

# Detecting Commuting Patterns by Clustering Subtrajectories\*

K. Buchin<sup>†</sup>   M. Buchin<sup>†</sup>   J. Gudmundsson<sup>‡</sup>   J. Luo<sup>†</sup>   M. Löffler<sup>†</sup>

## Abstract

In this paper we consider the problem of detecting *commuting patterns* in a trajectory. For this we search for similar subtrajectories. To measure spatial similarity we choose the Fréchet distance and the discrete Fréchet distance between subtrajectories, which are invariant under differences in speed. We give several approximation algorithms, and also show that the problem of finding the ‘longest’ subtrajectory cluster is as hard as MaxClique to compute and approximate.

## 1 Introduction

Technological advances of location-aware devices, surveillance systems and electronic transaction networks produce more and more opportunities to trace moving entities. Consequently, a variety of disciplines including geography, market research, data base research, animal behavior research, surveillance, security and transport analysis shows an increasing interest in movement patterns of entities moving in various spaces over various times scales [11]. In the case of moving animals, movement patterns can be viewed as the spatio-temporal expression of behaviors, e.g. flocking sheep or the seasonal migration of birds. In a transportation context, a movement pattern could be a traffic jam or a commuting route.

In this paper we will focus on the problem of detecting *commuting patterns* in a trajectory. For this we search for similar subtrajectories. To measure spatial similarity we choose the Fréchet distance and the discrete Fréchet distance between subtrajectories. Both of these are invariant under differences in speed: for instance, in a transportation context, this allows to detect a commuting pattern even in the presence of different traffic conditions and varying means of transport. We will also consider how to detect a common movement pattern of a group of entities. That is, we want to find similar subtrajectories in a given set of trajectories. This problems can be handled using the same approach as for one entity. We will focus on the problem for one entity, but also discuss the changes for a group of entities. Figure 1 shows examples for the two different problem statements, i.e., similar subtrajectories of a single trajectory and of several trajectories.

More formally, the input is a moving point object, called *entity*, whose location is known at  $n$  consecutive time-steps. Thus, the trajectory of an entity is a polygonal line that can self-intersect. We assume that an entity moves between two consecutive time steps on a straight line. Given the trajectory  $T$  of a moving entity in the plane, an integer  $m > 0$  and two positive real values  $\ell$  and  $d$ , we define a commuting pattern as a set of  $m$  subtrajectories of  $T$ , where the time intervals of two subtrajectories overlap in at most a point, the subtrajectories are within distance  $d$  from each other, and at least one subtrajectory has length  $\ell$  between its first and last vertex. See Figure 1 for an example. As mentioned above, we will use the Fréchet distance and the discrete Fréchet distance as distance measures.

---

\*This research was initiated during the GADGET Workshop on Geometric Algorithms and Spatial Data Mining, funded by the Netherlands Organisation for Scientific Research (NWO) under BRICKS/FOCUS grant number 642.065.503. The research was further supported by NWO through the GADGET and GOGO projects, and by the Australian Government’s Backing Australia’s Ability initiative, in part through the Australian Research Council.

<sup>†</sup>Utrecht University, the Netherlands. {buchin,maike,ljroger,loffler}@cs.uu.nl

<sup>‡</sup>NICTA, Sydney, Australia. joachim.gudmundsson@nicta.com.au



Figure 1: (a) The two subtrajectories within the shaded region form a subtrajectory cluster for  $m = 2$  if the length of the longest subtrajectory is longer than  $\ell$  and the Fréchet distance between the two subtrajectories is at most  $d$ . (b) Illustrating a subtrajectory cluster for  $m = 3$  in the case when a set of trajectories is given as input.

Recently there has been considerable research in the area of analysing and modelling spatio-temporal data [9]. In the database community, research has mainly focussed on indexing databases so that basic spatio-temporal queries concerning the data can be answered efficiently. Typical queries are spatio-temporal range queries, spatial or temporal nearest neighbours, see for example the work by Sältenis *et al.* [18] and Hadjieleftheriou *et al.* [12]. From a data mining perspective, Verhein and Chawla [19] used association rule mining to detect patterns in spatio-temporal sets. They defined a region to be a *source*, *sink* or a *thoroughfare* depending on the number of objects entering, exiting or passing through the region. Recently there have been several papers considering the problem of detecting flock patterns, leadership patterns and convergence patterns [4, 8].

Vlachos *et al.* [20] state that in the area of spatio-temporal analysis it is an important task to detect commuting patterns of a single entity, or to find objects that moved in a similar way. Thus, an efficient clustering algorithm for trajectories, or subtrajectories, is essential for such data analysis tasks. Gaffney *et al.* [7] proposed a model-based clustering algorithm for trajectories. In this algorithm, a set of trajectories is represented using a regression mixture model. Specifically, their algorithm is used to determine the cluster memberships and it only clusters whole trajectories. Lee *et al.* [17] argued that clustering trajectories as a whole could not detect similar portions of the trajectories. They instead suggested an approach that partitions a trajectory into a set of line segments and then group similar line segments. The obvious drawback is that only segments are clustered, while a commuting pattern might be a complicated path whose shape may not be captured by a single segment. Mamoulis *et al.* [15] used a different approach to detect periodic patterns. They assumed that the trajectories are given as a sequence of spatial regions, for example  $ABC$  would denote that the entity started at region  $A$  and then moved to region  $C$  via region  $B$ . Using this model data mining tools such as association rule mining can be used.

Vlachos *et al.* [20] also looked at discovering similar trajectories of moving objects. They mainly focussed on formalising a similarity function based on the longest common subsequence which they claim is very robust to noise. However, their approach matches the vertices along the trajectories which requires that the vertices along the trajectories are synchronised, or almost synchronised. Since it is not unusual that the coordinates are recorded with a frequency of five per second it has recently been argued [10] that this is an unreasonable assumption since trajectories will have to be compressed (simplified) to allow for fast computations. In this paper we will make no such assumptions on the input data.

The paper is organised as follows. In the next section we introduce the Fréchet distance and formally define the problem considered in this paper. In Section 3 we prove that the longest subtrajectory cluster problem is as hard as MAXCLIQUE to approximate. We therefore consider approximation algorithms in Section 4.

## 2 Preliminaries

In this paper we will present algorithms that find clusters of subtrajectories of a given trajectory. The idea is to select a reference subtrajectory and then to find all subtrajectories that are close to this using the Fréchet distance. The proposed algorithm can be extended to handle subtrajectory clustering in the case when the input is a set of trajectories and the aim is to find the longest sub-

trajectory cluster where each subtrajectory in the cluster is a subtrajectory of an input trajectory and no two trajectories in the cluster belong to the same input trajectory.

As mentioned in earlier work [4, 8], specifying exactly which of the patterns should be reported is often a subject for discussion. In this paper we consider the problems of finding the longest cluster of a fixed size and the largest cluster of a fixed length. We say that a cluster  $C$  of (sub)trajectories has length at least  $\ell$  if there exists a (sub)trajectory in  $C$  whose length between the first and last vertex is at least  $\ell$ .

**Definition 1** *Given a trajectory  $T$  with  $n$  vertices, a subtrajectory cluster  $C_T(m, \ell, d)$  for  $T$  of length  $\ell$  consists of at least  $m$  non-identical subtrajectories  $T_1, \dots, T_m$  of  $T$  such that the time intervals for two subtrajectories overlap in at most a point, the distance between the subtrajectories is at most  $d$ , and at least one subtrajectory has length  $\ell$ .*

**Problem 1** (SUBTRAJECTORY CLUSTER —  $\text{SC}(m, \ell, d)$ )

*Given a trajectory  $T$ , the subtrajectory cluster problem  $\text{SC}(m, \ell, d)$  is the decision problem: does there exist a subtrajectory cluster with these parameters? We also define the related optimisation problems  $\text{SC}(\max, \ell, d)$ ,  $\text{SC}(m, \max, d)$ , and  $\text{SC}(m, \ell, \min)$ , which keep two parameters fixed and aim to maximise or minimise the third.*

A variant of this problem is to find a subtrajectory cluster in a given set of trajectories, as shown in Figure 1(b). This variant has two further subvariants: for each trajectory at most one subtrajectory is allowed, or several. By concatenating the given set of trajectories to one trajectory, any instance of this problem can be made into an instance of the subtrajectory cluster problem. For the subvariant where only one subtrajectory of each trajectory is allowed, this restriction has to be enforced, as well.

In Definition 1 we omitted to define the distance metric. In previous papers several different distance functions and approaches have been used, for example the Longest Common Subsequence model [20], a combination of parallel distance, perpendicular distance and angle distance [17], the count of common subsequences of length two [1], the domain-dependent extraction of a single representative value for the whole series [14], the average Euclidean distances between paths [16].

