherical model for all geographical calculations within this paper.[Jon14]

Probably one of the most known systems to represent geographical positions within a spherical model is defined around the values of latitude, longitude, and altitude. A pair of latitude and longitude values determine the position on a sphere, whereas the altitude defines the height by means of a reference value. For the usage in this paper the altitude component will be neglected as we assume a spherical shape, as justified above. Figure 2.4 illustrates this system as well as the ECEF system.

Definition 2.4 (Latitude-Longitude System). *In the latitude-longitude system every coordinate c is defined as a tuple (ϕ, λ) of angles between the coordinate and the corresponding reference plane. Subsequently, the latitude value ϕ is the angle between the coordinate and a reference plane through the equator which is orthogonal to the earth's rotation axis. Its domain is $\phi \in [-90°, 90°]$, with the equator defined as $0°$ and increasing value towards the north. Whereas, the latitude angle λ is constructed using the meridian which passes through Greenwich, London. Its domain is $\lambda \in [-180°, 180°]$, with the Greenwich meridian defined as $0°$ and increasing value towards the east.*

Another seemingly natural representation of positions in three dimensional space is given by any three dimensional Cartesian system. One of these systems is the Earth-Centered Earth-Fixed system, with which the calculation of position measurements is convenient [KH05, p. 28]. It is presumably the most used coordinate reference system in mathematics and geographic information systems, as its vectorial nature allows further calculations based on these coordinates [Gal06, p. 142].

Definition 2.5 (Earth-Centered Earth-Fixed). *The Earth-Centered Earth-Fixed system is defined as a three dimensional Cartesian coordinate system composed of the axis x, y, and z. The center $(0, 0, 0)$ is given by the earth's center of mass, thus Earth-Centered. The $x - axis$ points in the direction of $0°$ longitude, whereas the $y - axis$ is directed towards $90°$ longitude. Finally, the $z - axis$ is normal to the equatorial plane and therefore points to the geographical north pole. As a result the coordinate system is bound to the earth's rotation, thus it is called Earth-Fixed. Now every coordinate c can be defined as a triple $(x, y, z) \in \mathbf{R}^3$ denoted as a certain distance unit. [Gal06; KH05]*

Position measurements are mostly given in terms of the latitude and longitude. Whereas, the proposed algorithms calculate within the Cartesian system. Hence, there is a need to convert forth and back between those systems.

Calculation 2.1 (Coordinate System Conversion [TM12; Wei16]). *Coordinates can be converted between the Latitude-Longitude and Earth-Centered Earth-Fixed systems. The transformation from latitude and longitude (ϕ, λ) into the Cartesian model (x, y, z) is given by the following equation:*

$$c = \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} \cos\lambda\cos\phi \\ \sin\lambda\cos\phi \\ \sin\phi \end{bmatrix} \tag{2.3}$$

By using the inverse functions one could obtain the reverse transformation:

$$(\phi, \lambda) = (\arcsin z, \arctan(y, x)) \tag{2.4}$$

One of the basic operations within almost every geographic information system is the distance calculation between two given coordinates. If one takes the spherical model as a basis, the shortest line between any two points is an arc of a so-called great circle. Great circles are all circles on the sphere's surface that have the same radius as the underlying sphere. The shortest distance can, therefore, be calculated by finding the great circle arc between demanded coordinates, calculating it's angle, and multiplying it by the sphere's radius.

Calculation 2.2 (Great Circle Distance [Wei16]). *The great circle distance d of two given coordinates $\vec{c_1} = (x_1, y_1, z_1)$ and $\vec{c_2} = (x_2, y_2, z_2)$ can be calculated by using the dot product and the sphere's radius r. Recap that the dot product in three dimensional space can be defined as $\vec{c_1} \cdot \vec{c_2} = |\vec{c_1}||\vec{c_2}| \cos \sphericalangle(\vec{c_1}, \vec{c_2})$. Subsequently, the great circle distance is calculated as follows:*

$$d = r \cdot \arccos\left(\frac{\vec{c_1} \cdot \vec{c_2}}{|\vec{c_1}||\vec{c_2}|}\right) \tag{2.5}$$

Another geometrical problem that will be useful later on is the calculation of great circle intersections. This loosely corresponds to finding intersections of lines in planar space but with the difference that for two great circles there are always two intersections. Unless, they are identical. A great circle is defined by a plane that cuts through the sphere's center, this can be expressed by a normal vector $\vec{n}$ of that defining plane. Every Cartesian coordinate $\vec{c}$ that satisfies $\vec{n} \cdot \vec{c} = 0$ lies on that great circle when assuming that all vectors are normalized.

Calculation 2.3 (Great Circle Intersection)**.** *Given two pairs of coordinates* $(\vec{a_1}, \vec{a_2})$ *and* $(\vec{b_1}, \vec{b_2})$*. The intersection points of great circles implied by those coordinate pairs can be calculated using the representation of each circle as an equation.*

$$\begin{aligned} \vec{n_a} \cdot \vec{x} = 0, \textit{ with } \vec{n_a} = \vec{a_1} \times \vec{a_2} \\ \vec{n_b} \cdot \vec{x} = 0, \textit{ with } \vec{n_b} = \vec{b_1} \times \vec{b_2} \end{aligned} \tag{2.6}$$

Those normal vectors $\vec{n_a}$ *and* $\vec{n_b}$ *itself span a plane orthogonal to the searched intersection points. Hence, the calculation of intersection points* i, i' *can be completed as follows:*

$$\begin{aligned} \vec{i} = \vec{n_a} \times \vec{n_b} \\ \vec{i'} = \vec{n_b} \times \vec{n_a} \end{aligned} \tag{2.7}$$

2.4 Data Structures

There are three types data structures that are essential to the proposed algorithms. These data structures are disjoint-sets, heaps, and structures for graph representation. This section will provide definitions as well as complexity information on each of those data structures. In addition, some applications will be mentioned, particularly these which are closely relate to their use within the proposed algorithm, as described in section 4.

2.4.1 Disjoint-Sets

Disjoint-set data structures or union-find data structures are used to operate on disjoint subsets of a given set of elements. Three operations are commonly provided to manipulate these subsets. The make-set-operation takes a new element and creates a subset, a singleton. Whenever the union-operation is invoked, the given subsets are thrown together and handled as one for the rest of the execution. In order to determine if two given elements are in the same subset one could use the find-operation. These structures are often used within greedy algorithms. For example, Kruskal's algorithm for minimal spanning trees uses a disjoint-set data structure to detect edges in a graph that would close a cycle. [DMS14; OW11]

Definition 2.6 (Disjoint-Set Data Structure ([DMS14; OW11]))**.** *Let B be a set of elements and $P = \{B_1, \ldots, B_k\}$ a partition of B. Therefore $B_i \cap B_j = \emptyset$ for $1 \leq i < j \leq k$ and $B_1 \cup \ldots \cup B_k = B$. Hence, P is the quotient set of a equivalence relation $\sim_P$ on B. Substantially, an element $b_r \in B_l \in P$ is called representative of B_l, $[b_r] = \{b \in B | b_r \sim_P b\}$. A disjoint-set data structure is defined by providing the following operations:*

$makeSet(b)$:
: *Adding a new element to the structure, $P_{new} = P_{old} \cup \{\{b\}\}$. In terms of the equivalence relation $\sim_p$ is extended by (b, b).*

$find(b)$:
: *Retrieves a representative b_r of the class $[b]$ that is unique within the data structure. So that for every pair $b_1, b_2 \in [b]$ the find-operation satisfies the condition $find(b_1) = find(b_2) = b_r$.*

$union(b, b')$:
: *Union the two subset $[b]$ and $[b']$ within P, therefore $\sim_P$ is extended by $[b] \times [b'] \cup [b'] \times [b]$.*

Most of the time disjoint-set data structures are implemented as a forest of trees which represent the subsets. Adding an element is done by introducing a new tree of size one to the forest. A unique representative is defined by the root vertex of every tree. The union of two subsets, and therefore two trees, is achieved by attaching one vertex to the other as parent.

A simple implementation of such a forest is given by storing the information in two arrays. The first one $i[]$ holds all elements and is used to index these elements. The second array $p[]$ holds the index of the parent vertex within the tree. Adding an element b is now done by appending it to the first array $i[x] = b$ and setting the reflexive parent link $p[x] = b$. For every element its unique representative can be found by traversing along the parent link until $p[i[b]] = b$. Union of two trees is accomplished by resetting the parent link of one vertex such that it points to the other vertex. It can easily be seen that the makeSet-operation and union-operation are in $\mathcal{O}(1)$ and that the find-operation has $\mathcal{O}(n)$ time. The whole structure needs $\mathcal{O}(n)$ space as there are two arrays of size n.[OW11]

There are two known modifications which result in much better performance regarding the time needed to find a representative. One improvement is to union the trees according to their size and height, called union by rank. By utilizing this method the trees are restricted in height by $\mathcal{O}(\log n)$. Hence, the time complexity of the find-operation also shrinks to $\mathcal{O}(\log n)$. The second improvement is called path-compression. A new functionality is added to the find-operation in order to further shorten the paths to the root vertices. Every vertex along the path from starting vertex to representative vertex gets its parent link reset such that it points to the representative vertex. The amortized time complexity of m find- and $n-1$ union-operations is given by $\mathcal{O}(m\alpha(m,n))$ where α is the extremely slow growing inverse Ackermann function. For all practical uses one could assume $\alpha(m,n) < 5$ and, therefore, assume that every single operation takes practically constant time.[DMS14; OW11; Tar83]

The union-find data structure will later be used to determine, whether there is a path between two vertices in a given graph.

2.4.2 Priority Queue

Quite often algorithms need to process data based on associated priority values. One data structure to deal with such requirements is the priority queue. It offers the insert-operation to insert a new element, the getMax-operation to retrieve the element of highest priority, and removeMax-operation to retrieve and remove the element of highest priority. This section will give a definition for priority queues, state one implementation approach, and sketch some application cases.

Definition 2.7 (Priority Queue ([Knu98; OW11])). *Let $(D, \leq)$ be a totally ordered set and let $A \subseteq D$ be the set of elements handled by the priority queue. The operations of a priority queue are then defined as follows:*

$insert(d)$*:*
: *Adds the element $d \in D$ to the set of elements handled by the queue, $A_{old} = A_{new} \cup \{d\}$.*

$getMax()$*:*
: *Returns an element a_{min} such that for every $a \in A$ the condition $a \leq a_{max}$ holds.*

$removeMax()$*:*
: *Returns an element a_{min} such that for every $a \in A$ the condition $a \leq a_{max}$ holds and a_{min} is removed from A.*

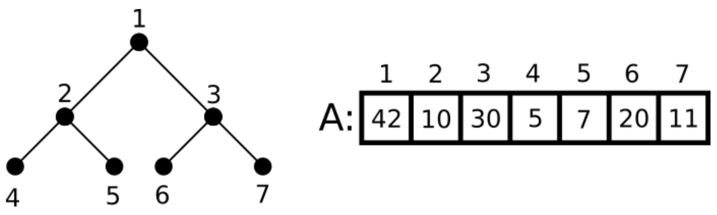

Figure 2.5: Heap example as binary tree and array

A common implementation uses a heap to maintain the data. A heap can be viewed as a nearly complete binary tree $T = (V, E)$. Where the operation $right(v)$ returns the right child and $left(v)$ returns the left child of v. By invoking $parent(v)$ one could retrieve the parent vertex. In order to be a heap T has to satisfy the heap condition:

$$\forall v \in V : v \leq parent(v)$$

This tree is mostly stored as a dynamic or fixed array A, in which the tree vertices are stored top to bottom and left to right. Every vertex is, therefore, mapped to an index within the array. The array itself stores the priority. Hence, $right(v) = 2v + 1$, $left(v) = 2v$, and $parent(v) = \lfloor \frac{v}{2} \rfloor$. See figure 2.5 for an illustration. Given a correct heap the insert operation of an element v is implemented by appending the new element to A and then switching it with its $parent(v)$ vertex as long as $A[perent(v)] < A[v]$. Due to the binary tree structure the traversed path has maximal length of $\log n$ and the insert-operation a time complexity of $\mathcal{O}(\log n)$. The maximal element of the heap is always the root vertex and, therefore, the first element in A. The first element of A is exchanged with the last element of A, then $A[A.length]$ is deleted. However, now the heap condition is violated at the root vertex. This is fixed by invoking the often called heapify-operation. It takes the array A and an index v as arguments and assumes that the subtrees rooted at $left(v)$ and $right(v)$ are valid heaps. It compares the priority $A[v]$ to $A[left[v]]$ and $A[right[v]]$. Then v is swapped with the vertex of highest priority. This is done recursively until both $A[left[v]]$ and $A[right[v]]$ are lower than $A[v]$. A path traversed by v is again bounded by $\log n$. Hence, $removeMax()$ can be executed in $\mathcal{O}(\log n)$ time, the included retrieval of the maximal element, however, only took $\mathcal{O}(1)$ time due to its root position.[Cor09]

The insert performance can be improved by implementing the heap as Fibonacci heap, see [OW11]. All of the definitions and implementations would also work for minimal priorities by exchanging the order relation. Therefore, heaps are distinguished as min- or max-heap. One prominent use of heaps can be found in Heapsort, see [Cor09]. In this paper, it will be crucial that one could retrieve minimal elements until a certain point and that it is not necessary to sort the whole set.

