# Constrained free space diagrams: a tool for trajectory analysis*

Kevin Buchin[1], M. Buchin[2], and J. Gudmundsson[3]

[1] Department of Mathematics and Computer Science, Technische Universiteit Eindhoven, The Netherlands. k.a.buchin@tue.nl
[2] Department of Information and Computing Sciences, Utrecht University, The Netherlands. maike@cs.uu.nl
[3] NICTA, Sydney, Australia. joachim.gudmundsson@nicta.com.au

**Abstract.** We propose a new and powerful tool for the analysis of trajectories, which in particular allows for more temporally aware analyses. Time plays an important role in the analysis of moving object data. For many applications it is neither sufficient to only compare objects at exactly the same times, nor to consider only the geometry of the trajectories. We show how to leverage between these two approaches by extending a tool from curve analysis, the free space diagram. Our approach also allows to take further attributes of the objects like speed or direction into account. We demonstrate the usefulness of the new tool by applying it to the problem of detecting single file movement. A single file is a set of moving entities, which are following each other, one behind the other. This is the first algorithm for detecting this movement pattern. For this application, we analyse and demonstrate the performance of our tool both theoretically and experimentally.

## 1 Introduction

Due to technological advances in location-aware devices, trajectory data is becoming more and more available. With this also the need for automatically analysing trajectory data is increasing. Recent work in the area of movement analysis has focused on detecting movement patterns among trajectories, for example flocks [Benkert et al., 2008, Laube et al., 2004], convoys [Jeung et al., 2008], leaderships [Andersson et al., 2008], and popular places [Benkert et al., 2007, Verhein and Chawla, 2006]. In this paper we will present a more general tool that can be used to analyse trajectories with various attributes and restrictions.

Previous work on trajectory analysis has focused on two types of analysis. The first only considers the geometry of the path, that is, ignoring temporal aspects. Much research has been done on this topic suggesting numerous metrics to measure the distance between two paths, in particular the Hausdorff distance [Alt et al., 1995] and the Fréchet distance [Alt and Godau, 1995]. Further distance measures have been considered, such as the longest common subsequence model [Vlachos et al., 2002], a combination of parallel distance, perpendicular distance and angle distance [Lee et al., 2007], the count of common subsequences of length two [Agrawal et al., 1995] and the domain-dependent extraction of a single representative value for the whole series [Kalpakis et al., 2001]. The second type of trajectory analysis focuses on the moving entities but at the same point in time. Several papers consider the average distance at equal times as similarity measure [Buchin et al., 2009b, Nanni and Pedreschi, 2006, Trajcevski et al., 2007]. Furthermore, extensive research has been done on detecting moving clusters (or flocks) [Benkert et al., 2008, Gudmundsson et al., 2007, Jensen et al., 2007, Jeung et al., 2008, Kalnis et al., 2005, Laube et al., 2004], where two entities are in a cluster if they are close to each other during the whole time interval.

* Author Posting. (c) Taylor & Francis, 2010. This is the author's version of the work. It is posted here by permission of Taylor & Francis for personal use, not for redistribution. The definitive version was published in International Journal of Geographical Information Science, Volume 24 Issue 7, July 2010. doi:10.1080/13658810903569598 (http://dx.doi.org/10.1080/13658810903569598). This is a preprint version, and does not contain all material from the final journal version, in particular a comparison to Andersson et al. [2008].

We will present a structure that leverages between the temporal and non-temporal approach, which we call *constrained free space diagram*. Alt and Godau [1995] introduced the *free space diagram* of two curves as a data structure to compute the Fréchet distance between them. The Fréchet distance is a distance measure for continuous shapes such as curves and surfaces, and is defined using parameterisations of the shapes. Since it takes the continuity of the shapes into account it is generally regarded as a more appropriate distance measure than the Hausdorff distance for curves [Alt and Guibas, 1999, Alt et al., 2004, Veltkamp and Hagedoorn, 2001]. We denote the Fréchet distance between to curves $f$ and $g$ by $\delta_F(f,g)$. For completeness, we include its formal definition here.

**Definition 1.** *Let $f : I = [l_I, r_I] \to \mathbb{R}^c$ and $g : J = [l_J, r_J] \to \mathbb{R}^c$ be two curves, and let $|\cdot|$ denote the Euclidean norm*[4] *in $\mathbb{R}^c$. Then the Fréchet distance $\delta_F(f,g)$ is defined as*

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

*where $\alpha$ and $\beta$ range over continuous and increasing functions with $\alpha(0) = l_I$, $\alpha(1) = r_I$, $\beta(0) = l_J$ and $\beta(1) = r_J$.*

Intuitively, the Fréchet distance can be described as follows: Imagine a man walking on one of the curves and his dog on the other curve. The man has the dog on a leash. Both may choose their speed independently but not walk backward. Then the Fréchet distance of the two curves is the shortest leash length that allows the man and his dog to walk on their respective curves from start to end.