The Fréchet distance is a distance measure for continuous shapes such as curves and surfaces, and is defined using reparameterisations of the shapes. Since it takes the continuity of the shapes into account it is generally regarded as being a more appropriate distance measure than the Hausdorff distance for curves [3]. For polygonal curves, the discrete Fréchet distance is a natural variant of the Fréchet distance. In this paper we will use the Fréchet distance [2] and discrete Fréchet distance [6]. For a set of  $m > 2$  curves there is a natural extension due to Dumitrescu and Rote [5]. We will denote the Fréchet distance between to curves  $f$  and  $g$  by  $\delta_F(f, g)$ , and the discrete Fréchet distance by  $\delta_{dF}(f, g)$ . For completeness, we include the formal definitions of these distance measures in Appendix A.

The study of the Fréchet distance from a computational point of view was initiated by Alt and Godau [2]. They showed that the Fréchet distance between two curves with  $m$  and  $n$  segments respectively can be computed in  $O(mn \log mn)$  time, and the associated decision question (is the Fréchet distance more than  $d$ ?) can be answered in  $O(mn)$  time. The study of the discrete Fréchet distance was started by Eiter and Mannila [6]. They gave a simple dynamic programming algorithm for computing the discrete Fréchet distance in  $O(mn)$  time. Furthermore, they showed that  $\delta_F(f, g) \leq \delta_{dF}(f, g) \leq \delta_F(f, g) + L/2$ , where  $L$  is the length of the longest segment of  $f$  or  $g$ . This implies that under sufficient sampling, the discrete Fréchet distance is an arbitrary good approximation of the Fréchet distance. Sometimes we will use the term continuous Fréchet distance to contrast the Fréchet distance to the discrete Fréchet distance.

In this paper we will focus on the optimisation version of Problem 1 where the length  $\ell$  is maximised, i.e., the longest subtrajectory cluster  $\text{SC}(m, \max, d)$  for fixed parameters  $m$  and  $d$ . We will prove in the next section that the general decision problem  $\text{SC}(m, \ell, d)$  is NP-complete, so the optimisation variants are also NP-hard. Therefore, we will turn our attention to approximation algorithms. We speak of a *size* approximation when we approximate  $m$ , a *length* approximation

when we approximate  $\ell$ , and a *distance* approximation when we approximate  $d$ . For instance, assume an optimal solution of  $\text{SC}(m, \max, d)$  has length  $\ell^*$ . A  $c$ -distance approximation algorithm for this problem would return a cluster  $C_T(m, \ell^*, cd)$  of length at least  $\ell^*$  and with  $m$  subtrajectories of  $T$  within Fréchet distance  $c \cdot d$  from each other. We will also briefly discuss the case where the size of the cluster is maximised, i.e., the problem  $\text{SC}(\max, \ell, d)$  for fixed  $\ell, d$ .

In the following, for an instance of the SC problem we will always use  $n$  to denote the number of vertices in the given trajectory,  $\ell$  for the length of the longest subtrajectory cluster,  $m$  for the size of the cluster, and  $d$  for the distance between two curves in the cluster. If the input is a set of trajectories, we will use  $k$  for the number of trajectories and  $n$  for the maximum length of a single trajectory.

### 3 Hardness results

In this section we give several hardness results. We will prove that the general decision problem  $\text{SC}(m, \ell, d)$  is NP-complete, so the optimisation variants are also NP-hard and we will have to turn our attention towards approximation algorithms. We will also prove some hardness results for several approximation problems. First we prove that any distance approximation of SC is 3SUM-hard. Then we show that the general problem is NP-complete, and prove that the problems of finding a subtrajectory cluster containing a maximum number of subtrajectories or having maximum length are NP-hard to approximate within a factor of  $n^{1-\varepsilon}$  for any positive constant  $\varepsilon$ , even if we allow a factor  $2 - \phi$  distance approximation. This also motivates why we study 2-distance approximation algorithms in the following sections.

The results in this section follow from similar results in [8]. We give the proofs in Appendix B.

**Lemma 1** *Finding any distance approximation of the  $\text{SC}(m, \max, d)$  problem is 3SUM-hard.*

**Theorem 1** *The decision problem  $\text{SC}(m, \ell, d)$  is NP-complete.*

The proof of Theorem 1 holds for any Fréchet distance in  $[1, 2)$ . Hence, even if we allow an approximate distance with a factor less than 2, the problem is still NP-hard.

**Corollary 1** *The problem of computing a  $(2 - \phi)$ -distance approximation of  $\text{SC}(\max, \ell, d)$ , for any  $0 < \phi \leq 1$ , is NP-hard.*

The above result can easily be strengthened by noting that an  $\alpha$ -approximation of the size of  $\text{SC}(\max, \ell, d)$  also corresponds to an  $\alpha$ -approximation of MAXCLIQUE, that is, we cannot hope to find a SC of approximately the maximum size efficiently. We restate from [13]:

**Fact 1** (*Håstad 1999*)

*For any constant  $\varepsilon > 0$ , MAXCLIQUE cannot be approximated in polynomial time within a factor of  $n^{1/2-\varepsilon}$  unless  $P = NP$ , and not within a factor of  $n^{1-\varepsilon}$  unless  $NP = ZPP$ .*

**Corollary 2** *The problem of computing a  $n^{1/2-\varepsilon}$ -size,  $(2 - \phi)$ -distance approximation of the  $\text{SC}(\max, \ell, d)$  problem, for any  $\phi, \varepsilon > 0$ , is NP-hard.*

However, the problem we will focus on in this paper is to find the longest subtrajectory cluster. A similar result as in the above corollary can be obtained for this problem in the following way. We build the same construction as in the proof of Theorem 1 (in the appendix) for an instance of MAXCLIQUE with  $N$  vertices. Note that the number of trajectories in such a construction is  $N$  and the length of the trajectories is also  $N$ . Let  $0 < \varepsilon < 1$  be a positive real constant and set  $\tau = N^{1/\varepsilon}$ . Connect  $\tau^{1-\varepsilon}$  copies of the constructions which gives an instance with  $N$  trajectories each of length  $\tau$ . This instance contains a subtrajectory cluster of length  $\tau$  and size  $N$  if and only if each copy contains a subtrajectory cluster of size and length  $N$  and this exists if and only if a MAXCLIQUE of size  $N$  exists. However this is NP-complete, and hence, computing a longest subtrajectory cluster within an approximation factor of  $\tau/N = \tau^{1-\varepsilon}$  is NP-hard. We have:

**Corollary 3** *The problem of computing a  $\tau^{1-\varepsilon}$ -length,  $(2 - \phi)$ -distance approximation of the  $\text{SC}(m, \max, d)$  problem, for any  $\phi, \varepsilon > 0$ , is NP-hard.*

## 4 A 2-distance approximation using the Fréchet distance

First we briefly outline the *free space diagram* of two polygonal curves  $f$  and  $g$ , which is a geometric data structure introduced by Alt and Godau [2] for computing the Fréchet distance. Let  $f$  be a polygonal curve with  $n$  vertices  $p_1, \dots, p_n$ . We use  $\phi_f$  to denote the following natural parameterisation of  $f$ . The map  $\phi_f: [1, n] \rightarrow \mathbb{R}^c$  maps  $i \in \{1, \dots, n\}$  to  $p_i$  and interpolates linearly in between vertices, where  $c$  is any dimension. The free space diagram of two polygonal curves  $f$  and  $g$ , with  $n$  and  $m$  vertices, respectively, is the set

$$F_d(f, g) = \{(s, t) \in [1, n] \times [1, m] : |\phi_f(s), \phi_g(t)| \leq d\}$$

which describes all tuples  $(s, t)$  such that the points  $\phi_f(s)$  and  $\phi_g(t)$  have Euclidean distance at most  $d$ . See Figure 2 for an example of a free space diagram. The complexity of the free space diagram is  $O(nm)$  and it can be constructed in time  $O(nm)$ . Alt and Godau [2] showed that the Fréchet distance between  $f$  and  $g$  is less than  $d$  if and only if there exists a monotone path in the free space diagram  $F_d(f, g)$  from  $(0, 0)$  to  $(n, m)$ .

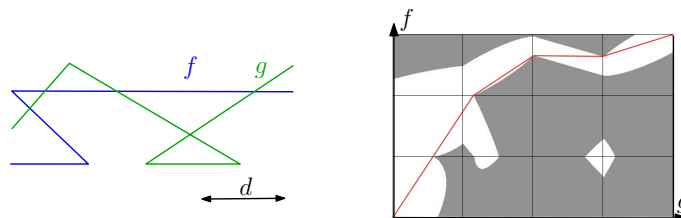


Figure 2: Polygonal curves  $f$  and  $g$ , distance value  $d$ , and a monotone path in the free space diagram.

For the discrete Fréchet distance the free space diagram consists of the  $nm$  grid points. Each grid point  $(x, y)$  is labeled free or not depending on whether  $|\phi_f(x) - \phi_g(y)|$  is less than or equal to the distance  $d$  or not. A monotone path is a sequence of free grid points from  $(0, 0)$  to  $(n, m)$  where a grid point  $(x, y)$  is followed either by  $(x + 1, y)$ ,  $(x, y + 1)$  or  $(x + 1, y + 1)$  [6]. Analogous to the continuous Fréchet distance, it holds that the discrete Fréchet distance is less than  $d$  if and only if there is a monotone path in the free space diagram.