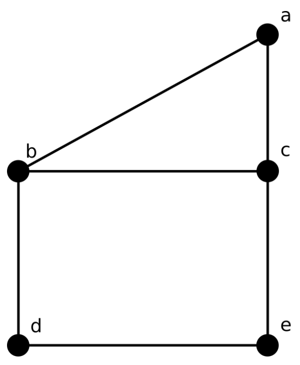

Figure 2.6: Example graph

2.4.3 Graph Data Structures

Graphs are a natural representation for map data and also for the free space of section 2.2. There are many ways of storing such graphs which differ in space complexity, support for certain graph types, and operations. In this section three of these representations will be described and discussed in regard of map and free space graphs. For each representation the graph of figure 2.6 will be used as an example.

One of the simplest ways to represent a graph is by storing an unordered list of edges. This approach needs $\mathcal{O}(|E|)$ space. However, it is not trivial to retrieve adjacent vertices, therefore, this representation is mostly used to store graphs but not used as the algorithms work piece. [DMS14] The example graph shown in figure 2.6 could be represented as follows:

$$G = (\{a, b\}, \{c, b\}, \{d, b\}, \{d, e\}, \{e, c\}, \{a, c\}).$$

An ordered variant using a heap implementation is used in the proposed algorithm to store the free space graph.

The second graph data structure is called adjacency matrix. For a graph $G = (V, E)$ every edge $(i, j) \in E$ becomes an entry $a_{i,j} = true$ of a boolean $|V| \times |V|$ matrix. The space needed to store this representation is, therefore, $\mathcal{O}(n^2)$. This structure is only appropriate for highly connected directed graphs. A undirected graph would result in a symmetric matrix, wasting a lot of space. However, a sparsely connected directed graph would also be wasting space as there are only a few entries switched to true in that case. Moreover, operations such as retrieving outgoing or ingoing edges of a given vertex need $\mathcal{O}(|V|)$ time. Due to the matrix initialisation most algorithms will take $\mathcal{O}(|V|^2)$ time [OW11]. With respect to map or free space graphs, this properties would result in bad performance, as these graphs are undirected and share the same

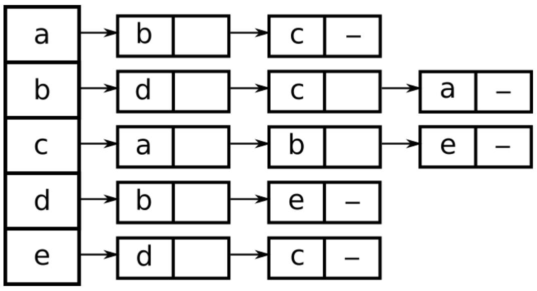

Figure 2.7: Adjacency list for the example graph shown in figure 2.6

low degree of incident edges. The adjacency matrix for the running example would look like this:

$$G = \begin{array}{c} \\ a \\ b \\ c \\ d \\ e \end{array} \begin{array}{c} \begin{array}{ccccc} a & b & c & d & e \end{array} \\ \begin{pmatrix} 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 \end{pmatrix} \end{array}.$$

A better way to store undirected graphs is the adjacency list. This data structure holds all vertices as field or list entries, which are pointing to a list of adjacent vertices. This representation needs $\theta(|V| + |E|)$ space and it is possible to implement efficient retrieval operations. Hence, an implementation of this structure is later used to store the map graph within the proposed algorithm. Figure 2.7 illustrates an adjacency list for an example graph.[OW11]

2.5 OpenStreetMap

The OpenStreetMap (OSM) project is a big community of volunteers aiming to create a free map of the whole world. An excerpt of the map is shown in figure 2.8. The community is backed up by the OpenStreetMap Foundation, which mainly performs fund-raising.[Fou16a]

However, the map data is not owned by the foundation, but available under Open Data Commons Open Database Licence (ODbL). Hence, it is necessary to use the same licence if one changes the data. In addition the cartography included in the map tiles and the documentation are licensed under the Creative Commons (CC).

Figure 2.8: OSM map excerpt of Hannover

If the creator and/or the distributor uses OSM-data, he/she has to indicate the usage of OSM-data by adding "© OpenStreetMap-Contributors" in a prominent place [Fou16b], which is hereby assigned for this thesis.

The map data can be downloaded in two formats, namely in xml- and pbf-format. The pbf-format is a compressed format that is used by the implementation described in this paper to initialise the algorithms map data. The download source used in this work is provided by the company Geofabrik GmbH, a member of the OSM-Foundation [Kar16].

3 Related Work

A variety of way more than 30 map-matching algorithms and approaches have been proposed by the research community till now [QON07]. This chapter will inform about simple categorizations of these approaches and will describe briefly some algorithms in section 3.1. Whereas, two closely related approaches will be presented in more detail in section 3.2.

One possible categorization was described by Mohammend A. Quddus et al. [QON07], defining the categories of geometric, topological, probabilistic, and advanced map-matching approaches. The geometric-category includes all approaches which use the geometric properties of the trajectory and the road network such as the coordinates of positional measurements and vertices of the road network. A very simple approach is to match each vertex of the trajectory with the closest vertex of the road network by means of the chosen distance metric. This is also called point-to-point matching. Another geometric method is point-to-curve matching, in which every trajectory vertex is compared to a curve of the road network. Taking it one step further by comparing the whole trajectory curve to any curve within the road network, we get curve-to-curve matching. As described later on, the proposed algorithm is a member of this category. Topology is meant to name properties of the road network such as adjacency, connectedness, or curvature. Approaches of this category define closeness by similarity of those properties. Furthermore, Quddus et al. defined the category of probabilistic algorithms, these algorithms consider the error distribution of measurement system, sample rate, and context. Eventually the category of advanced approaches contains algorithms that may combine techniques of the other categories and use more complex methods like fussy logic.[QON07]

All map-matching algorithms can also be differentiated into the two other categories. Incremental algorithms expand their results based on already matched parts of the trajectory. Incremental approaches are, therefore, *greedy* algorithms. The second category contains algorithms that calculate the matches in a global manner and, therefore, are called global algorithms. Main differences between these classes are performance and result related. Incremental algorithms are faster but produce inferior matching results. [WSP06]

Map-matching approaches of both classes are presented later in this chapter. This categorization also matches the headings of real-time and post-processed matching requirements. In case of real-time applications such as a navigation system that is matching a currently driving car in order to display the correct position, an incremental algorithm is needed. Whereas, global algorithms are better suited for post matching trajectories, because of more precise results.

3.1 Map-Matching Approaches

In this section two map-matching approaches will be described that utilise techniques different from the ones used in this paper. Whereas, section 3.2 will present algorithms that are closely related in terms of the techniques.

An interactive voting based algorithm was proposed by Yuan et al. in [YZZ+10]. It is specially targeted at the problem of low sampling rates. Low sampling rates are defined to have measurement intervals greater than one minute. Their approach uses three insides. The topology and context of the road network, as defined earlier, relations between measurement points, and a weighting factor. The weight for topology, context, and point relation influences is weighted dependent on distance to the measured point. For example, a topological property like a turn is weighted utilizing the euclidean distance between this property and the currently considered measurement. This situation can be seen in figure 3.1 for point f. Point relation influences are nicely illustrated by the points a, b, c, d, e. Here points a, b, d, and e clearly suggest that c should be matched to the E Yesler Way. All of the insides are combined in a voting manner. Hence, measurements a, b, d, and e are voting for the E Yesler Way each weighted by the distance to c. Due to the use of related points not only previous to the considered point but also subsequent, the algorithm needs to have a global view. Hence, it is a global map-matching approach as defined previously.

The algorithm consists of the four main phases candidate preparation, score matrix building, interactive voting, and path finding. The first step of the candidate preparation is to collect all segments of the road network which are located within a fixed radius. This set is called candidate road segments. Candidate points are retrieved by geometric projection of the measurement point onto each segment. If the projection is between the end points of the segment the projection point becomes a candidate otherwise the closest end point will be picked as candidate. Secondly, the static score matrix is build by using a so-called candidate graph. This graph connects every point of subsequent candidate sets and annotates transitions by a spatial analysis function

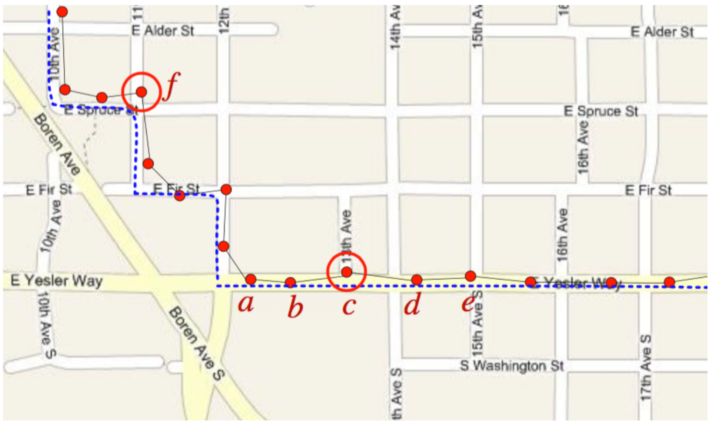

Figure 3.1: Example for the weighted insides [YZZ+10]

that combines the Gaussian observation probability with a transition probability. The transition probability function utilizes the distance between measurement points and candidate points, as well as the shortest path between consecutive candidate points, which is based on the assumption that the real path is most likely the shortest one. Using the static score matrix and the distance between sampling points matrices are build in order to model the weighted influence of each candidate point. Voting is done as follows. For every sampling point the weighted score matrix is calculated. The weighted score matrix contains all candidates for the current sampling point, for all of these candidates an optimal path is calculated that passes through this candidate. Every candidate that lies on this path is voted up. The matching result is then constructed by including the elected candidate for each candidate set.

The time complexity is stated to be $\mathcal{O}(nk^2 m \log m)$ where n denotes the number of measured points in the trajectory, m the number of road segments, and k is assumed to be the maximum number of candidates per sampling point. The nk^2 factor originates from the matrix building, whereas the $m \log m$ factor is caused by finding shortest paths using the Dijkstra algorithm. Evaluation was done using 26 trajectories from the GeoLife Dataset [ZCXM09]. The stated overall result is that the voting based approach is appropriate for low sampling rate matching.

Another theoretical model that is used to develop map-matching algorithms is the Hidden Markov Model (HMM). A HMM consist of hidden states, observations, transitions probabilities, emission probabilities, and the probability for any given state to be the initial state. Hidden states are connected through transitions probabilities. Observations are connected to hidden states via emission probabilities. After constructing a HMM one could use observation sequences to infer knowledge about the hidden states of the model.

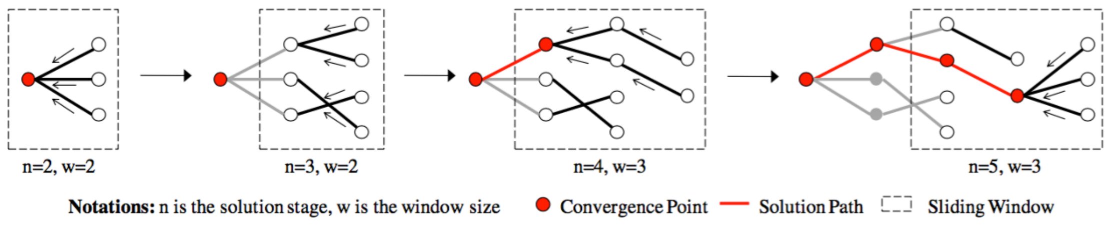

Figure 3.2: Sliding window of online HMM approach [GDM+12]

One approach that applies HMM in map-matching was proposed by Dauwels et al. in [GDM+12]. It is targeted at delivering a map-matching algorithm that suits the needs of low latency real time applications. The basic idea is to construct a HMM such that the candidate road segments of measured points are represented as hidden states with an associated emission that relates to the probability of being the true position. Intuitively, one could choose these emissions according to the distance between measurement and candidate point. The transition probability is computed for every pair of subsequent hidden states such that it follows the Markov assumption. The goal of finding a match can then be translated to finding the Markov chain with highest joint emission and transition probability.