Now, let $f$ be a polygonal curve with $n$ vertices $p_1, \ldots, p_n$. We use $\phi_f$ to denote the following natural parameterisation of $f$. The map $\phi_f \colon [1,n] \to \mathbb{R}^c$ maps $i \in \{1, \ldots, n\}$ to $p_i$ and interpolates linearly in between vertices, where $c$ is any dimension. Thus, for a trajectory, $\phi_f$ gives the position of the moving object at time $t$. The free space diagram of two polygonal curves $f$ and $g$, with $n$ and $m$ vertices, respectively, is the set

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

which describes all tuples $(s,t)$ such that the points $\phi_f(s)$ and $\phi_g(t)$ have Euclidean distance at most $d$. Thus, in $F_d(f,g)$ the two axes correspond to the two curves, and a tuple $(s,t)$ in the diagram is in free space if the distance between the corresponding two points $\phi_f(s)$ and $\phi_g(t)$ on the curves is smaller than the given distance threshold $d$. See Figure 1 for an example of a free space diagram, where free space is drawn in white and non-free space is drawn in gray. Now, if there exists an $x$- and $y$-monotone path in the free space, then the Fréchet distance between $f$ and $g$ is less than $d$, because such a path corresponds to suitable reparameterisations $\alpha$ and $\beta$. Furthermore, the Fréchet distance between $f$ and $g$ is the smallest value $d$ such that for all $\delta > d$ such a path exists in the free space diagram with distance value $\delta$.

In this paper, we propose the *constrained free space diagram* as an effective tool for analysing trajectories by taking several different constraints into consideration. We use the following natural generalisation of the free space diagram.

$$F_C(f,g) = \{(s,t) \in [1,n] \times [1,m] \,|\, C((\phi_f(s), A_f(s)), (\phi_g(t), A_g(t)))\},$$

where $C$ is a boolean relation given by a set of constraints and $A_f(s)$ is a set of attributes for the parameterised trajectory $f$ at time $s$. For instance, $C$ may include constraints on the direction or speed of the points. Again, if there is a monotone path in the free space then there exist suitable reparametrisations of $f$ and $g$ that fulfil constraint $C$. More details will be given in the next section.

[4] Other norms are possible as well, see Alt and Godau [1995].

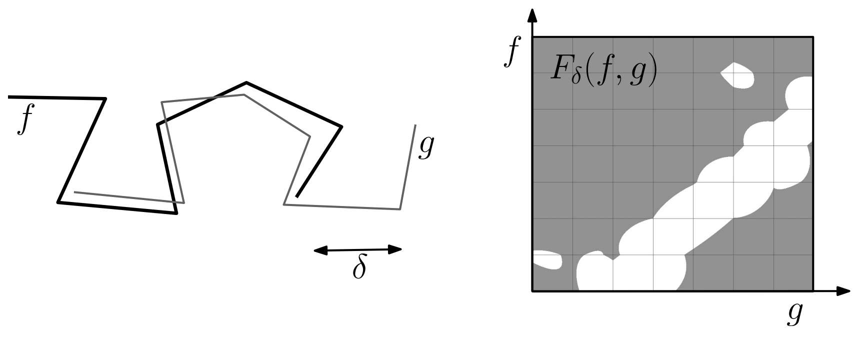

**Fig. 1.** Two parameterised trajectories $f$ and $g$ together with their free space diagram with respect to the distance threshold $\varepsilon$. The white area indicates free space.

For solving analysis tasks, we study the structure of constrained free space diagrams. As a simple example, consider the following question: Did the corresponding objects travel together? For this we can use the constrained free space diagram of a small geographic distance and test whether certain point pairs for the same time, or consecutive times, are in the constrained free space diagram. More examples and details will be given in Section 3.

We demonstrate the usefulness of constrained free space diagrams by applying them to the specific analysis tasks of detecting *following behind behaviour* and *single file movement*. A single file is a movement pattern, where several entities are following each other, one behind the other. This movement pattern occurs frequently among humans, animals, and vehicles. It is a more specific pattern than a flock or herd, and because it does not have a fixed layout, it is challenging to compute. Our algorithm, using constrained free space diagram is, to the best of our knowledge, the first algorithm for this problem. A related pattern is the *leadership pattern* Andersson et al. [2008].

*Overview.* In Section 2 we define the constrained free space diagram more formally and discuss standard constraints for trajectory analysis. Further, in Section 3 we give examples of applications and in Section 3.5 we specifically study the task of detecting *following behind behaviour*. In Section 4 we give algorithms for detecting single file movement based on the constrained free space diagram, and we analyse these algorithms both theoretically and experimentally.

## 2 Constrained free space diagrams

Next we define constrained free space diagrams in more detail and then, in Section 2.2, we state several useful examples of constraints for trajectory analysis.