Consider a set of  $m$  polygonal curves  $\mathcal{F} = \{f_1, \dots, f_m\}$ , where each curve  $f_i$  has  $n_i$  segments ( $i = 1, \dots, m$ ). The Fréchet distance of the set  $\mathcal{F}$  can be computed in time  $O((n_1 \cdot \dots \cdot n_m) \log(n_1 \cdot \dots \cdot n_m))$ . Even for small values of  $m$  this is deemed impractical. However, Dumitrescu and Rote [5] showed that a 2-approximation of the distance can be obtained by computing all the pairwise Fréchet distances and outputting  $\min_{1 \leq i \leq m} \max_{1 \leq j < k \leq m} (\delta_F(f_i, f_j) + \delta_F(f_i, f_k))$ , which only requires time  $O(\sum_{1 \leq i < j \leq m} n_i n_j \log n_i n_j)$ . For the discrete Fréchet distance the same approximation holds which can be computed in time  $O(\sum_{1 \leq i < j \leq m} n_i n_j)$ .

**Cluster Curves in the Free Space.** In our application, we have just one polygonal curve: this input trajectory  $T$ . Now consider the free space diagram  $F_d(T, T)$  of two copies of  $T$ , and distance value  $d$ . Let  $s, t \in [1, n]$  be two points of time for the trajectory  $T$ , and let  $l_s$  and  $l_t$  be the two vertical lines in  $F_d(T, T)$  corresponding to them. For the discrete Fréchet distance, we assume that  $s, t \in \{1, \dots, n\}$ , i.e.,  $l_s$  and  $l_t$  fall on vertices in  $F_d(T, T)$ . We say that there are  $m$  *cluster curves* between  $l_s$  and  $l_t$  in  $F_d(T, T)$  if and only if there are  $m$  monotonically increasing curves in the free space starting at  $l_s$  and ending at  $l_t$ , such that no two curves contain the same  $y$ -coordinate, except possibly at the endpoints. Figure 3 shows an example, and the corresponding cluster of subtrajectories of  $T$ .

We will make use of the following two implications for which we give proofs in Appendix C.

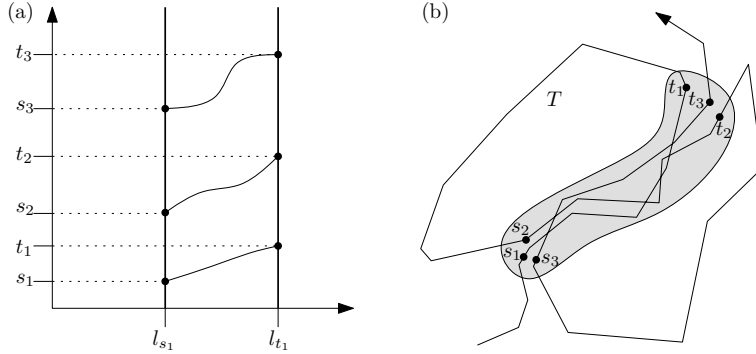


Figure 3: (a) There exists a trajectory cluster with three subtrajectories between  $s$  and  $t$ . A possible trajectory corresponding to the cluster curves is illustrated in (b). Note that the subtrajectory  $s, t$  is one of the subtrajectories.

**Lemma 2** *Let  $T$  be trajectory and  $T'$  the subtrajectory of  $T$  that starts at  $s$  and ends at  $t$ , where  $t - s = \ell$ . Then the following holds:*

- *If there exist  $m$  cluster curves within the free space diagram  $F_d(T, T)$  between  $l_s$  and  $l_t$ , then there exists a subtrajectory cluster  $C_T(m, \ell, 2d)$  containing  $T'$ .*
- *If there exists a subtrajectory cluster  $C_T(m, \ell, d)$  containing  $T'$ , then there exists a set of  $m$  cluster curves within the free space diagram  $F_d(T, T)$  between  $l_s$  and  $l_t$ .*

In the remainder of this section, we develop an algorithm for finding subtrajectory clusters that uses these implications. We give the algorithm first for the discrete Fréchet distance (Section 4.1), and then extend it to the continuous Fréchet distance where either all or only the reference subtrajectories must start and end at vertices of the trajectory (Sections 4.2.1 and 4.2.2), and to the continuous Fréchet distance for arbitrary subtrajectories (Section 4.2.3).

## 4.1 Discrete Fréchet Distance

### 4.1.1 Algorithm

Recall that as input we are given a trajectory  $T$ , an integer  $m$  and a positive real value  $d$ . Consider the free space diagram  $F = F_d(T, T)$  of  $T$ . For the discrete Fréchet distance, this consists only of the at most  $n^2$  grid points which lie in free space, i.e., where the distance between the corresponding two vertices is less than or equal to  $d$ . We sweep two vertical lines in  $F$ :  $l_s$  and  $l_t$ , with  $x$ -coordinates  $s$  and  $t$ , which represent the start point and the end point of the reference trajectory, respectively. Initially  $l_s$  and  $l_t$  are at the leftmost position in  $F$ . We will sweep  $l_s$  and  $l_t$  in discrete steps along the  $x$ -coordinates  $\{1, \dots, n\}$ .

The algorithm proceeds as follows. Move  $l_t$  to the right to the first position for which there are less than  $m$  cluster curves between  $l_s$  and  $l_t$ . Next move  $l_s$  to the right until  $l_s$  equals  $l_t$  or there are at least  $m$  cluster curves between  $l_s$  and  $l_t$ . Then move  $l_t$  again, etc. This continues until  $l_s$  reaches the rightmost position in  $F$ . During the sweep we maintain the longest length of a reference trajectory for which there exist at least  $m$  cluster curves between its endpoints. It is easily seen that the two sweep lines of  $l_s$  and  $l_t$  move monotonically from left to right over  $n$  event points. This implies that only  $O(n)$  possible pairs of start and endpoints are considered.

What remains to be done is to build and update a data structure that allows us to check if there are  $m$  monotone curves between  $l_s$  and  $l_t$  that overlap in at most a point along the  $y$ -axis. For this, we will store the free space between  $l_s$  and  $l_t$  as directed, labeled graph on the vertices in the free space. When updating  $l_t$ , we add a new column of the free space. When updating  $l_s$ , we delete the leftmost column. For checking whether  $m$  curves exist between  $l_s$  and  $l_t$ , we will search this graph.

**Data structure:** We store the free space between  $l_s$  and  $l_t$  as a directed, labeled graph on the vertices in the free space (see Figure 4). We store a directed edge from  $p$  to  $p'$  if  $p, p'$  lie in the free space of the same cell and  $p$  is to the left or above  $p'$ . We label each edge by the smallest  $x$ -coordinate reachable by a path along this edge. This graph can be efficiently computed column-by-column from left to right, and bottom to top within a column. The directed edges encode the free space. The edge labels are easy to compute: An edge pointing to a vertex without outgoing edges is labeled with the  $x$ -coordinate of that vertex. An edge pointing to a vertex  $p$  with outgoing edges is labeled by the smallest label of the outgoing edges of  $p$ . Since a vertex has outdegree at most three, an edge label can be computed in constant time.

**Update:** We can update this graph in  $O(n)$  time: When  $l_t$  is moved one position to the right, we need to compute the new column of the free space which has complexity  $O(n)$ . When  $l_s$  is moved one position to the right, we simply delete the currently leftmost column of the free space.

**Query:** Using our data structure, we can determine whether there are  $m$  cluster curves between  $l_s$  and  $l_t$ . We do this by greedily searching for monotone paths from  $l_t$  to  $l_s$ . Let  $i$  be the  $x$ -coordinate of  $l_s$ . We start at the topmost vertex on  $l_t$  which has an outgoing edge labeled less or equal than  $i$ . In each vertex we follow the topmost edge which again has a label less or equal than  $i$ . This ends on a vertex  $(i, j)$  on  $l_s$ . We continue with the topmost vertex on  $l_t$  at height at most  $j$  which as before has an outgoing edge labeled less or equal than  $i$ . We stop when we have found  $m$  curves, or no more vertices on  $l_t$  with an outgoing edge labeled less or equal than  $i$  exist. When we find a horizontal path in the free space, i.e., a path from  $(i', j)$  on  $l_t$  to  $(i, j)$  on  $l_s$ , we continue with the topmost vertex on  $l_t$  at height at most  $j - 1$  (else we would continue finding the same horizontal path). Note that the edge labels prevent us from going into dead ends in the graph.

Since we need to ensure that the reference subtrajectory is part of the output cluster, must take care that the vertical span  $[s, t]$  is not used in any of the curves. During the query, if the  $y$  coordinate of current vertex gets smaller than  $t$ , then we don't increase the number of cluster curves and we skip down to  $s$  and continue searching for cluster curves from the vertex  $(t, s)$ .

This query takes  $O(n + ml)$  time, because the  $m$  curves will in total consist of at most  $n$  edges directed downward and  $ml$  edges directed to the left (where we count diagonal edges as either down or left directed). There may not be more than  $n$  edges directed downward since the  $m$  curves cannot overlap in more than a point.