The algorithm starts for every new trajectory point by selecting candidate segments within a fix radius of 50 meters around the measured point. In the next step the emission probability for these candidates are calculated by using a Gaussian function. A speeding penalty is also included, assuming that the speed of observed objects does not greatly exceed the speed limit. Transition probabilities of adjacent states are calculated using the shortest path assumption together with the two scoring functions distance discrepancy T and momentum change M. The T function compares the measured travelling distance to the distance interpolated by the current path in the HMM. The Momentum function M is used to infer a vehicle heading. Backtracking is then used to derive a partial solution for the current progress. In order to calculate a global optimal match by incremental steps a variable sliding window technique is used. The sliding window contains the part of the HMM that will be considered when the next trajectory point is processed. Hence, the window expands with every new trajectory point. It shrinks whenever a convergence point is found in the HMM. Convergence points are points such that every subsequent path convenes in at that point and, thus, share a common prefix path [BR08].

This process is illustrated by figure 3.2. However, it may be the case that no convergence point appears, therefore, a bounded version of the variable sliding window is proposed. This version of the algorithm will restrict the window size to a given threshold. As not all relevant states are considered in this version, it will lead to suboptimal solutions. The algorithm terminates after the last trajectory point. This algorithm can be classified as advanced, incremental, and real time.

Evaluation was done using four bus routes of Singapore, two urban and two rural tracks. Due to the known real track, it is possible to compare it to the matching result. The trajectories were collected using GPS enabled smartphones configured to sampling intervals between 1 to 3 seconds. In order to simulate other sampling intervals the original data was sub-sampled ranging from 10 seconds to 5 minutes.

3.2 Weak Fréchet Map-Matching

This paper proposes a map-matching algorithm which is based on the weak Fréchet distance. In this section two related papers which make use of this distance metric, as well as similar calculation techniques, will be described.

In "On Map-Matching Vehicle Tracking Data" [BPSW05] Brakatsoulas et al. presented two algorithms to solve the map-matching problem. The incremental algorithm is designed to process portions of trajectories rather than the whole instance. In contrast, the global algorithm is working on complete trajectories. The differences regarding computation speed and accuracy will be stated after a detailed description of the algorithms.

A greedy point after point strategy is used by the incremental algorithm. For every measured point the matching is done based on the previously matched road segment. The candidate selection is done by selecting all segments adjacent to the previously matched road segment, this includes reflexivity. The final match is then selected from these candidates utilizing the two similarity measurements orientation and distance. The distance is calculated by projecting the trajectory point onto the great circle defined by the road segment. If the projection lies between the segments endpoints, the perpendicular line distance is used, otherwise the distance to the closest endpoint is chosen. Higher orientation difference or distance results in less similarity. Consequentially, the nearest candidate under these two metrics is chosen as matching segment. If the projection of the current trajectory point onto the matched road segment is not within the endpoints the algorithm will not continue with the next point but advance on the road network. Road segments that are matched this way are called traversed segments, as they are only matched due to a direct or transitive following segment. The algorithm initialisation is done by a range query of fix distance around the starting point and evaluation of all found candidate segments.

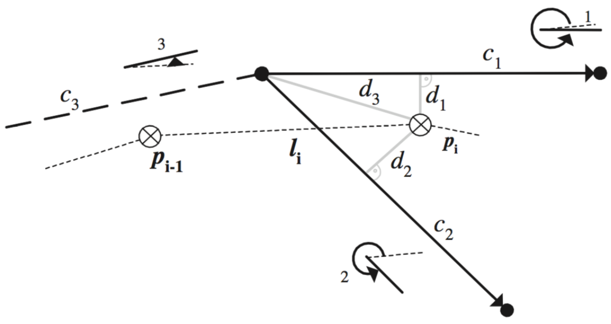

Figure 3.3: Example situation for the incremental approach [BPSW05]

An example can be seen in figure 3.3. The current point to be matched is p_i. The candidates are all adjacent segments of the last match c_3 of point p_{i-1}, those are c_1, c_2, c_3. Due to lower distance and orientation difference the candidate c_1 is chosen as matching segment.

One proposed modification of this simple algorithm is to use a fixed look-ahead. The matching changes by adding a new score function, which uses the best sub-path of all points in look-ahead distance. This modification adds computation time in a trade with accuracy. In regard of time complexity this leads to $\mathcal{O}(na^{l+1})$ running time. Here a denotes the number of adjacent road segments and l the number of segments in the look-ahead. It is stated that a and l are essentially constant and that the initialisation is dominated by the actual computation, thus a complexity of $\mathcal{O}(n)$ is assumed.

Another goal was targeted by the global algorithm proposed in [BPSW05]. In contrast to the fast incremental algorithm, the global approach aims to deliver guaranteed quality. The weak Fréchet metric was used to select a match as the curve in the road network which is closest to the curve described by the trajectory. The free space definition 2.3 and the corresponding free space diagram are extended by [BPSW05] to model the free space between the road network and the trajectory. The extension is called free space surface.

Definition 3.1 (Free space surface)**.** *The free space surface is the union of all free spaces, which in turn are formed by the combination of road and trajectory-segments. A single vertex v of the road network and a trajectory point form a one dimensional free space. Hence, given a road segment, all adjacent segments share a one dimensional free space at v, thus, these vertices can be seen as connecting points.*

The example free space surface in figure 3.4 helps to understand this model by visualising it in a similar way as the free space in section 2.2. On the left, one can see a graph representing a road network and on the right a corresponding free space

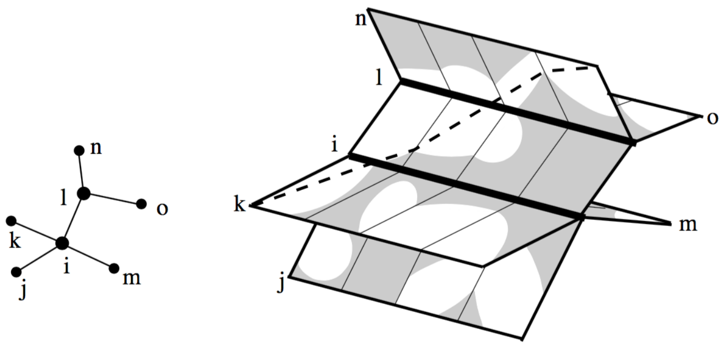

Figure 3.4: Example free space surface [BPSW05]

surface is shown. Presented in bold are the connecting one dimensional points which are created by the combination of vertices i, l and a trajectory show indirectly by the white areas of the free space surface.

The global algorithm uses depth-first search to find a path from any lower left corner to any upper right corner given some ε. As described in section 2.2, an existing or non-existing path corresponds to the solution of the following decision problem: Does any curve with weak Fréchet distance at most ε exist? Parametric search is then used to find the smallest ε and its corresponding curve. This curve is then returned as match for the given trajectory. Every time the decision problem has to be solved an $\mathcal{O}(mn)$ factor of computation time is added. Here m denotes the total amount of vertices and edges contained in the road network and n the number of points in the trajectory. Due to logarithmic parameter search, the algorithm solves the matching problem in $\mathcal{O}(mn \log mn)$.

An empirical evaluation was performed to show differences between the two algorithm in terms of speed and quality. They used 45 vehicle tracks recorded around the Greek city of Athens. Those trajectories are recorded using a sampling rate of 30 seconds. The conclusion regarding computation speed is rather simple, as stated above the incremental algorithm is faster, as its time complexity is essentially linear. Brakatsoulas et al. used two quality measurements to compare the quality of computed matches. The first metric used was the average Fréchet distance, which takes the average of certain Fréchet distances, see [BPSW05] for further definitions. Additionally the basic Fréchet distance was used as a quality indicator. The average Fréchet distance indicated far better results for the global algorithm while the less strict average Fréchet distance only showed a smaller advantage. Another important result stated in this paper is that the global algorithm and a variant using the basic Fréchet distance delivered the exact same match results. This implies that every match computed using

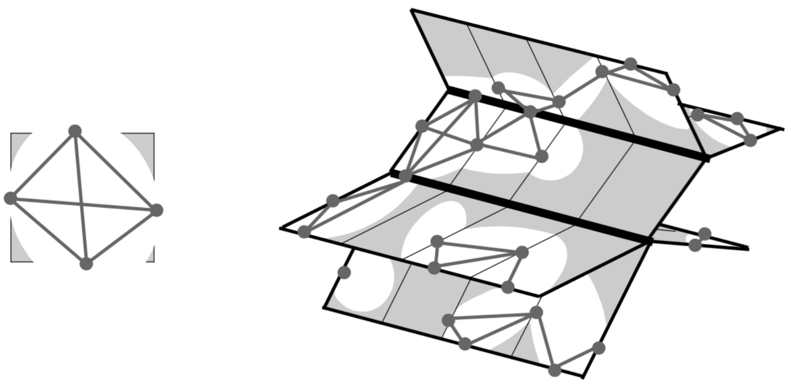

Figure 3.5: Example free space graph [WSP06]

weak Fréchet distance resulted in monotone curves through the free space surface. Hence, one could use the faster to compute weak Fréchet distance.

In "Addressing the Need For Map-Matching Speed: Localising Global Curve-Matching Algorithms" [WSP06] Wenk et al. presented an incremental and output-sensitive algorithm named Adaptive Clipping Algorithm. The main goal of this paper was to deliver a fast algorithm to solve the map-matching problem without sacrificing match quality.

One essential part of the algorithm is the free space surface representation. Free space is modelled as a graph called free space graph. For every white/open interval of a free space cell a vertex is introduced. Furthermore, all vertices of the same cell are connected by graph edges. This model is shown in figure 3.5. This representation is modified to allow incremental computation of the free space graph, and therefore the free space surface, to meet the algorithm's needs. Every pair (e, v), where e is a road network segment and v a point of the trajectory, is a possible edge of the free space graph. The algorithm calculates the smallest ε such that the corresponding interval is non-empty and annotates the free space graph vertex with this value. This ε is equivalent to the minimal distance between the road segment and the trajectory point. Let v_s be the point created through geometric projection of v onto the great circle defined by the segment e. If v_s is located between the end points of e, the smallest ε is given by the distance between v and v_s. Alternatively v_s is located outside of the segment and the minimal distance, and therefore the smallest ε, is given by the distances between v and the closest segment end point. By defining the weight of a path as the maximum ε that is encountered, the weight of a path from any lower left corner to any upper right corner of the free space surface corresponds to the weak Fréchet distance between the trajectory and the curve implied by the path. Wenk et al. then used Dijkstra's shortest path algorithm to create an output-sensitive algorithm. This was achieved by running Dijkstra on the free space graph and calculating free space cells and there minimal ε values as needed. In contrast to all other global algorithms using weak Fréchet distance, the free space surface is only calculated partly. This approach leads to a time complexity of $\mathcal{O}(K \log K)$ were K is the calculated part of the free space surface. Hence, K has an upper bound of nm, which is the complexity of the whole free space surface.

A modified version of this algorithm is also presented in order to localize global weak Fréchet map-matching, called Adaptive Clipping algorithm. Active regions are used to split the free space surface into intersecting parts. The active region for start and end points of the trajectory are defined as a circle with fix radius around the point. Intermediate active regions are defined by an ellipse which is constructed using two consecutive trajectory points. The radius and ellipse calculation are chosen according to the knowledge about measurement errors. However, the implementation was done using bounding boxes as an approximation. The modified algorithm runs through the following stages. Stage one begins by execution of the Dijkstra based approach within the first and second active region. All start vertices are defined by (p_1, e) where p_1 is the first track point and e is any edge within the first active region. Target vertices are defined in the same manner as (p_2, e). Every remaining stage uses the resulting shortest path of its preprocessor stage to construct the new and extended shortest path. Hence the last stage computes the shortest path between active regions $n-1$ and n and is seeded with the shortest path from region 1 to region $n-1$. The runtime of this localized approach is stated as $\mathcal{O}(M \log M) + n$ with $M = \sum_{i=1}^{n-1} M_i$ and M_i the number of edges and vertices of active region i. This time complexity can be simplified to $\mathcal{O}(n \log n)$ under the assumption that only adjacent regions intersect and that M_i is essentially constant.

This approaches are conceptionally close to the approach described by this paper. They share the same underlying distance measure and also the usage of some kind of free space surface. But they heavily differ in their ways to construct the resulting match. While the approaches described in this section use Dijkstra's algorithm or the decision problem and parametric search to find a match, the approach proposed by this paper will use a union find structure to construct a match. In contrast to all algorithms of this section it will be easy to extend the proposed algorithm to facilitated parallelization.