### 2.1 Definition

Let a spatio-temporal trajectory $f$ of a moving entity $a$ be given by $n$ time-space positions $((t_1, p_1), \ldots, (t_n, p_n))$, where $p_i$ are the coordinates of $a$ at time $t_i$. For simplicity we assume that in between two consecutive time stamps $t_i$ and $t_{i+1}$ the entity $a$ moves along a straight line from $p_i$ to $p_{i+1}$ for $i = 1, \ldots, n$. We could use any other model [Tremblay et al., 2006] that would allow for an efficient computation of the Fréchet distance, e.g., Bézier, hermite and cubic splines. Now, let $f$ and $g$ be two trajectories. A free space diagram of $f$ and $g$ is defined as described in the introduction. We will use also the natural parametrisations described in the introduction. That is, for a trajectory $f$ over the time interval $[t_1, t_n]$, we use the parametrisation $\phi_f\colon [t_1,t_n] \to \mathbb{R}^c$ where $\phi_f(t)$ gives the position of the moving entity at time $t$. We use parameterisations because they capture the movement of the entity in the flow of time. That is, by varying $t$ from $t_1$ to $t_n$, the point $\phi_f(t)$ moves along the trajectory from the start point to the end point.

For polygonal curves the free space diagram consists of $n_1 \times n_2$ cells where $n_1$ and $n_2$ is the complexity of $f$ and $g$, respectively. A single cell is the free space diagram of two line segments and the free space inside a cell is convex [Alt and Godau, 1995], see Fig. 1. It therefore usually suffices to compute the cell boundaries of the free space diagram which can be computed in constant time each. As noted above, a monotone path in the free space diagram corresponds to reparametrisations $\alpha$ and $\beta$ of the trajectories $f$ and $g$. More explicitly, the monotone path can be described as the function $\rho : [0,1] \to [t_1,t_n] \times [t'_1,t'_m]$ with $\rho(t) = (\alpha(t),\beta(t))$. We can also interpret this as a matching between the points $\alpha(t)$ and $\beta(t)$ on the two curves.

In contrast to the case of curves, for trajectories the parameterisations correspond to an important parameter, namely time. As a consequence many additional constraints may be of interest to add to the analysis. Sometimes additional attributes like the direction of movement or acceleration are relevant. If further attributes are used in the analysis, we denote the list of attributes for trajectory $f$ at a given time $t$ by $A_f(t)$. As stated above the *constrained free space diagram* can now be defined as:

$$F_C(f,g) = \{(s,t) \in [1,n] \times [1,m] \,|\, C((\phi_f(s),A_f(s)),(\phi_g(t),A_g(t)))\},$$

where $C$ is a binary relation given by a set of constraints.

## 2.2 Constraints for trajectory analysis

We distinguish between two types of constraints, which concern the matching between the points on the two curves; (1) properties of the matched points (local constraint) and (2) properties of the matching (global constraint). Multiple constraints, of the same or different type, can be combined.

*Constraints on matched points.* For many applications on curves, matched points should be close in space. For trajectories the points should often also be close in time, or have some other kind of temporal relation. Examples of this type of constraints are

- temporal constraint (e.g., similar or consecutive times),
- distance constraint (e.g., Euclidean distance),
- directional constraint (e.g., similar directions of movement), and
- attributional constraints (e.g., similar size).

An attributional constraint allows to match only points where certain attributes of the entities at these points (along these segments) fulfil some relation. For example, assume we know the type of underlying terrain. Then we can require that only points may be matched where the entities are moving over the same type of terrain. Or, more generally, for any numerical attribute, e.g., size or temperature, we may require that the difference in this attribute is small (bounded by some threshold).

*Constraints on the matching.* The previous constraints all make restrictions on the free space. Constraints on the matching restrict how the path passes through the free space. The most important such constraint is that the path should be monotone. When dropping this constraint, the path no longer corresponds to a one-to-one reparameterisation. Nonetheless, there might be situations where one might want to drop this constraint, and indeed the corresponding variant of the Fréchet distance the *weak Fréchet distance*, is well known [Alt and Godau, 1995].

Two general and useful constraints of this type are:

- bound on the slope of the path in free space, and
- bound on the length of the path in free space.

The effect of these constraints depends on the parametrisation of the free space diagram, i.e., the parametrisation of the curves defining the free space. Two natural choices are parameterising by time or distance, i.e., scaling the free space such that it is uniform either for time or distance of the curves.

Parameterising by time, or distance, affects the ratio of time, or distance, travelled. Bounding the slope restricts these ratios locally, whereas bounding the length restricts the ratios globally. For instance, if we parameterise by distance and bound the slope, this results in a bound on the distance ratio (everywhere). In contrast, if we parameterise by distance and bound the Euclidean length of the path, then we locally allow a large distance ratio, but overall the distance ratio does not deviate too much.

The length of the path can be measured in metrics other than Euclidean length. For instance, we can use the min-link distance, i.e., we count the number of segments of a polygonal path. This measures how often we have to cut the trajectories such that the pieces can be matched by a simple scaling of the time axes. We can also compare paths based on the difference of a quantitative attribute at matched points, e.g., based on the average distance between the entities. For this, it is useful to take the average with respect to the $L_1$ path length, because all monotone paths then have the same length. Thus, using $L_1$ path length the average cannot be manipulated by for instance zig-zagging to increase the length in areas of small difference in the attribute.