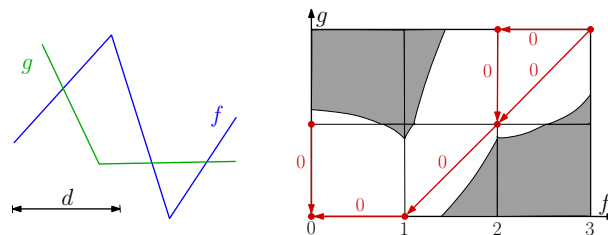


Figure 4: Illustrating the associated data structure.

#### 4.1.2 Analysis

The description of the algorithm immediately gives the following lemma. Note that in practice,  $ml$  should be of order  $n$ .

**Lemma 3** *The data structure of the algorithm of size  $O(\ell n)$  can be maintained in  $O(n)$  time per update such that number-of-cluster-curves queries can be answered in  $O(n + ml)$  time.*

**Theorem 2** *A 2-distance approximation of the  $SC(m, \max, d)$  problem can be computed in  $O(n^2 + nml)$  time and  $O(\ell n)$  space using the discrete Fréchet distance.*

**Proof.** We need to prove both the complexity of the algorithm and its correctness. We start with the complexity.

Since the two sweep lines,  $l_s$  and  $l_t$ , move monotonically from left to right there will be  $2n$  events in total. At each event we perform an update and a query. According to Lemma 3 each update and query can be handled in  $O(n + m\ell)$  time using  $O(\ell n)$  space which implies that the total time and space complexity is  $O(n^2 + nm\ell)$  and  $O(\ell n)$ , respectively. Note that to reduce the space complexity of the algorithm, we do not pre-compute the free space diagram which has  $O(n^2)$  size. Instead we only maintain the part of the free space which we are currently sweeping.

It remains to prove the correctness of the algorithm. Consider an optimal subtrajectory cluster  $C$  containing  $m$  subtrajectories and with Fréchet distance at most  $d$ . Let  $\ell^*$  be the length of  $C$ , i.e., there exists a subtrajectory  $\tau$  in  $C$  such that the distance along  $\tau$  between the first vertex ( $v_i$ ) to the last vertex ( $v_j$ ) on  $\tau$  is  $\ell^*$ .

Consider the process of the algorithm, and consider the point during the sweep when  $l_s$  reaches  $v_i$ . If  $l_t$  has not passed  $v_j$  then we know that there are at least  $m$  cluster curves between  $l_s$  and  $l_t$  thus  $l_t$  will be incremented. Note that this will continue until  $l_t$  pass  $v_j$ , thus the algorithm will test  $l_s = i$  and  $l_t = j$  and report a subtrajectory cluster of length  $\ell^*$ . In the case when  $l_t$  already passed  $v_j$  then there must have been an integer  $i'$  such that  $l_s = i'$  and  $l_t = j$  and  $l_t$  was incremented. Since  $i' < i$  the length of this interval must have been greater than  $\ell^*$ . Furthermore, since  $l_t$  was incremented the number of cluster curves must have been at least  $m$ . So a subtrajectory cluster starting at  $v_{i'}$  and ending at  $v_j$  must have been reported. Consequently the subtrajectory cluster that is returned by the algorithm must have length at least  $\ell^*$  in both cases. This proves the correctness of the algorithm. ■

## 4.2 Fréchet Distance

In this section we extend the algorithm to the continuous Fréchet distance. We first consider the case where subtrajectories in a cluster start and end at vertices and then further extend to arbitrary subtrajectories.

### 4.2.1 All Subtrajectories Start and End at Vertices

We first consider the continuous Fréchet distance with the restriction that all subtrajectories start and end at vertices. In  $F_d(T, T)$  each *cell* corresponds to two line segments of  $T$  and the free space in one cell is the intersection of the cell with an ellipse, possibly degenerated to the space between two parallel lines [2]. There are at most eight intersection points of the boundary of the cell with free space. We call these intersection points *critical points*, see Figure 5. For each critical point, we put a vertex on it. Furthermore, we propagate all critical points in the free space of the corresponding row or column, respectively. That is, we propagate critical points on vertical cell boundaries horizontally, and critical points on horizontal cell boundaries vertically. Figure 5 shows an example of horizontal propagation. In particular, the figure shows an example where a monotone polygonal path is only possible with propagated critical points as vertices (e.g. the one shown by a solid red line).

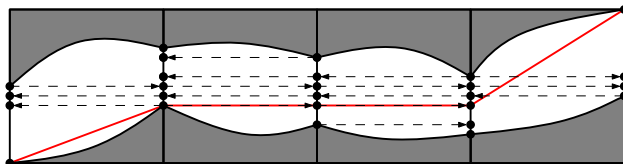


Figure 5: Adding and propagating *critical points* in the free space.

In the following, we use  $R$  to denote the set of propagated critical points and vertices of the free space. Using the propagated critical points and convexity of the free space in one cell we get

**Observation 1** *Let  $v, w$  be two vertices of  $R$ . If there exists a monotonically increasing curve  $P$  from  $v$  to  $w$  in the free space, then there exists a monotonically increasing polygonal path  $P'$  from  $v$  to  $w$  in the free space such that vertices of  $P'$  are vertices of  $R$ .*

By Observation 1 it suffices to compute monotonically increasing polygonal paths using vertices of  $R$ . We can extend the data structure and algorithm for the discrete Fréchet distance to work with vertices of  $R$  instead of only vertices of the free space. The size of  $R$  is  $O(n^3)$  since there are a linear number of critical points in each cell. However, in our algorithm we only store and update the part of the free space between the sweep lines  $l_s$  and  $l_t$ . Furthermore, we will see next that we only need to explicitly compute the critical points on vertical cell boundaries. With this, we will get a space complexity of  $O(\ell^2 n)$  where  $\ell$  is the length of a longest subtrajectory cluster.

For the query we are only interested in monotone paths between critical points on  $l_s$  and  $l_t$ , i.e., only between critical points on vertical cell boundaries. Therefore we do not need to explicitly compute critical points on horizontal cell boundaries. Instead we can do the following. For all but the bottom-most point on a vertical cell boundary, it suffices to have at most a downward and a leftward directed edge. Because of the propagation the diagonal downward edge can be replaced by these two. The bottom-most point on a vertical cell boundary receives at most a leftward and a diagonal downward directed edge. The diagonal downward edge is labeled with two values, one is as before the left-most column index reachable by this edge. The other value stores which lower cells of this free space column are reachable from the point. Note that “reachable” for both edge labels refers to reachability in the graph encoding the free space, i.e., the direction is opposite to the direction of reachability in the free space. More explicitly, consider the bottom-most point on the vertical cell boundary of the vertical line at column index  $i + 1$  in the  $j$ th row. From this point, possibly the free space on vertical cell boundaries at column index  $i$  in rows  $j - 1$  to  $k$  for some  $k \leq j - 1$  are reachable. As second value for the downward edge, we store this value  $k$  or we store  $j$  if no lower cells are reachable. Furthermore, we store for each column index between  $l_s$  and  $l_t$  an array of  $n$  pointers to the topmost points in free space intervals in all rows. Thus, when querying from a bottom-most point on a vertical cell boundary, we can jump to all lower free space cells that it can reach. In this way, we do not need to store any critical points on horizontal cell boundaries. In Appendix D we show that these edge labels can be computed efficiently, that is in linear time per column, during the update of  $l_t$ .

**Time and Space Complexity:** In the update of  $l_t$  we need to compute the new critical points of this column and their edge labels. This takes  $O(n)$  time as described in Appendix D. Furthermore, we also need to propagate critical points horizontally. That is, we need to propagate the  $O(n)$  points of the new row to the at most  $l$  rows to the left, and we need to propagate the  $O(ln)$  points of previous rows to the new column. This, and therefore the update of  $l_t$ , takes  $O(ln)$  time in total. In the update of  $l_s$  we also need to delete critical points propagated from the column being deleted. These are again  $O(ln)$  points.

The query is done as before only on a larger graph. The query time is  $O(n\ell + m\ell) = O(n\ell)$  because the  $m$  cluster curves now consist of at most  $n\ell$  downward and  $m\ell$  leftward directed edges.

The space complexity of the algorithm increases to  $O(\ell^2 n)$  since we now also store all horizontally propagated points. These are at most  $\ell$  points per cell in at most  $\ell n$  cells.

**Theorem 3** *A 2-distance approximation of the  $SC(m, \max, d)$  problem can be computed in  $O(\ell n^2)$  time and  $O(\ell^2 n)$  space using the Fréchet distance in the case that all subtrajectories start and end at vertices of the trajectory.*

#### 4.2.2 Reference Subtrajectory Starts and Ends at Vertices

In this section, we consider the continuous Fréchet distance with the restriction that only the reference subtrajectory has to start and end at vertices. We can use the same algorithm as in the previous Section 4.2.1. The main modification we need is that we perform the query on all points and not only on the vertices of the free space. This does not change the time and space

complexity of the algorithm. The algorithm stays correct, because in fact a stronger version of Observation 1 holds. Namely, let  $P$  be an arbitrary monotone curve in the free space from  $v$  to  $w$  where both  $v, w$  are points on a vertical cell boundary. Then, by the definition of the propagated critical points, there will also be a monotone polygonal curve  $P'$  from the first critical point above  $v$  to the first critical point below  $w$  such that the vertices of  $P'$  are all vertices in  $R$ . Furthermore, the  $y$ -span of  $P'$  is a subset of the  $y$ -span of  $P$ .