4 Algorithm

This chapter presents the developed incremental weak Fréchet map-matching algorithm. Firstly an overview of the general idea behind the algorithm is given. This overview is followed by the description of several necessary subroutines. All steps will be exemplified by the example match show in figure 4.1. A parallel variation of the algorithm is introduced. Then the incremental weak Fréchet map-matching algorithm is illustrated as pseudo code and analysed in terms of time complexity.

Overview

As already mentioned in section 2.1, the input of the map-matching problem is a graph representing some kind of map and a sequence of positional measurements often called track or trajectory. The map-matching problem is solved by computation of a curve within the map that is as close as possible to the trajectory by means of some distance function, which, in context of this work, is the Fréchet distance.

In order to give an overview over the algorithms inner workings one can divide it into its main tasks. Referring to the map-matching algorithm proposed in this thesis, the main four task are the following:

1. Extracting the required part of the map graph
2. Computing the free space graph
3. Finding a conditioned connection
4. Constructing the match

The first task is to extract a part of the map graph such that it contains just enough information. A map part contains enough information for the map-matching algorithm as long as it includes the theoretically best solution, which, in this context, is the closest curve compared to the given trajectory. However, it is also not feasible to just use the whole map, although it certainly contains enough information. The whole

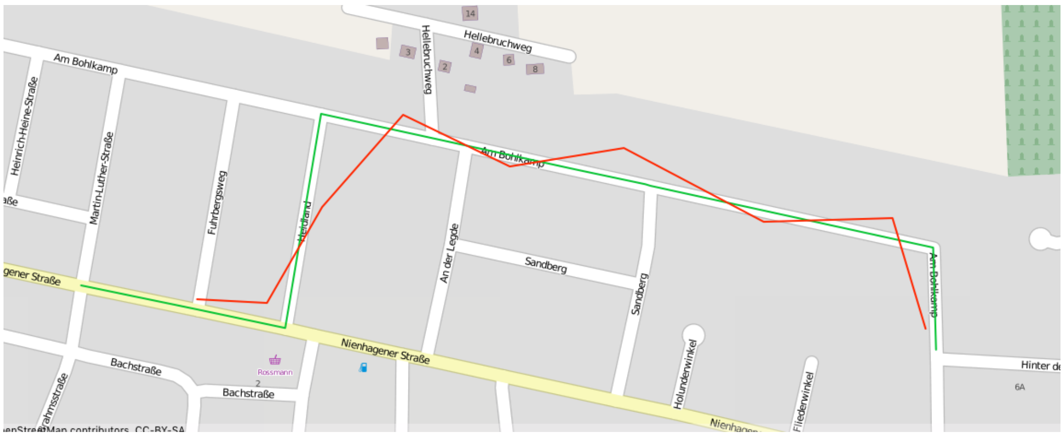

Figure 4.1: Example track (red) and its corresponding match (green)

map is most likely to big for the next steps of the algorithm, such that they would not be able to produce a result at an acceptable speed.

Henceforth, the extracted map graph is combined with the trajectory to compute the free space graph, which represents the free space surface as introduced in definition 3.1. Every free space cell becomes a vertex of the free space graph and for every pair of adjacent free space cells their corresponding vertices get connected via an annotated edge in the free space graph. The annotation contains the minimal distance ε such that the two cells are connected by free space in the free space diagram. Additional start and end vertices are added while constructing the free space graph. Every vertex that had its origin in a cell that is made up on the first segment of the trajectory and any map segment receives one start vertex. Analogue for end vertices.

Now, one can find a path through this free space graph from any start vertex to any end vertex such that there is no other path within the free space graph from any start vertex to any end vertex that has a lower maximum of annotated distances ε. This path, as described in 2.2, directly leads to a parametrisation for the weak Fréchet distance and, therefore, delivers a path through the map graph. This path is, due to the stated condition, the closest curve compared to the trajectory under the weak Fréchet distance.

Finally, the found path must be projected back to the original map graph and converted into the desired format.

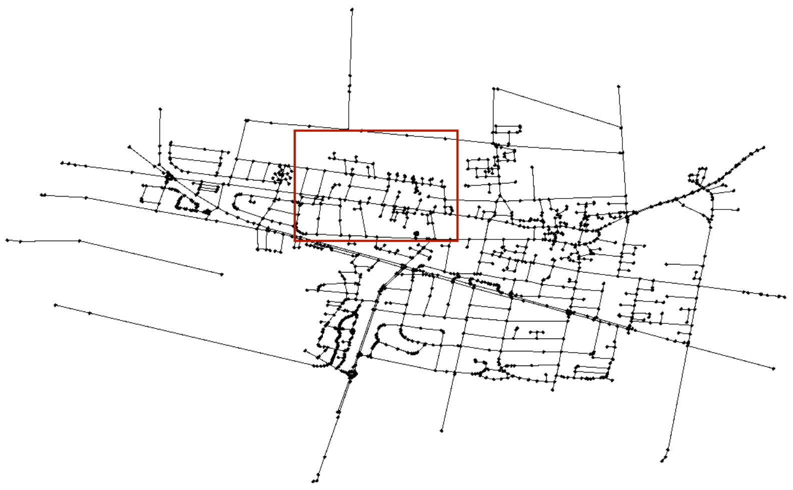

Figure 4.2: Roadmap graph of Wathlingen

4.1 Map Extraction

As stated in the overview, it is necessary to reduce the map to an excerpt that contains enough information and is, at the same time, sufficiently small to produce acceptable computation times. The map graph of the given example is illustrated in figure 4.2. It contains more information than needed, because the track only covers a small part of the map. This part is indicated by the red rectangle.

Due to the measurement and sample errors described in section 2.1, the area of interest can be formalised as a sequence of ellipsoids around the measured trajectory. [WSP06] This representation, however, is computationally hard to handle and, therefore, not appropriate for the proposed algorithm. Hence, the same approximation is used as in [WSP06]. Every measured point is interpreted as the center of a square which contains all the possible actual positions, those squares are called bounding boxes.

In order to construct a square of the desired length a somewhat related problem to the distance calculation described in 2.3 is used, the problem to find the coordinates of a point which has a desired great circle distance in respect of some already given reference point. The result is highly ambiguous due to the fact that there are infinite coordinates around the reference point which would form a circle with a center in the reference point. For most use cases, a bearing will also be given and, therefore, reduce the range to an unambiguous result coordinate. As stated by Chris Veness [Ven16] these can be calculated as described below.

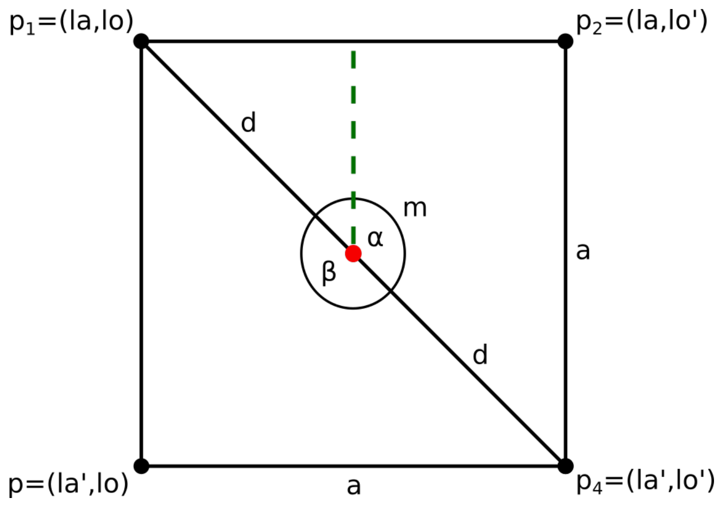

Figure 4.3: Bounding box

Calculation 4.1 (Point in given distance [Ven16])**.** *Given a start point* (ϕ, λ)*, a demanded distance* d*, an initial bearing* θ*, and the earth's radius* r *one can calculate the end point* (ϕ', λ') *as follows*

$$\begin{aligned}\phi' &= \arcsin(\sin(\phi) \cdot \cos\left(\frac{d}{r}\right) + \cos(\phi) \cdot \sin\left(\frac{d}{r}\right) \cdot \cos(\theta)) \\ \lambda' = \lambda + \arctan(\sin(\theta) \cdot \sin\left(\frac{d}{r}\right) &\cdot \cos(\phi), \cos\left(\frac{d}{r}\right) - \sin(\phi) \cdot \sin(\phi'))\end{aligned} \tag{4.1}$$

The desired square is constructed by calculating the coordinates of the top left and bottom right corners and constructing the top right and bottom left out of the calculated latitude and longitude pairs. As illustrated in figure 4.3, the top left point $p_1 = (la, lo)$ is calculated with the parameters $2d$ desired diagonal and the initial bearing β of $115°$. The second point $p_4 = (la', lo')$, located bottom right, is calculated using an initial bearing α of $45°$. The missing corners are then arranged as $p_2 = (la, lo')$ and $p_3 = (la', lo)$, again shown in figure 4.3. The side length of the constructed square is given by $a = r \cdot \arccos(\sqrt{\cos(\frac{2d}{r})})$, which can be obtained through the Pythagorean theorem for spheres.

The bounding box size has to be big enough such that the resulting subgraph does not fall apart at sections needed by the match. However, it should be as small as possible. The size used in this papers evaluation was chosen as the smallest possible per test set. Note that this parameter is highly dependent on the underlying map graph and track properties. After calculating all bounding boxes the resulting subgraph of the given example turns out as shown in figure 4.4.

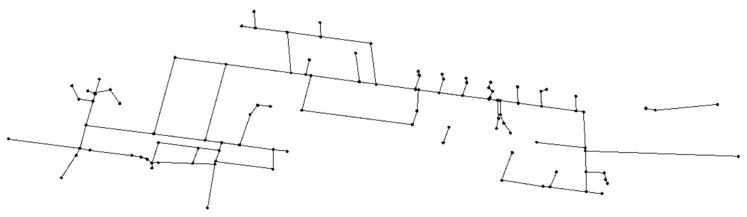

Figure 4.4: Roadmap within bounding boxes ($d = 0.15$ km)

4.2 Free Space Graph Computation

After the extraction of the relevant map graph, the free space graph has to be calculated. As stated in the overview, the vertices of the free space graph are the free space cells of the free space surface, which can be constructed by combining map and track data. Vertices are connected if the corresponding free space cells are adjacent. Every edge must be annotated with the minimal distance ε such that there is free space between those two cells within the free space diagram. In this section, the detailed definition of those vertices and edges is given. In addition, the construction procedure for the free space graph will be explained.

Every free space cell and, therefore, free space vertex v is made up of one map and track edge, $v = ((v_m, v'_m), (v_t, v'_t))$. All the map edges have to be orientated in one direction corresponding to one start vertex in order to guarantee connectivity of the resulting free space graph. This restriction originates from the calculation method, which will be described later. Track edges are constructed by starting at one end of the track and creating one edge for every subsequent pair of measurement points, $(t_1, t_2), (t_2, t_3), \dots, (t_{n-1}, t_n)$.

There are two types of adjacent cells and, therefore, vertices, Right and Upper-Cells. The Right-Cell and the current cell share the same map edge but the Right-Cells track edge is shifted one position towards the end of the track. Hence, a transition from the current cell to its Right-Cell does not add a new map edge to the match result and only advances on the track. Another interpretation is that the added track point is matched to the same map edges as the previous track point. Every cell has exactly one Right-Cell except cells containing the track end (t_{n-1}, t_n). Transitions to Upper-Cells are advances on the map part of the free space graph. Hence, these transitions extend the match result by one map edge. The number of Upper-Cells n connected to a

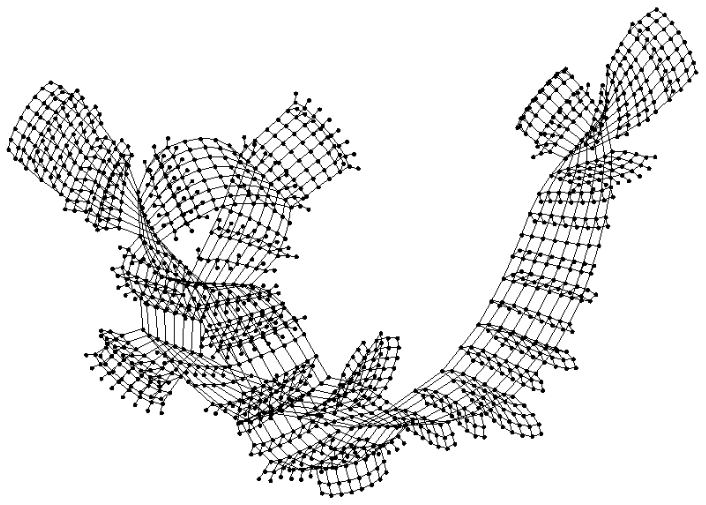

Figure 4.5: Calculated free space graph

current cell $v_c = ((v_m, v'_m), (v_t, v'_t))$, therefore, corresponds to the degree of the "upper" vertex of the map edge contained in the current cell, $n = |v'_m| - 1$.