In a free space diagram all parameters are thresholded, and if there is at least one monotone path in it, there are typically infinitely many. In practice, it will often be relevant to select one path, which gives a concrete matching of the curves. For this, we can employ path lengths. Instead of simply bounding the path length, we now optimize a criteria on the path length. For instance, we can select the path with the smallest Euclidean length or the smallest average distance between the entities at matched points.

**Computing constrained free space diagrams** Recall from the work of Alt and Godau [1995] that the free space diagram can be computed cell by cell. For the Fréchet distance, the free space is convex inside cells and only cell boundaries need to be computed. Furthermore, instead of explicitly searching for a path in the free space, its existence can be determined by computing the *reachable free space diagram*, i.e., the free space reachable by a monotone path. Most constraints on matched points result in convex cells. In this case, we can compute the reachable free space in the same manner and incorporate further constraints. Constraints that do not result in convex cells are for instance the geodesic distance or visibility. For these cases, other techniques have been developed [Cook IV and Wenk, 2008a]. All constraints we consider from now on generate convex cells.

The computation and time complexity for incorporating constraints on matched points (local constraints) depends on the complexity of the (constraining) boolean function of the attributes. In particular, it depends on whether the values of the boolean function can be automatically extracted from the vertices (cell boundaries in the free space) to the points along the edges (interior of free space cells). For example this is the case for the Fréchet distance, which uses a distance constraint. Therefore a cell in the free space diagram of the Fréchet distance can be computed in constant time [Alt and Godau, 1995]. The computation of attributive constraints per cell depends on the interpolation of the attribute from the vertices to the segments. For instance, if attribute values are measured on an interval scale and are interpolated linearly, the constraint can be incorporated in a similar way as distance. For nominal data the attribute of one of the vertices can be used. Thus, for a cell in the free space diagram the constraint is fulfilled either for the whole cell or for none of the cell, since the attribute is constant on the corresponding segments of the trajectory. Similarly, directional constraints are easy to integrate since the direction of movement is constant on segments.

Linear temporal constraints of the form $t - \tau_{\min} \geq s \leq t + \tau_{\max}$ can be realised by intersecting the free space with the two half planes $s - t \geq -\tau_{\min}$ and $s - t \leq \tau_{\max}$. Such a constraint signifies that the time difference between the trajectories is at least $\tau_{\min}$ and at most $\tau_{\max}$. This cuts out a diagonal "strip" of the free space, as will be discussed in more detail in Section 3.5 where we give an example of this. See also Figure 6. Note that if $\tau_{\max} - \tau_{\min}$ is constant, which it typically is, then such a temporal constraint reduces the complexity of the remaining strip of the free space to linear, in contrast to the quadratic complexity of the whole free space diagram. We can use the low complexity of the free space diagram to reduce the running time and the space requirement of the algorithm (see Section 4.1). For this, we simply look only at actually reachable cells and store only cells of which the neighbors have not yet been processed. This technique is generally useful for constraints on the matched points, since they are likely to reduce the complexity of the free space diagram.

Constraints on the matching (global constraints) concern the path in free space, e.g., its slope and length. They can be incorporated in the reachability of free space. A slope constraint is applied to each cell of the free space, when computing the reachable free space. The computation is done in an order that respects monotone paths. One such order is column by column from left to right and in each column from bottom to top. In this computation only reachable cells need to be stored. An example of a free space with slope constraints is given in Figure 2(c). It shows the free space (boundaries) in a cell that are reachable by a path of slope at most $\alpha$ from the bold bottom segment. Here, bold segments indicate the boundaries of the free space. For a length constraint, we add a shortest path map construction to the reachable free space. Note that this increases the complexity of a cell to linear, and thus the complexity of the whole free space diagram to cubic. A shortest path map construction can also be used when bounding the average value of an attribute [Buchin et al., 2009a]. Cook IV and Wenk [2008b] show how to compute the min-link distance. Note that some constraints may be costly to incorporate. In the following, we will consider example applications of the constrained free space diagram and discuss how to incorporate the constraints for these applications efficiently.

## 3 Applications and constraints

Constrained free space diagrams can be applied to different trajectory analysis tasks. The most natural application of constrained free space diagrams is determining *trajectory similarity*, which in turn can be used for *trajectory simplification* (Section 3.1). In Section 3.2 we briefly discuss two types of constraints to bound the difference in speed between two moving entities. Constrained free space diagrams can also be used to detect specific movement patterns. We will illustrate this by the example of *single file* and *following behind movement* (Section 3.4).