We also need the following minor modification to the query operation. We now consider subtrajectories starting and ending at arbitrary points of  $T$  and we have no restriction on the minimum length of a subtrajectory. Therefore a subtrajectory cluster may also consist of a set of points of  $T$ . In particular, a situation as described in Section 6 can occur. We can detect these situations as follows (and prevent our algorithm from running in an infinite loop). When we find a horizontal path in free space, i.e., a subtrajectory consisting of a single point, then we jump to the next critical point below. If from this point we again find a horizontal path, then we know by the convexity of free space cells, that there is a corridor of width larger than 0 in the free space. In this corridor, there are infinitely many horizontal paths, i.e., we have found infinitely many subtrajectories consisting of single points. In particular, the cluster has size at least  $m$  and we can stop the query by returning true.

In total we have the following theorem.

**Theorem 4** *A 2-distance approximation of the  $SC(m, \max, d)$  problem can be computed in  $O(\ell n^2)$  time and  $O(\ell^2 n)$  space using the Fréchet distance in the case when the reference subtrajectory must start and end at vertices of the trajectory.*

### 4.2.3 Arbitrary Subtrajectories

We now consider the case where all subtrajectories (including the reference subtrajectory) may start and end at arbitrary points on the given trajectory. For this, we use the same algorithm as in Section 4.2.2 and extend it to handle the case of the reference subtrajectory not starting and ending at vertices of the trajectory. The algorithm requires two major changes: (1) we add more critical points and event points and (2) we also need to search for optimal solutions inbetween event points. The new critical and event points are points where free space starts or ends within a cell. For the searching inbetween event points we need to do computation on the quadratic curves describing the cell boundaries of a free space cell. To our knowledge, this is the first algorithm explicitly working on the boundaries of free space cells. A complete description of the algorithm can be found in Appendix E. Here we just state the resulting theorem.

**Theorem 5** *A 2-distance approximation of the  $SC(m, \max, d)$  problem can be computed in time  $O(n^3 m 2^{\alpha(n/m)} (\log n \log(n/m) + m))$  using  $O(n\ell + nm 2^{\alpha(n/m)} \log n \log(n/m))$  space using the continuous Fréchet distance for arbitrary subtrajectories. In these bounds,  $n$  denotes the size of the given trajectory and  $\ell$  the length of the longest subtrajectory cluster found.*

## 4.3 Extensions

The algorithms can be modified to handle other settings, without using any extra time or space.

**Subtrajectory Cluster of a Set of Trajectories.** The case when the input is a set  $S$  of trajectories and the aim is to find the longest subtrajectory cluster of  $S$ , allowing either one or several subtrajectories of each trajectory in  $S$ , can be handled by small modifications to the above algorithms.

**Maximising the Number of Cluster Curves.** Above we optimised the length of the cluster, but one might also be interested in maximising the number of subtrajectories in a cluster provided the length of the cluster is fixed. The algorithms can easily be adapted to compute a 2-distance approximation of the  $SC(\max, \ell, d)$  problem, with the same running times as stated above.

Table 1: Result of clustering algorithm given input of a trajectory

Total Points	Standard Mode		Memory Efficient Mode	
	Time (sec)	Memory (MB)	Time (sec)	Memory (MB)
50	0.07	1.14	0.03	0.67
75	0.08	2.59	0.08	0.70
250	0.25	7.51	0.20	0.89
375	0.44	18.93	0.28	1.31
500	0.81	27.78	0.62	1.34
1000	3.59	120.53	1.66	5.66
2500	N/A		13.71	7.30
5000	N/A		88.28	7.32
7500	N/A		750.50	9.15
25000	N/A		2227.08	10.73

## 5 A note on the experiments

To test our ideas we implemented two simplified versions of the algorithm presented in Section 4.1. Both implementations build upon an earlier version of our algorithm which had a running time of  $O(\ell^2 n^2)$ . The first (naïve) approach precomputes the entire freespace and hence uses  $O(n^2)$  space, the second algorithm builds the freespace during the sweep and uses only  $O(\ell n)$  space. The experiments were performed on an Intel Pentium-4 3.0 GHz processor and 1GB of RAM. The clustering algorithms were implemented and compiled using Java SE Development Kit 6 under the Eclipse SDK environment. The test data was generated using a modified version of NetLogo [21] and generated a set of trajectories whose complexity varied from 50 to 25,000. We measured the time required to run different experiments as well as the amount of memory each experiment consumed. Each experiment was performed three times and the number in the table is the average value of the running times.

As can be seen in the table the running time increases rapidly with the complexity of the trajectory. To test the usefulness of our algorithm we tested it on a real trajectory. The trajectory was obtained by carrying a GPS for one month in a  $50 \times 50$  km region and it generated 82,160 time-points. Running the memory efficient algorithm without any optimisation options took approximately 7 hours and used 42 MB of memory. This is not manageable in practice but there are many simple ideas that can be used to improve the running time. A simple and effective improvement is to simplify the trajectory [10]. Using a distance threshold of 10 meters compressed the trajectory to 5,228 time-steps and is done in about 170ms. With the compressed trajectory the memory efficient algorithm computed the clusters in roughly 2 minutes using 2.65 MB of memory where the distance threshold  $d$  was set to 15 meters. The result of the clustering is illustrated in Appendix F. The approach looks promising and we believe the running time can be improved considerably by simple ideas, such as pruning the freespace so that time intervals that obviously not are of any interest are omitted with only minor preprocessing.

## 6 Concluding remarks and Acknowledgments

Our definition of the length of a cluster allows the case of a cluster consisting of one curve of length  $\ell$  and (in the case of arbitrary subtrajectories) infinitely many subtrajectories of length 0, i.e., points. This can occur only when a reference subtrajectory of length  $\ell$  stays in a disk of radius  $d$ . In practice, this is unlikely to occur. An alternative approach could be the length of the shortest curve in the cluster. However, this seems much harder to handle algorithmically. Currently in the sweep we aim at cluster curves with a short  $y$ -span (and long  $x$ -span). For maximising the length of the shortest curve we would need to find cluster curves with large  $x$ - and  $y$ -spans.

The authors would like to thank Matthew Lowry for introducing us to the problem and giving several insightful comments and Cahya Ong for the implementation.

## References

- [1] R. Agrawal, K.-I. Lin, H. S. Sawhney, and K. Shim. Fast similarity search in the presence of noise, scaling, and translation in time-series databases. Proc. 21st Internat. Conference on Very Large Data Bases, pp. 490-501, 1995.
- [2] H. Alt and M. Godau. Computing the Fréchet distance between two polygonal curves. Internat. Journal on Computational Geometry and Applications, 5:75-91, 1995.
- [3] H. Alt and L. Guibas. Discrete geometric shapes: Matching, interpolation, and approximation. In J.-R. Sack and J. Urrutia, editors, Handbook of Computational Geometry, pp. 121-153. Elsevier, 1999.
- [4] M. Benkert, J. Gudmundsson, F. Hübner and T. Wolle. Reporting Flock Patterns. Proc. 14th European Symposium on Algorithms, pp. 660-671, 2006.
- [5] A. Dumitrescu and G. Rote. On the Fréchet distance of a set of curves. Proc. 16th Canadian Conference on Computational Geometry, 2004.
- [6] T. Eiter and H. Mannila. Computing discrete Fréchet distance. Tech. Report CD-TR 94/64, Information Systems Department, Technical University of Vienna, 1994.
- [7] S. Gaffney, A. Robertson, P. Smyth, S. Camargo and M. Ghil. Probabilistic Clustering of Extratropical Cyclones Using Regression Mixture Models. Climate Dynamics, 29(4):423-440, 2007.
- [8] J. Gudmundsson and M. J. van Kreveld. Computing longest duration flocks in trajectory data. Proc. 14th ACM Symposium on Geographic Information Systems, pp. 35-42, 2006.
- [9] J. Gudmundsson, P. Laube, and T. Wolle. Encyclopedia of GIS, chapter Movement Patterns in Spatio-Temporal Data. Springer, 2008. To appear.
- [10] J. Gudmundsson, J. Katajainen, D. Merrick, C. Ong and T. Wolle. Compressing spatio-temporal trajectories. Proc. 18th Internat. Symposium on Algorithms and Computation, pp. 763-775, 2007.
- [11] R. H. Güting and M. Schneider. Moving Objects Databases. Morgan Kaufmann Publishers, 2005.
- [12] M. Hadjieleftheriou, G. Kollios, V. J. Tsotras and D. Gunopulos. Indexing spatio-temporal archives. The VLDB Journal, 15(2):143-164, 2006.
- [13] J. Håstad. Clique is hard to approximate within  $n^{1-\epsilon}$ . ACTA Mathematica, 182:105-142, 1999.
- [14] K. Kalpakis, D. Gada and V. Puttagunta. Distance measures for effective clustering of arima time-series. Proc. 1st IEEE Internat. Conference on Data Mining pp. 273-280, 2001.
- [15] N. Mamoulis, H. Cao, G. Kollios, M. Hadjieleftheriou, Y. Tao, and D. Cheung. Mining, indexing, and querying historical spatiotemporal data. Proc. 10th ACM SIGKDD Internat. Conference On Knowledge Discovery and Data Mining, pp. 236-245, 2004.
- [16] M. Nanni and D. Pedreschi. Time-focused density-based clustering of trajectories of moving objects. Journal of Intelligent Information Systems, 27(3):267-289, 2006.
- [17] J.-G. Lee, J. Han and K.-Y. Whang. Trajectory clustering: a partition-and-group framework. Proc. ACM SIGMOD Internat. Conference on Management of Data, pp. 593-604, 2007.
- [18] S. Sältenis, C. S. Jensen, S. T. Leutenegger and M. A. Lopez. Indexing the positions of continuously moving objects. Proc. ACM SIGMOD Internat. Conference on Management of Data, pp. 331-342, 2000.
- [19] F. Verhein and S. Chawla. Mining spatio-temporal association rules, sources, sinks, stationary regions and thoroughfares in object mobility databases. Proc. 11th Internat. Conference on Database Systems for Advanced Applications (DASFAA), pp. 187-201, 2006.
- [20] M. Vlachos, D. Gunopulos and G. Kollios. Discovering Similar Multidimensional Trajectories. Proc. 18th Internat. Conference on Data Engineering, pp. 673-684, 2002.
- [21] U. Wilensky. NetLogo (and NetLogo User Manual). <http://ccl.northwestern.edu/netlogo>, 1999.