The free space graph is constructed from the lower left to the upper right. The starting point can be found by taking a random start edge $e_s = (v, v')$ of the map graph. In order to avoid separation of the created graph the start edge has to be processed in both orientations, (v, v') and (v', v). Then breadth-first search is used on the map graph starting at e_s. This guarantees the needed orientation of map edges. For every encountered map edge (v_m, v'_m) all track edges $(t_1, t_2), (t_2, t_3), \ldots, (t_{n-1}, t_n)$ are traversed and all corresponding vertices $((v_m, v'_m), (t_1, t_2)), \ldots, ((v_m, v'_m), (t_{n-1}, t_n))$ are created. For each of these vertices their Right and Upper-Cells are calculated, connected and annotated with the minimal ε such that these cells are connected by free space in the free space surface.

Additional Start and End-Vertices are added to the graph as follows: For each vertex $v = ((v_m, v'_m), (t_1, t_2))$ that contains the first track point t_1 a new Start-Vertex $v_s = ((v_m, v_m), (t_1, t_1))$ is introduced. Edges $(v_s, v, dist(v_m, t_1))$ are added to connect the Start-Vertices to the graph. In a similar way End-Vertices are added and connected to all vertices that contain the last track point using the second tuple elements. The free space graph of the given example can be seen in figure 4.5.

One main task, while building the free space graph, is to calculate the minimal ε values. For transitions from a current cell $v_c = ((v_{m_c}, v'_{m_c}), (t_i, t_{i+1}))$ to the Right-Cell $v_r = ((v_{m_c}, v'_{m_c}), (t_{i+1}, t_{i+2}))$ these values correspond to the minimal distance between the segment (v_{m_c}, v'_{m_c}) and the point t_{i+1}. Transitions towards an Upper-Cell

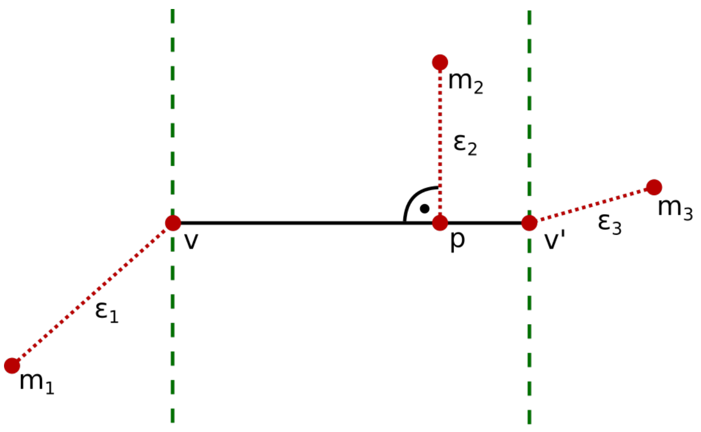

Figure 4.6: Minimal distance between point and segment

$v_u = ((v'_{m_c} = v_{m_u}, v'_{m_u}), (t_i, t_{i+1}))$ correspond to the minimal distance between the segment (t_i, t_{i+1}) to the point $v'_{m_c} = v_{m_u}$, note that the minimal distance is the same for all Upper-Cells. Hence, all transition values can be determined by calculating the minimal point-segment distance. Three cases for the minimal point-segment distance are illustrated in figure 4.6. The segment is defined by the two points v and v'. Case m_1 and m_3 are located outside of the area enclosed by the two circles orthogonal to v and v', whereas case m_2 lies within this area. The minimal distance for m_1 and m_3 is given by the great circle distance between the point and the start or end-point of the segment as described in section 2.3. Therefore, the minimal point-segment distance for m_1 is given by ε_1 and m_2 has a minimal distance of ε_3. For case m_3 the minimal distance is equivalent to the great circle distance between m_2 and p, which is the orthogonal projection of m_2 onto the segment.

Calculation 4.2 (Point-Segment Distance)**.** *Given a segment defined by two points $\vec{s_1}$ and $\vec{s_2}$ and one point $\vec{p}$ in vectorial representation, the minimal point-segment distance can be calculated as follows: Calculate the two intersection points $\vec{i_1}$ and $\vec{i_2}$ of the great circle g_1 defined by $\vec{s_1}$, $\vec{s_2}$ and the great circle g_2 defined by $\vec{p}$ and the normal of the first great circle $\vec{n_1} = \vec{s_1} \times \vec{s_2}$, see 2.3. These two great circles are orthogonal to each other, hence, the intersections are lot points of $\vec{p}$ onto the first circle g_1. The intersection points are calculated using the cross product of the normal vectors $\vec{n_1}$ and $\vec{n_2} = \vec{n_1} \times \vec{p}$, $\vec{i_1} = |\vec{n_1} \times \vec{n_2}|$ and $\vec{i_2} = |\vec{n_2} \times \vec{n_1}|$. Also calculate the angles $a_p^{s_1}$, $a_p^{s_2}$ from $\vec{p}$ and the segments start and end point. Now, one can calculate the minimal point-segment distance by using the point to point distance calculation described in 2.2:*

$$dist_s(s_1, s_2, p) = \begin{cases} min\{dist(i_1, p), dist(i_2, p)\} & , a_p^{s_1} < a_{s_2}^{s_1} \vee a_p^{s_2} < a_{s_2}^{s_1} \\ min\{dist(s_1, p), dist(s_2, p)\} & , else \end{cases} \tag{4.2}$$

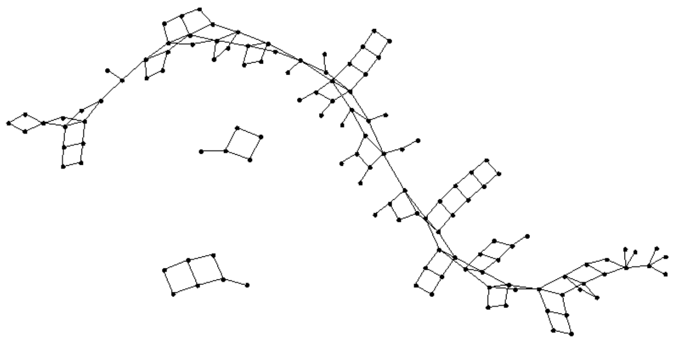

Figure 4.7: Freespace graph till connectivity

One possibility to speed up the free space graph computation is to parallelize it. It is also a simple way of improvement due to the free space graph and computation structure that suggests such a modification. One can divide the free space graph along the track or the map graph axis in order to package graph parts that can be created independent. The package size can be adjusted to the execution environment, for example the track could be partitioned such that it fits the available parallel computation cores.

4.3 Match Search

The next task of the algorithm is to find a connection in the created free space graph from the lower left to the upper right, such that the maximal path annotation value is minimal for the whole free space graph. This corresponds to finding the minimal ε such that there is free space in the free space surface between the lower left and the upper right, which in turn states that this ε is the minimal weak Fréchet distance between the given map and track. This path, therefore, provides the optimal match under weak Fréchet distance.

While creating the free space graph, it is stored as edge tuples. A priority queue that considers edges with smaller annotation value to have higher priority is used as data structure. It supports the retrieval of the next edge with minimal transition ε in logarithmic time, see section 2.4.2. Due to the same ε value transitions to Upper-Cells are stored as a single queue entry. A union-find structure is used to detect the desired connection, see section 2.4.1.

This union-find structure is build by successively removing free space graph edges from the priority queue and invoking the union-operation on the corresponding two vertices. After every processed free space edge, the find-operation is used to test for connectivity. The find-operation is invoked on the first start and end-vertex that was found within this process. All start vertices are held within one union class by always calling the union-operation on the first start vertex and the currently encountered start vertex. In the same way end vertices are held in one class. Hence, a positive find call states a connection from the lower left (any start vertex) to the upper right (any end vertex).

The vertices and edges are not only added to the union find structure but also stored in an adjacency list based graph structure. This graph is used to extract the match path by using breadth-first search from the first start vertex to the first end vertex if the union find structure has detected a connection. For the given example this graph is shown in figure 4.7.

The final step is to extract the match from the match path in the free space graph. The match path is traversed from start to end. For every vertex the map part is extracted and merged to construct the final result. Due to the nature of the free space surface subsequent vertices may have the same map part. In this situation no new map edges are added to the match. After this step the final match result can be formatted into the desired format and returned to the invoker.

4.4 Code and Complexity

In this section the proposed algorithm will be illustrated as Pseudocode. Furthermore, the time complexity of the algorithm is developed and discussed.

The overall matching procedure is shown in Algorithm 4.1. Every matching process is started by supplying the two parameters map graph and track sequence. The map graph $G_m = (V_m, E_m)$ defines the match target, whereas the track sequence $P = (p_1, \ldots, p_n)$ defines the data to be mapped into the match target.

The first hurdle that needs to be cleared is the calculation of the free space graph from the given input G_m and P. This task is done in the subroutine freespace, see algorithm 4.1 line 4 and algorithm 4.2. One important question in order to determine the complexity is the following: How big does the edge set of the free space graph get? This value is important as it defines the maximal number of priority queue entries as well as the maximal number of combined while and *for loop* passes seen in 4.1.

Algorithmus 4.1 Matching procedure

```
1: procedure MATCH($G_m = (V_m, E_m)$,$P = (p_1, \ldots, p_n)$)
2:     $c_s, c_e \leftarrow \emptyset$
3:     $G_r \leftarrow (V_r = \emptyset, E_r = \emptyset)$
4:     $F \leftarrow$ freeSpace$(G, P)$
5:     while find$(c_s)! =$ find$(c_e)$ do
6:         $(c_i, C_j, \varepsilon) \leftarrow$ deleteMin$(F)$
7:         unionPointCells$(c_s, c_e, c_i)$
8:         for $c_l \in C_j$ do
9:             unionPointCells$(c_s, c_e, c_l)$
10:            $V_r \leftarrow V_r \cup \{c_i, c_l\}$
11:            $E_r \leftarrow E_r \cup \{\{c_i, c_l\}\}$
12:            union$(\mathsf{c_i}, \mathsf{c_l})$
13:            addEdge$(\mathsf{c_i}, \mathsf{c_l}, \mathsf{G_r})$
14:        end for
15:    end while
16:    return shortestPath$(G_r, c_s, c_e)$
17: end procedure
```

Algorithmus 4.2 Free space graph creation procedure

```
1: procedure FREESPACE($G_m = (V_m, E_m)$,$P = (p_1, \ldots, p_n)$)
2:     $F \leftarrow$ newPriorityQueue()
3:     for $e \in E_m$ do
4:         for $p_i \in P$ do
5:             if $i = 1$ then
6:                 add$(F,$ startTransition$(e, p_i))$
7:             end if
8:             if $i = n$ then
9:                 add$(F,$ endTransition$(e, p_i))$
10:            end if
11:            add$(F,$ rightTransition$(e, p_i, P))$
12:            add$(F,$ upperTransitions$(e, p_i, G_m))$
13:        end for
14:    end for
15:    return $F$
16: end procedure
```

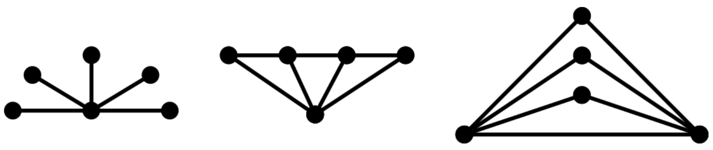

Figure 4.8: Critical subgraphs $K_{1,5}$, $P_4 + K_1$, and $K_2 + \bar{K}_3$

First, the free space graph is defined as $G_f = (V_f, E_f)$. As described in section 4.2, the vertices V_f are cells of the free space surface $V_f = E_m \times P_E$ with $P_E = \{(p_i, p_{i+1}) | p_i, p_{i+1} \in P \wedge 0 \leq i < n\}$. Hence, free space graph edges are defined as $E_f \subset V_f \times V_f \times \mathbb{R}$, but they are processed as so called transitions. Transitions are used to bundle edges from a given start vertex with identical annotation value, therefore, the transition structure is $V_f \times \mathcal{P}(V_f) \times \mathbb{R}$. Going back to the original question, the number of queue entries is maximal if every transition only contains one end vertex. In this case, the *for loop* body within the match procedure will always be executed once. The other extreme would be reached if there was only one transition containing all edges. In this case, all edges are handled through the *for loop*. One can see that the combined number of while and *for loop* passes is determined by the number of free space graph edges $|E_f|$.