### 3.1 Trajectory similarity and simplification

For similarity tasks the constraints depend on the definition of "trajectory similarity". We may want to assess the similarity of two trajectories taking into account a combination of their difference in time, speed, direction or distance. For this, we simply apply those constraints to the free space diagram and run the known algorithm [Alt and Godau, 1995]. Their algorithm computes the reachable free space cell-by-cell, in an order that respects monotonicity, e.g., left-to-right and bottom-to-top. If the upper left corner is reachable, the algorithm answers "yes", else it answers "no". Without a time constraint, this algorithm runs in quadratic time, and with a time constraint it runs in time proportional to the number of cells of the free space diagram within the diagonal strip shown in Fig. 6, which is typically linear. More explicitly, let $k$ be the horizontal width of the diagonal strip. The algorithm then runs in $O(kn)$ time. The value $k$ is given by the time constraint and will typically be constant, which then results in a linear run time.

We can use trajectory similarity with constraints for trajectory simplification. That is, to compute for a given trajectory a trajectory with fewer vertices which captures the essential properties of the original trajectory. For a given application, the constrained free space diagram gives the flexibility to maintain the properties essential to the application, e.g., if speed is important we add a speed constraint. A fast algorithm for trajectory simplification that can be easily adapted to work with the constrained free space diagram is the near-linear time algorithm by Agarwal et al. [2005].

## 3.2 Bounding the speed difference

For trajectories an important property is speed. For bounding the difference in speed using constrained free space diagrams we have two options from which we choose according to the application. We can locally bound the speed ratio before reparameterising using an attributional constraint. Alternatively, we can bound the ratio of travel distances after reparameterisation, i.e., bound the ratio $d_1/d_2$ where $d_1, d_2$ are the distances travelled on the trajectories after parameterisation. This bound may be interpreted as difference in speed after the reparameterisation by the matching. For an illustration of these two options, consider the trajectories in Figure 2(a). Both trajectories travel with equal constant speed a similar route, but first one “detours” and then the other. If we consider speed as a fixed attribute, then we can match these two trajectories with speed ratio 1 and a distance constraint $\delta$ (shown in the figure). In this matching a distance of 0.5 on one trajectory is matched to a distance of 1 on the other trajectory, which results in a ratio 2 of the travel distances. The path in the free space diagram in Figure 2(b) corresponds to such a matching. If on the other hand, we want to bound the ratio of travel distances, we restrict to paths of slope close to $45^\circ$ in the free space.

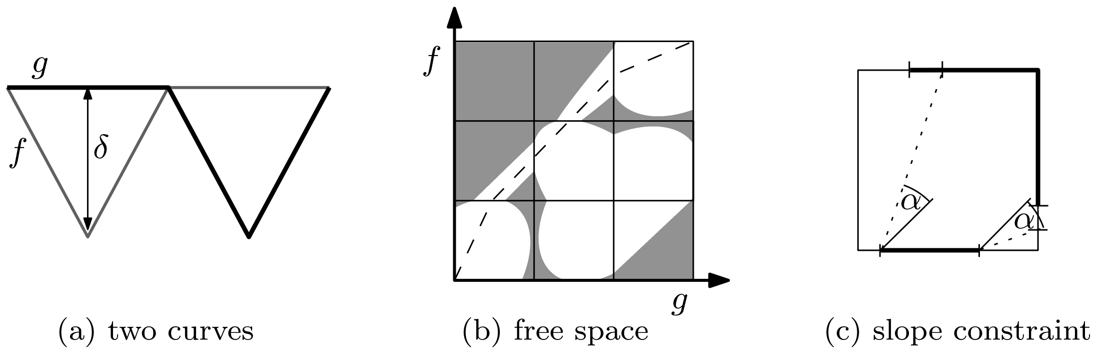

**Fig. 2.** (a) Computing the reachable free space in a cell with a slope constraint. (b) Two trajectories and distance value $\delta$. (c) A monotone path (dashed) in the free space of the trajectories in (b).

## 3.3 Different distance constraints

When using a distance constraint on matched points, distances other than the Euclidean distance might be relevant, e.g., geodesic distance (i.e., taking into account obstacles), network distance, or visibility (as 0-1-distance). Figure 3 shows an example of a constrained free space diagram where $F_C$ is 1 if the moving objects are mutually visible and 0 otherwise. Processing the constrained free space diagram for these distance measures requires different techniques than for the Fréchet distance. Recent progress has been made for the geodesic distance [Cook IV and Wenk, 2008a] and for visibility constraints [Cook IV and Wenk, 2008b].