## A Definitions

**Definition 2** Let  $f : I = [l_I, r_I] \rightarrow \mathbb{R}^c$  and  $g : J = [l_J, r_J] \rightarrow \mathbb{R}^c$  be two curves, and let  $|\cdot|$  denote the Euclidean norm<sup>1</sup> in  $\mathbb{R}^c$ . Then the Fréchet distance  $\delta_F(f, g)$  is defined as

$$\delta_F(f, g) = \inf_{\substack{\alpha: [0,1] \rightarrow I \\ \beta: [0,1] \rightarrow J}} \max_{t \in [0,1]} |f(\alpha(t)) - g(\beta(t))|,$$

where  $\alpha$  and  $\beta$  range over continuous and non-decreasing parameterisations with  $\alpha(0) = l_I$ ,  $\alpha(1) = r_I$ ,  $\beta(0) = l_J$  and  $\beta(1) = r_J$ .

For a set of  $m > 2$  curves there is a natural extension due to Dumitrescu and Rote [5].

**Definition 3** For a set of  $m$  curves  $\mathcal{F} = \{f_1, \dots, f_m\}$ ,  $f_i := I[a_i, a] \rightarrow \mathbb{R}^2$  we define the Fréchet distance as

$$\delta_F(\mathcal{F}) = \inf_{\substack{\alpha_1: [0,1] \rightarrow [a_1, a'_1] \\ \vdots \\ \alpha_m: [0,1] \rightarrow [a_m, a'_m]}} \max_{t \in [0,1]} \max_{1 \leq i, j \leq m} |f_i(\alpha_i(t)) - f_j(\beta_j(t))|,$$

where  $\alpha_1, \dots, \alpha_m$  range over continuous and increasing functions with  $\alpha_i(0) = a_i$  and  $\alpha_i(1) = a'_i$ ,  $i = 0, \dots, m$ .

In this paper we will also consider the discrete Fréchet distance, which is a variant of the Fréchet distance for polygonal curves. For the discrete Fréchet distance only distances between vertices are computed. It is defined using *couplings* of the vertices of the curves [6]. Consider two polygonal curves  $P, Q$  in  $\mathbb{R}^c$  given by the sequences of their vertices  $\langle p_1, \dots, p_n \rangle$  and  $\langle q_1, \dots, q_m \rangle$ , respectively. A coupling  $C$  of the vertices of  $P, Q$  is a sequence of pairs of vertices  $C = \langle C_1, \dots, C_k \rangle$  with  $C_r = (p_i, q_j)$  for all  $r = 1, \dots, k$  and some  $i \in \{1, \dots, n\}, j \in \{1, \dots, m\}$ , fulfilling

- $C_1 = (p_1, q_1)$  and  $C_k = (p_n, q_m)$
- $C_r = (p_i, q_j) \Rightarrow C_{r+1} \in \{(p_{i+1}, q_j), (p_i, q_{j+1}), (p_{i+1}, q_{j+1})\}$  for  $r = 1, \dots, k - 1$ .

**Definition 4** Let  $P, Q$  be two polygonal curves in  $\mathbb{R}^c$ . Let  $|\cdot|$  denote an underlying norm on  $\mathbb{R}^c$ . Then the discrete Fréchet distance  $\delta_{dF}(f, g)$  is defined as

$$\delta_{dF}(f, g) = \min_{\text{coupling } C} \max_{(p_i, q_j) \in C} |p_i - q_j|,$$

where  $C$  ranges over all couplings of the vertices of  $f$  and  $g$ .

## B Hardness Proofs

**Lemma 1** Finding any distance approximation of the  $\text{SC}(m, \max, d)$  problem is 3SUM-hard.

**Proof.** To prove this, we make a reduction from the problem POINT-ON-3-LINES to a special case of our clustering problem.

In the POINT-ON-3-LINES problem, you are given a set of lines and asked the question whether there is any point that is on three of the lines simultaneously. This problem was proven to be 3SUM-hard by Gajentaan and Overmars [24]: this means that it is at least as hard as 3SUM for which no subquadratic time algorithm has been found yet, which is an indication of the inherent hardness of the problems. For a weak model of computation a lower bound of  $\Omega(n^2)$  for those problems exists [26].

Consider a set  $S = \{s_1, \dots, s_n\}$  of  $N$  lines in the plane. Compute in  $O(n \log n)$  time a bounding box  $B$  such that all intersections between lines in  $S$  occur within  $B$ . Remove the parts of the lines

<sup>1</sup>Other norms are possible as well, see [2].

outside  $B$  such that we obtain a set  $S' = \{s_1, \dots, s_n\}$  of  $N$  segments, and connect the endpoints outside  $B$  such that we obtain one trajectory  $T$ . The segments can be connected by polygonal paths outside  $B$  in such a way that no three lines intersect in a point outside  $B$ .

Assume we have an approximation algorithm  $A$  that computes, for a given trajectory and parameters  $m$  and  $d$ , a set of  $m$  subtrajectories of length  $\ell$  that are all within distance  $c \cdot d$  from each other, where  $\ell$  is the optimal value for these  $m$  and  $d$ . Then we can run this algorithm on  $T$  with parameters  $m = 3$  and  $d = 0$ . If we obtain any solution, then there must be three lines in  $S$  that intersect in a common point, otherwise no such point exists. The observation follows. ■

**Theorem 1** *The decision problem  $SC(m, \ell, d)$  is NP-complete.*

**Proof.** The problem is obviously in NP since a candidate subtrajectory cluster can be verified in polynomial time by computing the Fréchet distance between every pair of subtrajectories in the candidate cluster.

Next, we prove NP-hardness by a reduction from MAXCLIQUE [25]. In this problem, you are given a graph  $G = (V, E)$  with  $n$  vertices  $v_1, \dots, v_n$ , and want to determine whether a *clique* of size  $m$  exists: a complete subgraph with  $m$  vertices.

We construct an instance of the SC problem as follows, see Fig. 6. We choose the distance  $d = 1$ ; the instance can be scaled to realise any value of  $d$ . Every vertex  $v$  in  $V$  is represented by a subtrajectory  $T_v$  containing  $3n + 1$  points. We define the locations of all subtrajectories at all steps as follows. The first point of a subtrajectory is at the origin of the plane. For vertex  $v_j$  and  $0 < i \leq n$ , the trajectory  $T_{v_j}$  at step  $3i$  is at  $(3i, 0)$  and at steps  $3i - 1$  and  $3i - 2$  is at

- $(3i - 1, 1)$  and  $(3i - 2, 1)$  if  $i = j$ ,
- $(3i - 1, 0)$  and  $(3i - 2, 0)$  if  $i \neq j$  and  $(v_i, v_j) \in E$ , and
- $(3i - 1, -1)$  and  $(3i - 2, -1)$  if  $i \neq j$  and  $(v_i, v_j) \notin E$ .

We connect all consecutive subtrajectories to obtain the trajectory  $T$ . Note that two subtrajectories can easily be connected by a path containing two edges, in such a way that no two subpaths connecting two subtrajectories are close to each other.