This set can be split up into the three subsets $E_f = E_{f_{point}} \cup E_{f_{right}} \cup E_{f_{upper}}$. The amount of edges from start cells and to end cells $|E_{f_{point}}|$ can easily be calculated if $|E_m|$ is known. One could also divide the whole free space graph into layers. Each layer $G_{f_i} = (V_{f_i}, E_{f_i})$ is a subgraph induced by the vertices $V_{f_i} = \{(e_m, (p_i, p_{i+1}) | e_m \in E_m)\}$, note that all layers are equivalent apart from naming. Now $E_{f_{point}}$ contains $2|V_{f_i}|$ edges due to the fact that there is one start cell connected to every vertex of layer one and one connected end cell for every vertex in layer n. The number of edges between layers $|E_{f_{right}}|$ is given by $|V_f| - |V_{f_n}|$, because there is exactly one edge between every equivalent vertex pair per layer.

All missing edges $E_{f_{upper}}$ are now connections within each layer. Every layer G_{f_i} is created by traversing E_m in breadth first style and exchanging vertices with edges and vice versa. The resulting layer graph is conceptionally close to the line graph $L(G_m)$ but edges are only created along the orientation given by the breadth first traversal. In fact every layer graph can be interpreted as a subgraph of $L(G_m)$.

In "Forbidden Subgraphs for Graphs with planar Line Graphs" Greenwell and Hemminger proofed that a line graph $L(G)$ is planar if and only if G does not contain a subgraph homeomorphic to $K_{3,3}$, $K_{1,5}$, $P_4 + K_1$, or $K_2 + \bar{K}_3$ [GH72]. Thus, if the free space graph construction procedure does not create non-planar results for the subgraphs $K_{1,5}$, $P_4 + K_1$, and $K_2 + \bar{K}_3$ and the map graph is planar, the resulting layer graphs will be planar. The forbidden subgraphs are shown in figure 4.8. The construction method was tested against these three graphs, always leading to planar results for all possible traversals. For example the line graph of $K_{1,5}$ is K_5, which is one of two basic non-planar graphs of Kuratowski's theorem. However, the layer graph of $K_{1,5}$ is $K_{1,4}$ and thus planar. Any subdivision of these three graphs just leads to a subdivision of the layer graph produced from the original graph, thus preserving planarity. As a consequence, all layer graphs are planar if G_m is planar. In the domain of matching road-maps and their representation as graphs, it might be sufficient to assume that the match target G_m is planar. In this case, a upper bound for the size of $E_{f_{upper}}$ is given through Euler's formula [DKR13, p. 146]. Every E_{f_i} has an upper bound of $|E_{f_i}| \leq 3|V_{f_i}| - 6$ and the sum of all layer graph edges are therefore bounded to $n(3m-6)$, for $|V_{f_i}| = |E_m| = m$ and $0 \leq i \leq n = |P_E|$.

Otherwise it is certainly possible to assume a constant c_d as the maximal degree of a map vertex. The map traversal returns $|E_m|$ edges which are processed along the orientation and, thus, only adding free space graph edges to upper cells of the leading vertex. Resulting in a bound of $|E_{f_i}| \leq |E_m|c_d$ for $|E_{f_i}|$. Hence, the size of $|E_f|$ can be bounded as follows:

$$\begin{aligned} |E_f| &= |E_{f_{point}}| + |E_{f_{right}}| + |E_{f_{upper}}| \\ &= 2|V_{f_i}| + |V_f| - |V_{f_n}| + |P_E||E_m|c_d \\ &= 2m + mn - m + nmc_d \in \mathcal{O}(mn) \end{aligned} \tag{4.3}$$

The operations startTransition, endtransition, and rightTranisition are considered to be of constant time because of the explanations made above. The sum of all upperTransitions operation times follows the bound for $|E_{f_{upper}}|$ and, therefore, only adds a summand of nmc_d to the time complexity of the algorithm 4.2. Hence, the time complexity of the free space procedure is given by $\mathcal{O}(mn)$ if the priority queue F supports insertions in constant time. This can be achieved by using a Fibonacci heap, see 2.4.2.

The next step of the algorithm is to find the conditioned connection as described in section 4.3. This corresponds to the *while loop* within the match procedure. As mentioned above, the combined *while* and *for loop* passes are bounded to the amount of free space graph edges and, therefore, $\mathcal{O}(mn)$. While executing those $\mathcal{O}(mn)$ passes the interesting procedure calls are union, find, and deleteMin as they are of non trivial nature. However, the time complexity of union-find operations can be expressed by amortized $\mathcal{O}(m\alpha(m,n))$ for m find and $n-1$ union operations. As descried in section 2.4.1, one can assume constant time per operation. By using the bound of free space graph edges this leads to $\mathcal{O}(mn)$ time for union and find operations. More expensive are the maximal mn deleteMin operations which consume $\mathcal{O}(mn \log mn)$ time and, therefore, dominate the *loop* execution time.

Extracting the final match is done via breadth-first search on $G_r = (V_r, E_r)$, see algorithm 4.1. In regard of complexity, this adds a time factor of $\mathcal{O}(|V_r| + |E_r|)$ [DMS14]. This graph can get almost equal to G_f implying $\mathcal{O}(mn + mn)$ time for the match search described in section 4.3.

In summary, the time complexities of the algorithm parts are $\mathcal{O}(mn)$ free space building, $\mathcal{O}(mn \log mn)$ match search, and $\mathcal{O}(mn + mn)$ match extraction. Hence, the match search dominates the resulting time complexity of $\mathcal{O}(mn \log mn)$. However, it is important to be aware that one would expect a considerably lower amount of *loop* passes in practical cases. This is based on the assumption that connections in the free space graph will be found much earlier. In contrast, the free space building time can only be influenced by reducing the considered target graph or track length.

5 Evaluation

The algorithm developed in this thesis will be evaluated in this chapter based on its running time and match quality. First of all, the test setup is described in section 5.1. The evaluation environment explanations include implementation details, hardware, and software specifications, as well as information about the employed test data. In section 5.2 the running time is evaluated by comparing it with an implementation of the Dijkstra based Adaptive Clipping algorithm proposed by Wenk et al. [WSP06], which is described in section 3.2. The evaluation omits the calculation of the required map part and concentrates on the free space graph building and match search. Eventually, the match quality is discussed and exemplified in section 5.3.

5.1 Test Setup

In order to compare the proposed algorithm with the Adaptive Clipping algorithm described in [WSP06], these algorithms had to be implemented. Furthermore, test data and a test system is needed. All these aspects are treated in this section.

The algorithms were both implemented in the Java programming language in version 1.8. Furthermore, these implementations are using the same libraries for various tasks. Both algorithms share the same import method for the map which should be used to match tracks. The compressed map format pbf, see [Fou16c] for further information, is used as input format. Pbf files are processes using the osmosis library in version 0.44.1. The imported map is then stored in an embedded H2 database. The import implementation extracts all vertices and edges that are part of way's in the OpenStreetMap and are tagged as *highway*. However, the test data is explained later on. In order to match a given track, the required map excerpt is loaded into the main memory and stored in an adjacency list implementation of the JGraphT 0.9.2 library [NC16]. All mathematical base operations, such as the dot product and other vector operations, are implemented by calling the apache commons math library in version 3.6.1. Another important data structure for both approaches is the priority queue to store the free space graph transitions. The implementation used for this data structure is the java.util.PriorityQueue, which provides $\mathcal{O}(\log n)$ dequeuing operations. The disjoint-set data structure needed for the algorithm proposed in this thesis is implemented through the UnionFind class of the JGraphT library. This implementation achieves an amortized cost of $\mathcal{O}(\alpha(n))$ per operation call. The resulting matches are returned in the GPS Exchange Format (GPX). This export functionality is implemented by using the Java Architecture for XML Binding (JAXB) to generate the class model from the GPX definition given as XML schema. The schema can be downloaded from various locations, for example [Fos16]. These GPX files not only contain the resulting match, but also the input track, information about the map used, and detailed evaluation information.

The comparison of these algorithms is mainly done with tracks which were extracted from the *planet gps* file [Oc16]. The extracted tracks are bounded by the residential area of Stuttgart and are at least 10 points long. This extraction was done by Martin Seybold, the supervisor of this thesis. This dataset was then further sampled in terms of length and density to fit the needs of this evaluation. Thus, the map of the residential region of Stuttgart was used as the match target. It was downloaded in pdf-format from the servers provided by the Geofabrik company [Kar16]. The tracks are described in more detail later on. Some tracks were synthetically created to evaluate the time complexity and to demonstrate certain quality properties.

All evaluation tests were executed on a single machine. This computer runs a Intel Xeon E3-1230 v3 processor with 4 cores each running on 3.3 GHz. The build-in main memory has a size of 2×8 Gb and runs in dual channel mode. These components are connected through a mainboard which runs an Intel Z97 chip set. The operation system used is a Windows 10 Pro. The installed Java virtual machine is the Oracle JVM in version 1.8.

5.2 Running Time Comparison

In this section the algorithm proposed in this thesis and its parallel version are compared to the Adaptive Clipping algorithm proposed by Wenk et al. [WSP06]. Only the running times of the phases free space graph building and match search are compared. The retrieval of the required map part is not taken into account. However, the retrieval phase could be implemented in the same way for both algorithms, as they require the same input. Firstly, all three algorithms are invoked with seven tracks of size $4, 8, \ldots, 256$ in order to deliver a first insight to the algorithms running times. Followed by a detailed comparison of the free space graph building time of the non-parallel and parallel versions of the proposed algorithm. Finally, the three algorithms

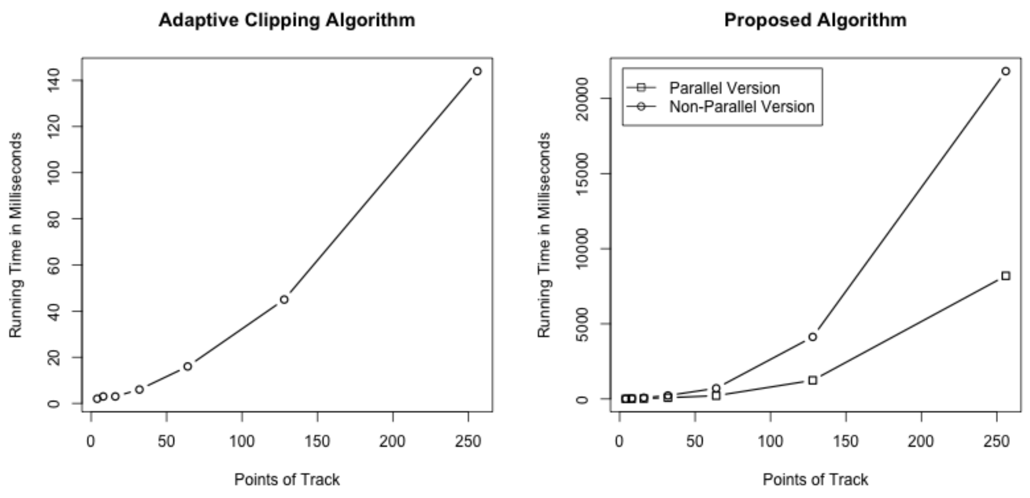

Figure 5.1: Small test with increasing track length

are compared using 80 tracks of four differed sizes. The results are discussed and some possible reasons are described.

For the first test seven tracks of sizes $4, 8, \ldots, 256$ are used to deliver a first insight to the experimental running times. These tracks were created by cutting a track, from the dataset mentioned above, of size 300 to the desired sizes. Thus, this test shows the increasing running times when the given track is extended. The averaged results of five runs are shown in figure 5.1. On the left side one can see the results of the Adaptive Clipping algorithm implementation. The algorithm managed to calculate matches for the first five tracks in under 20 milliseconds. The track of length 128 took an average of 45 milliseconds, whereas the track with 256 points took 144 milliseconds. On the right side the results for both the non-parallel and parallel version of the proposed algorithm can be seen. The non-parallel version took 12 milliseconds for the track of length four and up to 21,81 seconds for the last track of size 256. However, the parallel version, which utilizes eight threads, only took 4 millisecond for the first and 8,19 seconds for the last track. Two important observations can be made by looking at this test. Despite the similar time complexity of $\mathcal{O}(mn \log mn)$, the first observation is that the Adaptive Clipping algorithm outperforms the proposed algorithm within this test by a fairly big magnitude. The other observation is that parallelization of the free space graph building phase seems to be a good way to improve the overall execution time of the proposed algorithm. This supports the assumption that a connection is detected much earlier than after mn free space graph transitions are added to the union-find structure. This implies that the practical running time is dominated by the free space graph building phase.