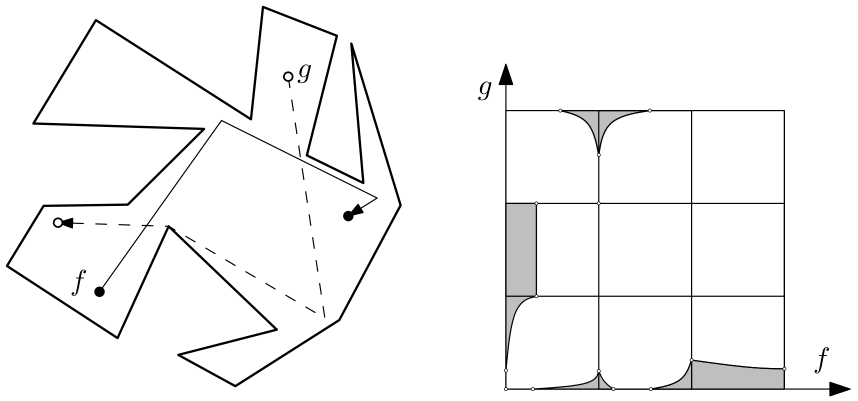

**Fig. 3.** Two parameterised trajectories $f$ and $g$ and the constrained free space diagram $F_C$ which is 1 if the points are mutually visible and 0 otherwise.

### 3.4 Single file and following behind movement

A *single file* and *following behind behaviour* are frequently occurring movement patterns. One entity is considered to be following another, if it moves along a similar track, but later in time. A single file consists of a group of moving entities where one entity is leading the group and all others are following, one behind the other. This pattern can be found among animals (e.g., ducks), humans (e.g., hikers), and vehicles (e.g., bikes). Reasons for this moving pattern are energy efficiency, road conditions, orientation, and security. An overview of studies on the energy efficiency of single files is given by Fish [1999].

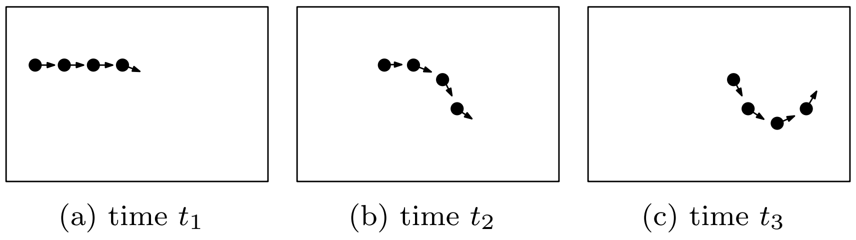

**Fig. 4.** The layout of a single file typically changes over time.

The layout of a single file is not fixed, but changes over time, i.e., it differs at different time stamps, see Figure 4. At each time stamp, a relation between every pair of entities in the sequence needs to be maintained. This makes detecting a single file pattern challenging to compute. In the remainder of this paper, we give models and algorithms for single file and following behind behaviour as a case study for the use of constrained free space diagrams. First, we give definitions for *following behind* in Section 3.5 and then, based on these definitions, we give algorithms for detecting single files in Section 4.

Before we give our algorithms, we would like to note that a single file cannot be detected in a straightforward manner using most existing approaches for other movement patterns. We briefly discuss existing definitions of movement patterns and their connection to the single file movement pattern. One of the first movement patterns studied [Jensen et al., 2007, Jeung et al., 2008, Kalnis et al., 2005] was moving clusters. A moving cluster is sometimes also called a *variable subset flock* [Gudmundsson and van Kreveld, 2006]. Closely related to moving clusters is the *flock* pattern, or *fixed subset flock*. This problem has been studied in several papers [Benkert et al., 2008, Gudmundsson et al., 2007, Laube et al., 2004]. Even though different papers use different definitions the main idea is that a flock consists of a fixed or variable set of entities moving together as a cluster at all times. Note that the definition of a cluster only requires all the entities to be close to the center of the cluster thus a flock can be detected without calculating a relationship between every pair of entities. The authors have attempted to modify existing flock definitions to the single file application, but we have not been able to see any simple restrictions that would allow similar arguments to be used for the single file. This seems to be mainly due to the strong property between pairs of entities that needs to be maintained for a single file.

More recently, further movement patterns have been studied. Jeung et al. [2008] modified the definition of a flock to what they call a convoy, using the notion of density connection [Ester et al., 1996]. Intuitively, two entities in a group are density-connected if a sequence of objects exists that connects the two objects and the distance between consecutive objects does not exceed a given constant. This would detect a convoy but also many other patterns, e.g., a flock would in most cases also be a convoy using this definition. Again, it is not obvious how this definition could be used to detect single file patterns.

## 3.5 A model for following behind behaviour

Intuitively, it is clear what is meant by one entity following another. However, it is not obvious how to formalise this notion. We now we give a definition for this which formalises the notion of following the track of the leader within a certain time frame.

**Definition** For defining following behind behaviour, we will use the following notation: $a$ denotes a moving entity and $f$ its parameterised trajectory. For positions in space we use $p,q$ and for points in time $s,t$. For a time interval we use $T$. In case of multiple objects, we index all these variables.

For the definition of following behind we fix three parameters $\tau_{\min}$, $\tau_{\max}$, and $\delta \in \mathbb{R}$ with $\tau_{\min} < \tau_{\max}$. The parameters $\tau_{\min}, \tau_{\max}$ specify minimum and maximum offsets in time and $\delta$ specifies a maximum offset in space. In the definition of following behind, we reparameterise one trajectory by a varying time difference in $[\tau_{\min}, \tau_{\max}]$, such that the trajectories are "similar" in the sense that at all times (except at the very beginning and end), the spatial positions of the two trajectories differ by at most $\delta$. The parameters $\tau_{\min}, \tau_{\max}, \delta$ shall be chosen depending on the application and exactness of the input and are discussed in more detail in Section 4.3.