We observe that two subtrajectories  $T_{v_i}$  and  $T_{v_j}$  have Fréchet distance at most  $d$  if the vertices  $v_i$  and  $v_j$  are connected by an edge. Hence, the question whether a clique of size  $m$  exists in  $G$  can be answered by answering the question whether a subtrajectory cluster of size  $m$  exists for the duration of all  $\ell = 3n + 1$  time steps. This proves the NP-completeness of the decision version of the maximum subset SC problem. ■

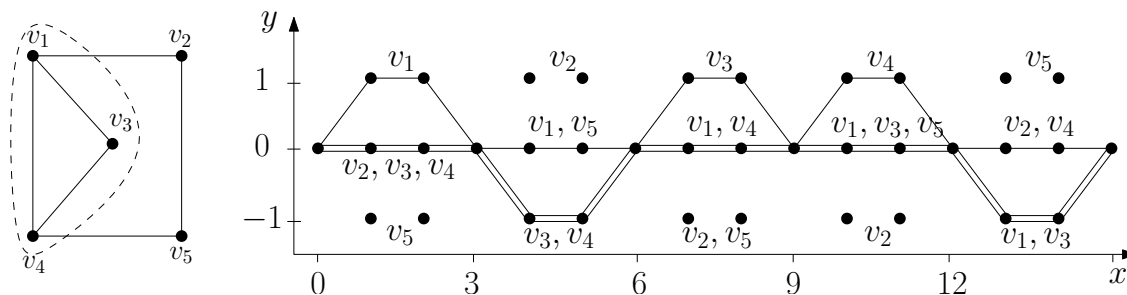


Figure 6: Illustrating the reduction in the proof of Theorem 1, in particular showing the subtrajectories corresponding to the 3-Clique  $v_1, v_3, v_4$ .

## C Relation Between Cluster Curves in the Free Space and Subtrajectory Clusters

Here, we give formal proofs of the implications in Lemma 2 used for the algorithm in Section 4. For this, we use the following lemma of Alt and Godau [2].

**Lemma 4** (*Alt and Godau [2]*)

*Two polygonal curves  $f$  and  $g$  in the plane, with  $n$  and  $m$  vertices respectively, have Fréchet distance  $\delta_F(f, g) \leq d$ , if and only if there exists a curve in the corresponding free space diagram  $F_d(f, g)$  from  $(0, 0)$  to  $(n, m)$  that is monotone in both coordinates.*

We now repeat the lemma and present its proof.

**Lemma 2** *Let  $T$  be trajectory and  $T'$  the subtrajectory of  $T$  that starts at  $s$  and ends at  $t$ , where  $t - s = \ell$ . Then the following holds:*

- *If there exist  $m$  cluster curves within the free space diagram  $F_d(T, T)$  between  $l_s$  and  $l_t$ , then there exists a subtrajectory cluster  $C_T(m, \ell, 2d)$  containing  $T'$ .*
- *If there exists a subtrajectory cluster  $C_T(m, \ell, d)$  containing  $T'$ , then there exists a set of  $m$  cluster curves within the free space diagram  $F_d(T, T)$  between  $l_s$  and  $l_t$ .*

**Proof.** Let  $C = \{c_1, \dots, c_m\}$  be the set of  $m$  cluster curves within the free space diagram  $F_d(T, T)$  from  $s$  to  $t$ . According to Lemma 4 this implies that the pairwise Fréchet distance between every pair of subtrajectories corresponding to the curves is at most  $d$ . Since this holds for every pair of curves it follows from the result by Dumitrescu and Rote [5] that the set of trajectories has Fréchet distance at most  $2d$ . Finally, since the  $y$ -coordinates of two curves in  $C$  overlap in at most a point, the time intervals for two subtrajectories also overlap in at most a point. This completes the proof of the first part.

Let  $C$  be the solution to  $\text{SC}(m, \ell, d)$  containing  $T'$ , i.e.,  $C = \{f_1, \dots, f_m\}$  is a set of subtrajectories of  $T$  and the Fréchet distance of  $F$  is at most  $d$ . This implies that also all pairwise Fréchet distances in  $C$  are at most  $d$ . W.l.o.g. let  $T' = f_1$  the subtrajectory from  $t_a$  to  $t_b$ . From Lemma 4 it follows that for every trajectory in  $C$  there exists a cluster curve within the free space diagram  $F_d(T, T)$  from the  $x$ -coordinate  $s$  to the  $x$ -coordinate  $t$  that is monotone in both coordinates. Finally, since the time intervals for two subtrajectories overlap in at most a point, their corresponding cluster curves within  $F_d(T, T)$  also overlap in at most a point. This completes the proof. ■

Note that the above lemma also implies that if there is no subtrajectory cluster  $C_T(m, \ell, 2d)$  containing  $T'$  then there do not exist  $m$  cluster curves between  $t_a$  and  $t_b$ .

## D Computing the Vertically Propagated Points Implicitly

Consider the column of the free space that is being added. From the cell in row  $j$  we can reach a lower cell if the horizontal cell boundaries inbetween have a non-empty intersection. Formulated differently, for the horizontal cell boundaries inbetween the maximum left boundary is smaller or equal the minimum right boundary. We can test this condition for all columns using two doubly linked lists of length at most  $n$ , one for the left boundaries and the other for the right boundaries. See Figure 7 for an illustration. We iteratively add the horizontal cell boundaries from top to bottom, updating these lists in each step. The list entries contain three values: (1) the row index, (2) the position of the left or right boundary, respectively, at that row index, and (3) the furthest left column index reachable from this or a reachable upper cell of this column. For the right boundary, we are interested in small boundary positions, whereas for the left boundary we are interested in large boundary positions. We now describe the process for the right boundary list, for the left boundary list it is analogous.

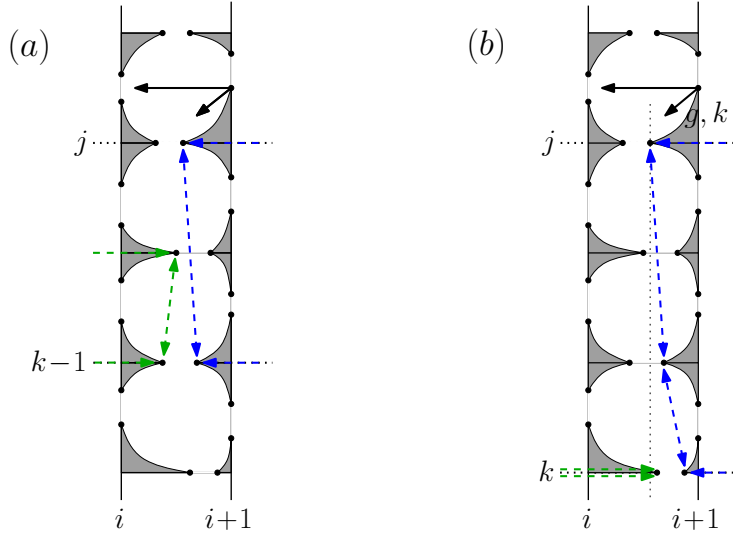


Figure 7: Updating  $l_t$  for the continuous Fréchet distance. The doubly linked list are shown – by dashed arrows – (a) before and (b) after inserting the horizontal cell boundary at row index  $k$ .

The first list entry for the right boundary list stores the currently smallest boundary position at some row index  $l$  (in Figure 7,  $l = j$ ). The next list entry stores the next smallest boundary position at a column index smaller than  $l$ , and so on. To add a new horizontal cell boundary, we first compute this cell boundary. If it is non-empty, we compare its position with the entries of the list, starting at the last entry. We delete all entries which are larger and store it at the first position where it is not smaller than the previous entry. During this process, we also compute the third list entry, i.e., the furthest reachable left column index. First we set this to the column index reachable in this row. Whenever a list entry is deleted (because its position was larger) we take the minimum of the current column index and the one stored by the entry which is being deleted. Finally, we also take the minimum with the column index stored by the last list entry not deleted (if this exists). The column indices stored in the lists will therefore be non-increasing. For one such step we may need to do a linear number of comparisons and deletions, however in total each entry is inserted and deleted exactly once in the process. After the insertion, we check whether the current maximum right position is larger or equal than the current minimum left position, i.e., we compare the positions in the first entries of the two lists. If this test is positive, the free space intervals still have a non-empty intersection and we continue with adding the next horizontal cell boundary. If the test is negative or if the horizontal cell boundary did not contain free space, then there is no longer an intersection and we first set edge labels and modify the lists before we continue with the next horizontal cell boundary.

Let the  $j$ th row be the first row for which the edge label has not been set and assume we have just added the horizontal cell boundary at row index  $k < j$ . If this cell boundary was empty we set all edge labels from  $j$  to  $k$  to  $g, k$ , where  $g$  is the column index stored as third list entry in the last list elements of both lists. Then we delete all entries of both lists. If the cell boundary was not empty, then the minimum right position was smaller than the maximum left position. One of these two positions is in the just added row index  $k$ . Assume w.l.o.g. that the minimum right position is at row index  $l$ , with  $k \leq l \leq j$ , and the maximum left position is at row index  $k$ . Then we set the edge labels for rows  $j$  to  $l$  to  $g, l$ , where  $g$  is the third list entry stored in the right list for row index  $l$ . Then we delete all entries of the right list up to and including  $l$ . We proceed by again comparing the top list elements. If now the minimum right position is not larger than the maximum left position, we can proceed with adding the next horizontal cell boundary at row index  $k - 1$ . If there still is no intersection, we also set the edge labels to the next entry of the

right list, and so on.