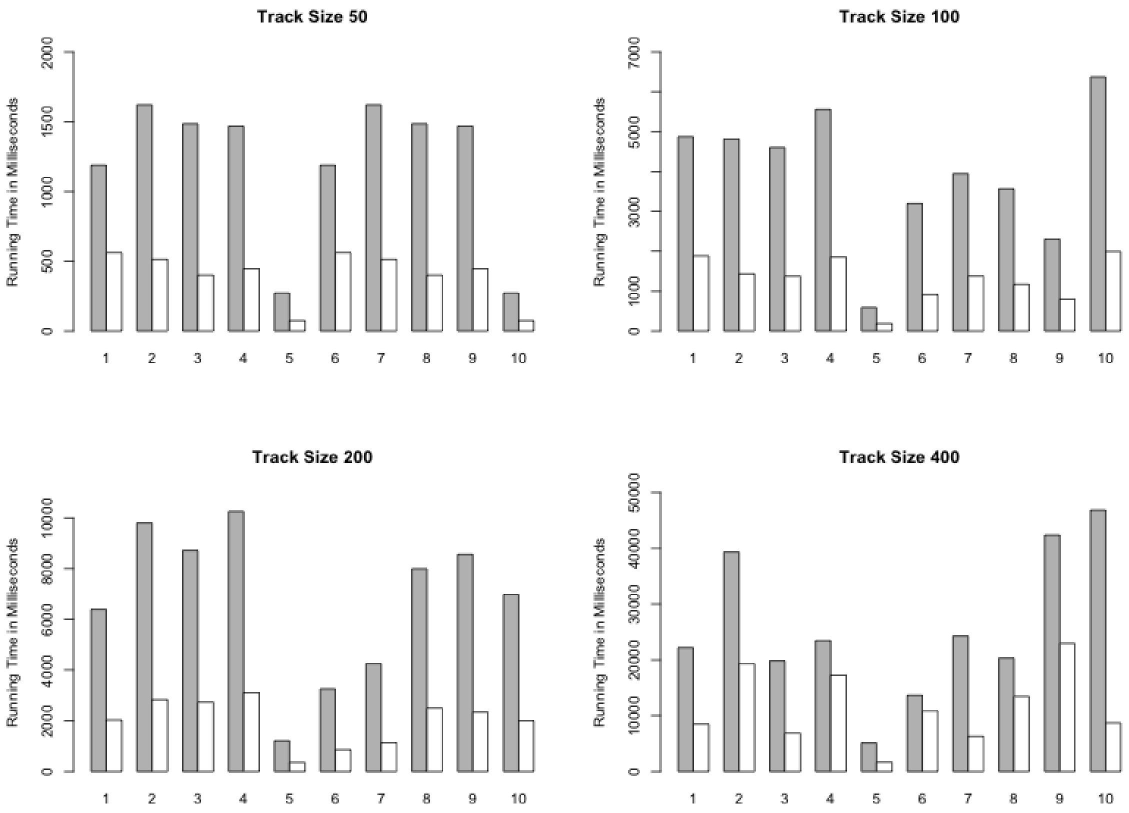

Figure 5.2: Comparison of non-parallel (grey) and parallel (white) free space graph computation

In order to further examine the benefit of parallelizing the free space graph building phase, 80 tracks have been used to evaluate the differences between a single-threaded and a multi-threaded calculation of the free space graph. The multi-threaded executions used eight threads on a four core processor, as mentioned earlier. A part of the results are visualized in figure 5.2. As one can see, four different track sizes were tested, 50, 100, 200, and 400 points. The minimal improvement factor encountered in this test was around 1.3, see track number 6 within the "Track Size 400" chart. A maximal improvement was measured at track 10 within the same chart, improving the free space graph building time at a factor of 5.9. The average of all improvement factors, regarding all 80 tracks, was found to be 3.1 within this test setting. These numbers show that this phase of the algorithm can be improved considerably by using parallelization.

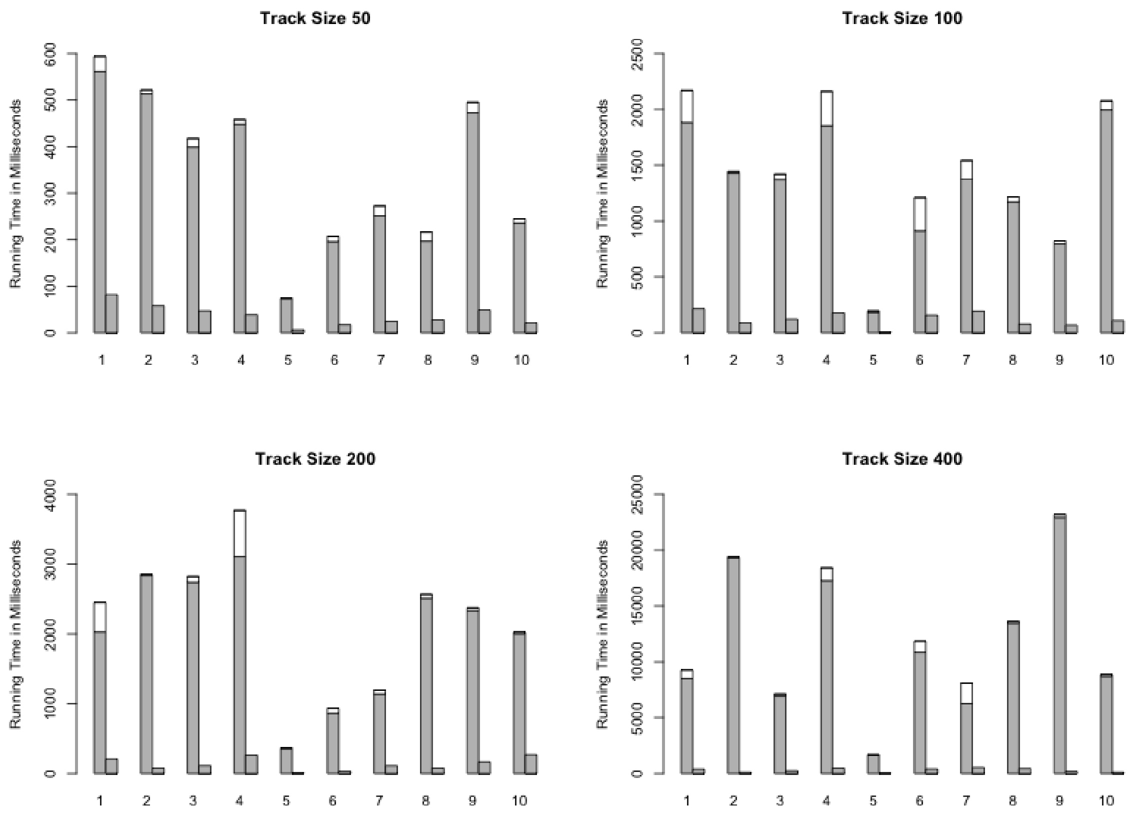

Figure 5.3: Comparison of the proposed algorithm with the Adaptive Clipping algorithm

However, the test runs clearly indicate that even the parallel version, using eight threads, is no rival for the Adaptive Clipping algorithm. Figure 5.3 illustrates the comparison results of 40 test runs between the parallel version of the proposed algorithm, left bar, and the Adaptive Clipping algorithm, right bar. For the proposed algorithm the figure indicates free space graph building time in grey and match search time in white. This distinction reveals that the free space graph building time is still the main factor. Additionally, in only 71% of the test runs the match search time was faster than the whole Adaptive Clipping algorithm. Therefore, in the remaining 29% an improvement in the building phase would not be enough to surpass the Adaptive Clipping algorithim. Assuming that one could extrapolate the improvement factor of parallelization in a linear fashion, one would need at least 40 cores such that in some cases the proposed algorithm runs faster.

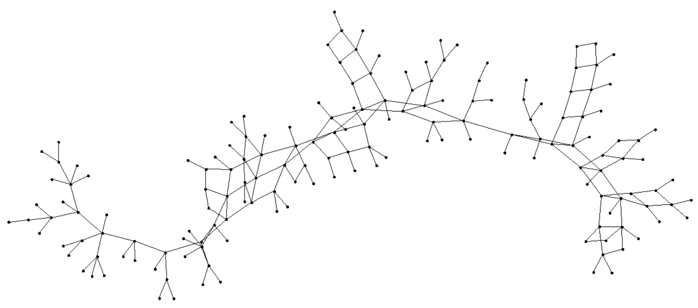

Figure 5.4: Free space graph build by the Adaptive Clipping implementation

The most important reason for this difference in running time can now easily be located. The advantage of the Adaptive Clipping algorithm is that it only calculates those parts of the free space graph that it actually traverses. In contrast, the proposed algorithm has to calculate the whole graph prior to the match search phase. Figure 5.4 shows the graph calculated while running the example match of chapter 4. The difference between this graph and the one build by the proposed algorithm, see figure 4.5, is conspicuous. Even in this small example the difference in graph size are at a factor of three.

In conclusion, this evaluation provides evidence that Adaptive Clipping is the fastest weak Fréchet based algorithm available to solve the map-matching problem for most cases. Yet, the proposed algorithm may outperform the Adaptive Clipping algorithm by utilizing heavy parallelization. It is important to note that this statement is based on the assumption that the parallelization scales as presumed.

5.3 Match Quality

Although the match quality is no focus of this thesis, this section will shortly discuss one quality issue, as it is closely related to the approach described in this thesis. The proposed algorithm guarantees to produce a match curve within the map graph such that it has minimal weak Fréchet distance to the supplied track. However, there can be more than one curve fulfilling this requirements leading to multiple possible matches.

An issue arises due to the fact that the map may not only contain one but many curves that are of minimal weak Fréchet distance to the track. Due to the definition of the weak Fréchet metric and the algorithms approach of match search, the solution space is defined by all transitions added until the last free space graph transition that is

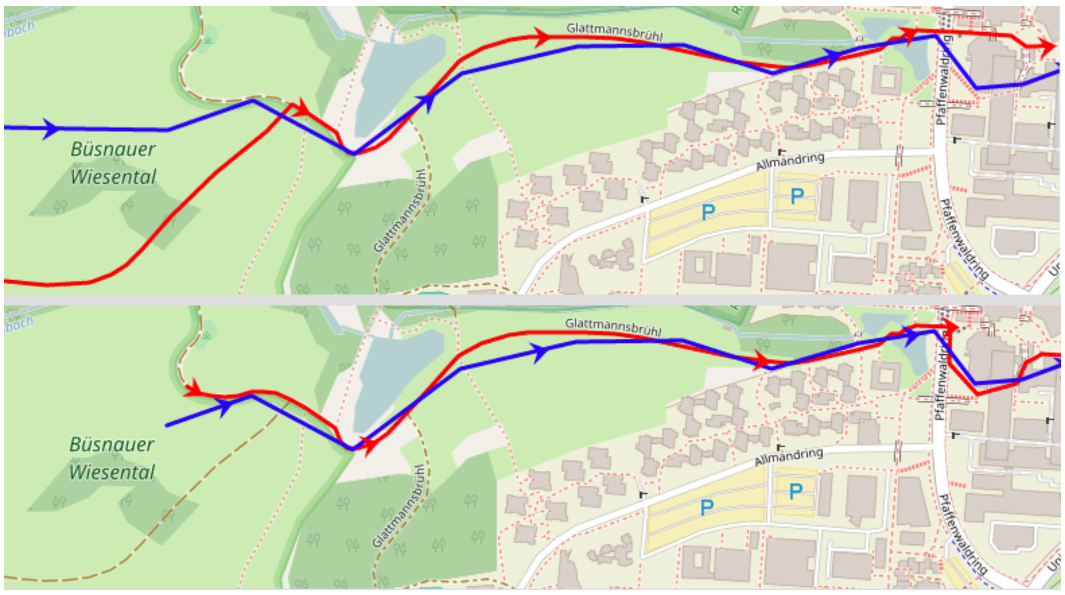

Figure 5.5: Example matches

necessary to form a connection between any start and end vertex that has been added. Most of the time the graph formed by these transitions will contain more than one path from a start to an end vertex and, thus, more than one valid solution.

Figure 5.5 tries to illustrate this issue by showing two pairs of tracks and their matches. The track and match pair on the top exhibits the situation that a part of the track has higher distance to any edge of the map and, therefore, influences the solution space by allowing more transitions to be added before the match search terminates. On the far left of this match picture, one could see that the track given in blue has a big distance to any map edge, as it runs across the Büsnauer Wiesental. As a consequence, more free space graph transitions are added allowing an inferior match path, as can be seen on the far right of the picture. The bottom track and match pair illustrates what happens if one removes the track part around the Büsnauer Wiesental, hence, removing the need to add free space graph transitions up to this outlier in track-map distance. Thus, this track can be matched resulting in a superior match path, as can be seen on the far right. This behaviour can be observed due to the fact that the final match path is created by applying breadth-first search to the solution space.