Our definition of following behind is illustrated by the two trajectories in Figure 5. Assume that entity $a_i$ is given by the parameterised trajectory $f_i$ over the time interval $[s_1,t_2]$. Let $p_i$ be the position of entity $a_i$ at time $s_i$ and $q_i$ the position of entity $a_i$ at time $t_i$, respectively, for $i = 1,2$, where $s_1 < s_2 < t_1 < t_2$. Furthermore, let $s_2 \in [s_1+\tau_{\min}, s_1+\tau_{\max}]$ and $t_2 \in [t_1+\tau_{\min}, t_1+\tau_{\max}]$. Assume that the distances $d(p_1,p_2) \leq \delta$ and $d(q_1,q_2) \leq \delta$ are small and that $a_1,a_2$ are moving with a similar (e.g. constant) speed in between $p_1,p_2$ and $q_1,q_2$. Then we have that *at every point in time $s \in [s_2,t_1]$ entity $a_2$ is at a position which is $\delta$-close to the position of $a_1$ at an earlier point in time $s' \in [s-\tau_{\max}, s-\tau_{\min}]$.*

Formally, we define the notion of following behind as follows.

**Definition 2.** *Let $a_1$ be an entity with parameterised trajectory $f_1$ over the time interval $[s_1,t_1]$ and let $a_2$ be an entity with parameterised trajectory $f_2$ over the time interval $[s_2,t_2]$, where $s_2 \in [s_1+\tau_{\min}, s_1+\tau_{\max}]$ and $t_2 \in [t_1+\tau_{\min}, t_1+\tau_{\max}]$. Entity $a_2$ is following behind $a_1$ in the time interval $[s_1,t_1]$ if there exists a continuous, bijective function $\sigma\colon [s_1,t_1] \to [s_2,t_2]$ such that $\sigma(s_1) = s_2$ and*

$$\forall t \in [s_1,t_1]\colon \sigma(t) \in [t-\tau_{\max}, t-\tau_{\min}] \ \wedge\ d(f_1(\sigma(t)), f_2(t)) \leq \delta.$$

*As distance measure $d(\cdot,\cdot)$ in $\mathbb{R}^c$ we use the Euclidean distance.*

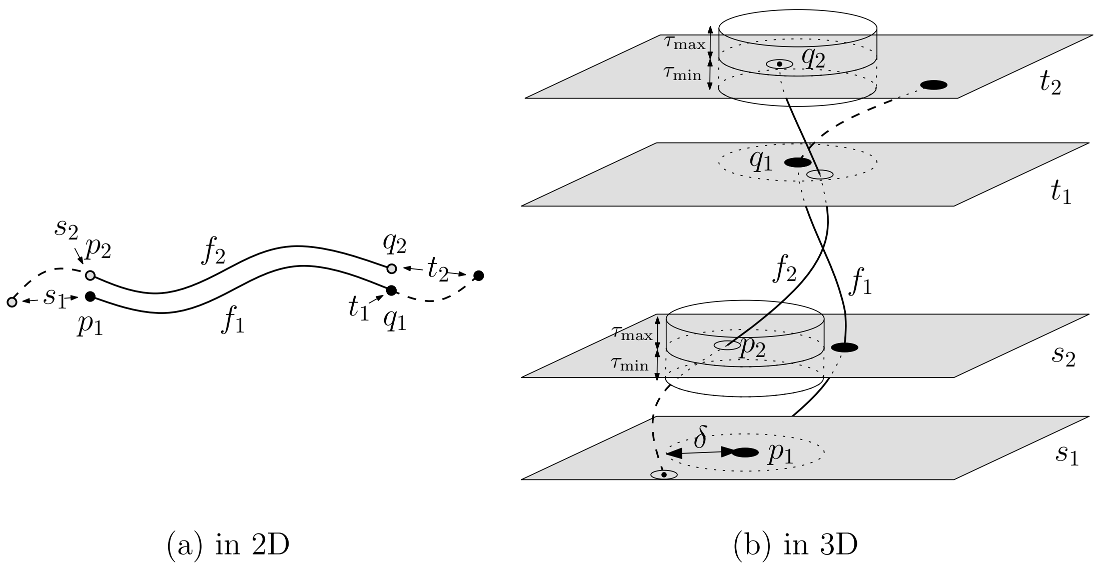

**Fig. 5.** Trajectories of two entities moving in single file.

We can reformulate this definition using a constrained free space diagram with a time and a distance constraint.

$$F_C(f_1, f_2) = \{(s,t) \,|\, d(f_1(s), f_2(t)) \leq \delta \wedge t - s \in [\tau_{\min}, \tau_{\max}]\}.$$

**Observation 1** *Entity $a_2$ is following behind $a_1$ in the time interval $[s_2, t_1]$ if the constrained free space diagram*

$$F_C(f_1, f_2) := \{(s,t) \,|\, t - s \in [\tau_{\min}, \tau_{\max}] \vee d(f_1(s), f_2(t)) \leq \delta\},$$

*contains a monotone path from $(s_1, s_2)$ to $(t_1, t_2)$.*

See Figure 6 for an illustration. In the following, we always use $F_C$ to denote this specific constrained free space diagram.