## E Arbitrary Subtrajectories

In this section, we cluster subtrajectories using the Fréchet distance where all subtrajectories (including the reference subtrajectory) may start and end at arbitrary points on the given trajectory. We use the same algorithm as in Section 4.2.2 and extend it to handle the case of the reference subtrajectory not starting and ending at vertices of the trajectory.

For this, we first add more critical points to the free space, namely those points where – intuitively – free space starts or ends within a cell. That is, all points where the intersection of a vertical segment with the free space is a single point. See Figure 8 for an example. These critical points are also propagated to the right in the free space. The total number of critical points introduced in each cell is still at most eight. We will use these new critical points also as new event points, thus we now have  $O(n^2)$  event points.

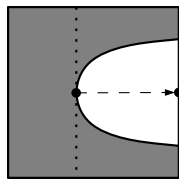


Figure 8: Further critical points needed for arbitrary subtrajectories.

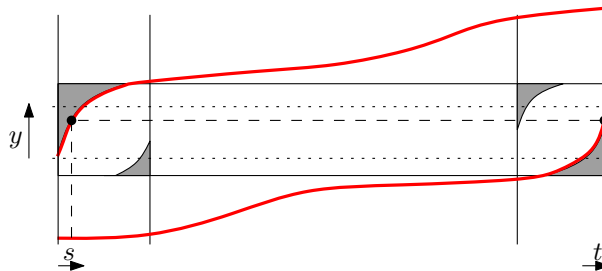


Figure 9: A dependency between two cluster curves for arbitrary subtrajectories. The dotted lines indicate the region of overlap in the  $y$ -coordinate.

With the added critical points and event points, we run the same algorithm as in the previous section. However, now in between event points we also need to find the longest cluster of  $m$  curves. Suppose we have just moved either the sweep line  $l_s$  to  $l_{s+1}$  or  $l_{t-1}$  to  $l_t$ . And suppose we have found  $m$  cluster curves between  $l_{s+1}$  and  $l_{t-1}$  but not between  $l_s$  and  $l_{t-1}$  or  $l_{s+1}$  and  $l_t$ , respectively. Then we search for the longest cluster of  $m$  curves between  $l_s$  and  $l_t$ . For this we first select a set of possible curves, which may have *dependencies* between each other. Figure 9 shows an example of a dependency: Suppose the sweep line  $l_s$  is at the left vertical dashed line. If we want to include both curves in the cluster then  $l_t$  may not be to the right of the right vertical dashed line. We can consider the largest feasible  $t$  as a function in  $s$ . This is an algebraic function of degree at most 4 since the free space boundaries are algebraic functions of degree at most 2 [2].

We can select a set of at most  $n$  possible curves, in which each curve may have a dependency on at most two other curves (to one curve above and one curve below). For this, we start by finding and selecting the top most curve as before. Dependencies only occur between a curve ending at the top most point of an interval and a curve starting at the bottom most point of an interval (see again Figure 9). Thus, when a selected curve ends on a top most point, we also add – if this exists – the dependent curve starting at the bottom most but higher point on  $l_t$  in the same row. Furthermore, we add as before the next non-dependent curve starting at the same height or lower. Because dependent curves start on bottom most points, the dependent and non-dependent curve start in different rows. Thus, we get at most one curve starting in each row, and  $n$  curves in total.

In total, the set of possible curves contains at most  $n$  dependencies (at most one per row). For these dependencies, we consider the arrangement of the functions  $l_i(s) := t_i(s) - s$  where  $t_i(s)$  is the largest feasible  $t$  for  $s$  for the  $i$ th dependency ( $1 \leq i \leq 2n$ ). If we have  $m$  compatible curves in the free space at a position in the arrangement then we still have these curves at any position in the corresponding cell of the arrangement. We therefore only need to consider vertices of the arrangement and local maxima of the functions  $l_i$ .

Furthermore, we do not need to consider the whole arrangement but only the  $\leq (m - 1)$ -level, where we count levels from the top, e.g., the 0th level is the upper envelope. Below the  $(m - 1)$ st level we have  $m - 1$  pairs of compatible curves. This yields at least  $m$  compatible curves, for instance by choosing the top most curve, and from each pair of curves the lower curve. The  $\leq (m - 1)$ -level has complexity  $O(m^2 \lambda_4(n/m)) = O(nm2^{\alpha(n/m)})$  and can be computed in  $O(nm2^{\alpha(n/m)} \log n \log(n/m))$  time [27] (see [22] for incremental algorithms).

This gives us  $O(nm2^{\alpha(n/m)})$  vertices of the arrangement and local maxima of the functions at which the maximum length might be achieved. For these points we can query whether there are  $m$  curves in  $O(m)$  time. For this, we store the  $n$  possible curves in an array with pointers from each curve to the next dependent and non-dependent curve.

In total there are  $O(n^2)$  events and for each event we build the arrangement for the  $\leq (m - 1)$  levels which requires  $O(nm2^{\alpha(n/m)} \log n \log(n/m))$  time. Each of the  $O(nm2^{\alpha(n/m)})$  vertices in the arrangement can be checked in  $O(m)$  time. Thus, we have obtained the following results for subtrajectory clustering under the continuous Fréchet distance.

**Theorem 5** *A 2-distance approximation of the  $SC(m, \max, d)$  problem can be computed in time  $O(n^3 m 2^{\alpha(n/m)} (\log n \log(n/m) + m))$  using  $O(n\ell + nm 2^{\alpha(n/m)} \log n \log(n/m))$  space using the continuous Fréchet distance for arbitrary subtrajectories. In these bounds,  $n$  denotes the size of the given trajectory and  $\ell$  the length of the longest subtrajectory cluster found.*

Instead of querying for all of the points we can perform a binary search on the levels, i.e., on the range 0 to  $m - 1$ . By this we query for at most  $O(\min\{nm 2^{\alpha(n/m)}, \log mL(n, m)\})$  points, where  $L(n, m)$  denotes the worst case complexity of the  $m$ th level. While we assume that the binary search improves the worst-case run-time, from the known upper bounds on  $L(n, m)$  we only get an improvement if  $m$  is large. For instance Chan [23] proved that  $L(n, m) \in O(n^{2-1/6-\delta})$  for a positive constant  $\delta$ .

## F Illustration for Experiments

Fig. 10 shows the trajectory described in Section 5. The parts in the longest cluster are thickened and shown in red.

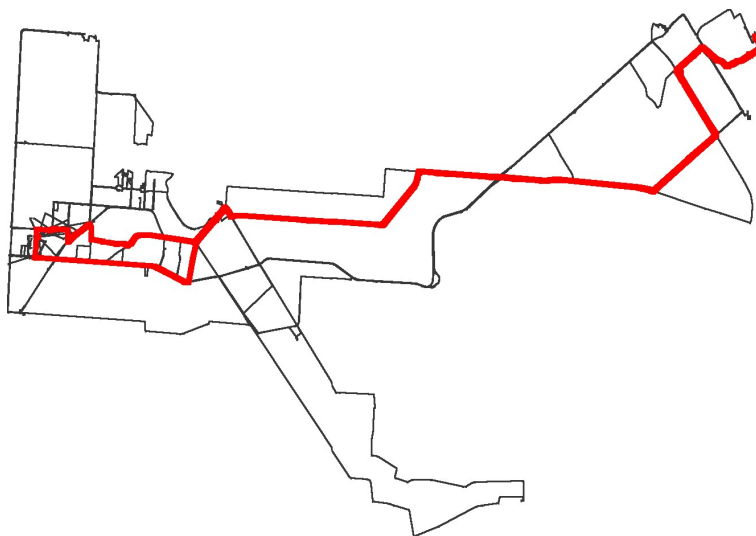


Figure 10: A trajectory with 5,228 time-points where the longest trajectory cluster is marked.

## References

- [22] P. K. Agarwal, M. de Berg, J. Matousek, and O. Schwarzkopf. Constructing Levels in Arrangements and Higher Order Voronoi Diagrams. Proc. 10th Annual ACM Symposium on Computational Geometry, pp. 67–75, 1994.
- [23] T. Chan. On levels in arrangements of curves, III: further improvements. To appear in Proc. 24th ACM Symposium on Computational Geometry, 2008
- [24] A. Gajentaan and M. H. Overmars. On a Class of  $O(n^2)$  Problems in Computational Geometry. Computational Geometry - theory and applications, 5:165–185, 1995.
- [25] M. R. Garey and D. S. Johnson. Computers and intractability. W. H. Freeman, 1979.
- [26] J. Erickson and R. Seidel. Better Lower Bounds on Detecting Affine and Spherical Degeneracies. Discrete & Computational Geometry 13:41–57, 1995.
- [27] M. Sharir. On  $k$ -sets in arrangements of curves and surfaces. Discrete & Computational Geometry, 6:593–613, 1991.