6 Summary and Future Work

In this final chapter a summary of this thesis is provided, including the goals, methods, approaches, and results. Conclusions are drawn based on the results and experiences made while working on this thesis. Furthermore, subjects for future work are described.

The goal of this thesis was to develop a geometric map-matching algorithm based on the weak Fréchet metric. The main objective was to deliver a faster algorithm compared to the known approaches. The developed algorithm should also work in an incremental way, such that it supports parallelization. These goals have been evaluated by analysing the time complexity and an experimental evaluation which compared the developed algorithm with a map-matching algorithm proposed by Wenk et al. [WSP06].

The first task was to understand and describe the building blocks needed to develop and implement a map-matching algorithm based on the weak Fréchet metric. Sections 2.1 and 2.2 are describing the general map-matching problem and the Fréchet metrics including the concept of free space diagrams. These diagrams are the basic idea of the free space surface and, hence, the free space graph, which is the core structure of the proposed algorithm. Basic subjects needed to implement a geometric map-matching algorithm are the geometric models to represent the earth's shape, coordinate systems, and calculations within these models. The spherical model was chosen and explained together with its coordinate systems and calculations in section 2.3. All positional data used in this thesis, such as trajectories and map data of the OpenStreetMap project, used the latitude and longitude coordinate system, while all calculations were implemented using the vector based Earth-Centered Earth-Fixed system. The main types of data structures needed for the algorithm were examined in section 2.4. These are the adjacency list graph structure for the map representation, priority queues for the fast retrieval of smallest transitions in the free space graph, and union find structures to detect connections in the free space graph.

In order to understand the state of map-matching approaches some related papers were examined in chapter 3. The paper "On Map-Matching Vehicle Tracking Data" by Brakatsoulas et al. [BPSW05] introduced the free space surface, which was also a main part of the subject of this thesis. Furthermore, Wenk et al. proposed the Adaptive Clipping algorithm in [WSP06], which solves the map-matching problem by applying a version of Dijkstra's shortest-path algorithm. This paper can be seen as a solution to some initial goals of this thesis, as it solves the weak Fréchet map-matching without using parametric search, but is not suitable for parallelization. Hence, this algorithm was the best candidate to compare against the algorithm developed in this thesis, which was done in the experimental evaluation.

The developed algorithm, described in chapter 4, processes trajectories in three phases. The first phase extracts a map part such that it contains enough information but not too much. This is done by calculating bounding boxes of a given size around every point in the trajectory and extracting the contained map part from the whole map. Secondly, the algorithm builds up the free space graph. As mentioned earlier, this structure is based on the free space surface, which has its origins in the free space diagram. Every vertex of the free space graph represents a combination of one trajectory and one map edge. An edge in the free space graph is annotated with the minimal distance between the corresponding trajectory and map edges of the connected vertices. Edges are introduced along the trajectory and map axis. This graph is stored by putting all edges into a priority queue such that it is possible to successively retrieve the edges with minimal annotation value. Thus, the third phase is able to construct a match by adding edges, retrieved from the queue, to a union find structure until a path from the lower left to the upper right of the free space graph and, therefore, free space surface is detected. The last edge added to the union find structure, therefore, contains the minimal distance such that there is free space from the lower left to the upper right in the free space surface. In turn, the sub graph removed from the queue contains at least one curve and all contained curves are of minimal weak Fréchet distance regarding trajectory and map by definition of the free space surface. The resulting match is then produced by traversing the sub graph from trajectory start to end in breadth-first search style and extracting the map edges. The analysis of the algorithm showed a time complexity of $\mathcal{O}(mn \log mn)$.

As mentioned before, the evaluation was done by comparing the developed algorithm with the Adaptive Clipping algorithm proposed by Wenk et al. [WSP06]. The comparison was made using 80 trajectories of four different sizes. The results clearly indicated that the Adaptive Clipping algorithm outperformed the non-parallel version of algorithm proposed by this thesis. The main factor for this negative result is the difference in free space graph size that has to be calculated. In contrast to calculating the whole free space graph, the Adaptive Clipping algorithm only calculates parts of this graph, as needed by the Dijkstra style traversal, leading to mature running time differences. Even the parallel version, using eight processing cores, could not surpass the Adaptive Clipping algorithm. However, one could further reduce the running time by adding more cores. Thus, it is possible that the proposed algorithm undercuts the running time of the Adaptive Clipping algorithm under heavy parallelization.

More work could be made to improve the developed algorithm. The map extraction phase has a lot of potential for improvements. For example the bounding box sizes could be calculated dynamically from track and map properties, which may leads to smaller free space graphs and, therefore, faster computations. Another approach would be to use some kind of expanding search to retrieve the needed map edges. Additionally, one could try to push parallelization to the match search phase by using multiple disjoint-set instances. The current version and further improvements should be tested extensively in order to verify the results of this thesis, especially the parallelization scaling assumption. However, due to the imperfections in both algorithms, being sequentiality for the Adaptive Clipping algorithm and free space building times for the proposed algorithm, one might also want to allocate future work on searching for a combination of both algorithms in order to eliminate these issues.

List of Figures

List of Algorithms

Bibliography

[AG95] H. Alt, M. Godau. “Computing the Fréchet distance between two polygonal curves.” In: *International Journal of Computational Geometry & Applications* 5.01n02 (1995), pp. 75–91 (cit. on pp. 13, 15).

[BPSW05] S. Brakatsoulas, D. Pfoser, R. Salas, C. Wenk. “On map-matching vehicle tracking data.” In: *Proceedings of the 31st international conference on Very large data bases*. VLDB Endowment. 2005, pp. 853–864 (cit. on pp. 14, 15, 29–31, 57).

[BR08] J. Bloit, X. Rodet. “Short-time Viterbi for online HMM decoding: Evaluation on a real-time phone recognition task.” In: *2008 IEEE International Conference on Acoustics, Speech and Signal Processing*. IEEE. 2008, pp. 2121–2124 (cit. on p. 28).

[Col87] R. Cole. “Slowing Down Sorting Networks to Obtain faster Sorting Algorithms.” In: *Journal of the Association for Computing Machinery* 34.1 (1987), pp. 200–208 (cit. on p. 15).

[Cor09] T. Cormen. *Introduction to Algorithms*. MIT Press, 2009 (cit. on p. 21).

[DKR13] V. Diekert, M. Kufleitner, G. Rosenberger. *Elemente der diskreten Mathematik: Zahlen und Zählen, Graphen und Verbände*. De Gruyter Studium. De Gruyter, 2013 (cit. on p. 46).

[DMS14] M. Dietzfelbinger, K. Mehlhorn, P. Sanders. *Algorithmen und Datenstrukturen: Die Grundwerkzeuge*. eXamen.press. Springer Berlin Heidelberg, 2014 (cit. on pp. 19, 20, 22, 47).

[EFH+11] J. Eisner, S. Funke, A. Herbst, A. Spillner, S. Storandt. “Algorithms for Matching and Predicting Trajectories.” In: *ALENEX*. SIAM. 2011, pp. 84–95 (cit. on p. 11).

[Fos16] D. Foster. *GPX: the GPS Exchange Format*. 2016. URL: http://www.topografix.com/gpx.asp (visited on 07/04/2016) (cit. on p. 50).

[Fou16a] O. Foundation. *About OpenStreetMap*. 2016. URL: https://wiki.openstreetmap.org/wiki/About_OpenStreetMap (visited on 07/04/2016) (cit. on p. 23).

[Fou16b] O. Foundation. *Copyright and License*. 2016. URL: https://www.openstreetmap.org/copyright/en (visited on 07/04/2016) (cit. on p. 24).

[Fou16c] O. Foundation. *PBF Format*. 2016. URL: http://wiki.openstreetmap.org/wiki/PBF_Format (visited on 07/04/2016) (cit. on p. 49).

[Fré06] M. M. Fréchet. “Sur quelques points du calcul fonctionnel.” In: *Rendiconti del Circolo Matematico di Palermo (1884-1940)* 22.1 (1906), pp. 1–72 (cit. on p. 13).

[Gal06] S. Galati. *Geographic Information Systems Demystified*. Artech House communications library. Artech House, 2006 (cit. on pp. 16, 17).

[GDM+12] C. Goh, J. Dauwels, N. Mitrovic, M. Asif, A. Oran, P. Jaillet. “Online map-matching based on hidden markov model for real-time traffic sensing applications.” In: *2012 15th International IEEE Conference on Intelligent Transportation Systems*. IEEE. 2012, pp. 776–781 (cit. on pp. 7, 28).

[GH72] D. Greenwell, R. L. Hemminger. “Forbidden subgraphs for graphs with planar line graphs.” In: *Discrete Mathematics* 2.1 (1972), pp. 31–34 (cit. on p. 45).

[Jon14] C. B. Jones. *Geographical information systems and computer cartography*. Routledge, 2014 (cit. on p. 16).

[Kar16] G. G. Karlsruhe. *OpenStreetMap Data Extracts*. 2016. URL: http://download.geofabrik.de (visited on 07/04/2016) (cit. on pp. 24, 50).

[KH05] E. Kaplan, C. Hegarty. *Understanding GPS: Principles and Applications*. Artech House, 2005 (cit. on pp. 16, 17).

[Knu98] D. Knuth. *The art of computer programming: Sorting and searching*. The Art of Computer Programming. Addison-Wesley, 1998 (cit. on p. 20).

[LRT15] A. Leick, L. Rapoport, D. Tatarnikov. *GPS Satellite Surveying*. Wiley, 2015 (cit. on p. 12).

[NC16] B. Naveh, Contributors. *JGraphT*. 2016. URL: http://jgrapht.org/ (visited on 07/04/2016) (cit. on p. 49).

[Oc16] OpenStreetMap, contributors. *planet.openstreetmap.org - gps*. 2016. URL: http://planet.openstreetmap.org/gps/ (visited on 07/04/2016) (cit. on p. 50).

[OW11] T. Ottmann, P. Widmayer. *Algorithmen und Datenstrukturen*. Spektrum Akademischer Verlag, 2011 (cit. on pp. 19–23).

[PJ99] D. Pfoser, C. S. Jensen. “Capturing the uncertainty of moving-object representations.” In: *Advances in Spatial Databases*. Springer. 1999, pp. 111–131 (cit. on p. 12).

[QON07] M. A. Quddus, W. Y. Ochieng, R. B. Noland. “Current map-matching algorithms for transport applications: State-of-the art and future research directions.” In: *Transportation research part c: Emerging technologies* 15.5 (2007), pp. 312–328 (cit. on pp. 7, 25).

[Tar83] R. Tarjan. *Data Structures and Network Algorithms*. CBMS-NSF Regional Conference Series in Applied Mathematics. Society for Industrial and Applied Mathematics, 1983 (cit. on p. 20).

[TM12] W. Torge, J. Müller. *Geodesy*. Walter de Gruyter, 2012 (cit. on p. 17).

[Ven16] C. Veness. *Calculate distance, bearing and more between Latitude/Longitude points*. 2016. URL: http://www.movable-type.co.uk/scripts/latlong.html (visited on 07/04/2016) (cit. on pp. 37, 38).

[Wei16] E. W. Weisstein. *Great Circle*. 2016. URL: http://mathworld.wolfram.com/GreatCircle.html (visited on 07/04/2016) (cit. on p. 17).

[WSP06] C. Wenk, R. Salas, D. Pfoser. “Addressing the need for map-matching speed: Localizing global curve-matching algorithms.” In: *Scientific and Statistical Database Management, 2006. 18th International Conference on*. IEEE. 2006, pp. 379–388 (cit. on pp. 3, 25, 32, 37, 49, 50, 57, 58).

[YZZ+10] J. Yuan, Y. Zheng, C. Zhang, X. Xie, G.-Z. Sun. “An interactive-voting based map matching algorithm.” In: *Proceedings of the 2010 Eleventh International Conference on Mobile Data Management*. IEEE Computer Society. 2010, pp. 43–52 (cit. on pp. 8, 26, 27).

[ZCXM09] Y. Zheng, Y. Chen, X. Xie, W.-Y. Ma. “GeoLife2. 0: a location-based social networking service.” In: *2009 Tenth International Conference on Mobile Data Management: Systems, Services and Middleware*. IEEE. 2009, pp. 357–358 (cit. on p. 27).

All links were last followed on November 10, 2016.

Declaration