The time constraint $t - s \in [\tau_{\min}, \tau_{\max}]$ restricts us to what we call the $[\tau_{\min}, \tau_{\max}]$-*strip* of the free space, see again Figure 6. Since we only need to consider the $[\tau_{\min}, \tau_{\max}]$-strip of the free space diagram, our algorithms will have a running time that is linear in the number of cells within the strip, which for reasonable parameters is linear in $n$, in contrast to the quadratic complexity of the entire free space diagram.

Our definition of following behind does not give a strict order on the entities, i.e., it may be that by this definition both $a_1$ is following $a_2$ as well as $a_2$ is following $a_1$. This can have two reasons. First, it may be that the parameters $\tau_{\min}, \tau_{\max}, \delta$ are chosen inadequately, i.e., $\tau_{\min}$ is too small or $\delta$ is too big, which allows too large an offset. Or it may be that it is indeed not clear which entity is following which. For instance, imagine two entities standing still or moving back and forth along the same line. Also, consider the situations depicted in Figure 7. Figure 7 (a) clearly shows one entity following behind another and Figure 7 (b) shows two entities moving beside each other. However, where does this change, i.e., what about Figure 7 (c)?

We therefore consider the following strict order of following.

**Definition 3.** *Entity $a_2$ is* strictly following behind *$a_1$ if $a_2$ is following behind $a_1$ but $a_1$ is not following behind $a_2$.*

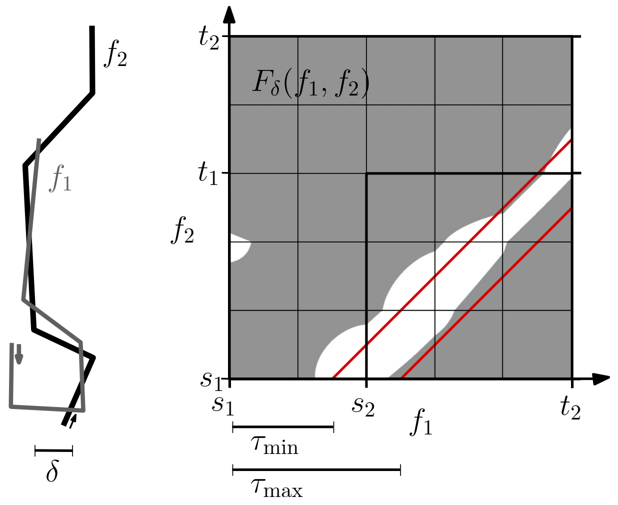

**Fig. 6.** Two parameterised trajectories $f_1$ and $f_2$ over the time interval $[s_1, t_2]$, a value $\delta > 0$, and the constrained free space diagram $F_C$.

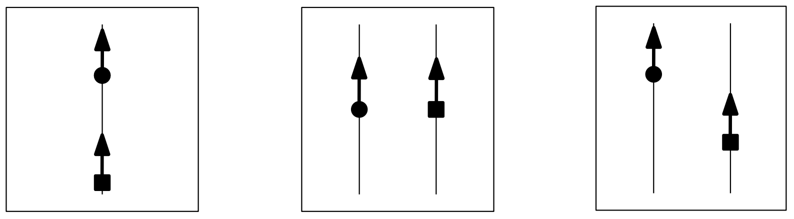
(a) following behind (b) beside each other (c) behind or beside?

**Fig. 7.** Different moving patterns

Thus, we only consider two entities to be following each other, if this holds only in one direction. This guarantees that two (or more) entities moving together as a group (or a flock) will not be considered to move as a single file.

**Travel condition** To see that it is reasonable to consider only strict following behaviour, we give Lemma 1, which states conditions under which two entities cannot follow each other. In particular, these conditions hold for natural cases of following, e.g., for two trajectories which follow the same simple curve with a time shift in $[\tau_{\min}, \tau_{\max}]$. In this case, the value $\delta$ in the lemma can be set to zero.

**Definition 4.** *An entity a with parameterised trajectory f fulfils the* travel condition *in the time interval T if*

$$\forall t \in T\colon\ f(t) \notin N_{2\delta}(f[t - 2\tau_{\max}, t - 2\tau_{\min}]),$$

*where* $N_\alpha(B) := \{x \mid \exists y \in B : d(x,y) \leq \alpha\}$ *denotes the* $\alpha$*-neighbourhood of the region B.*

Intuitively, the travel condition says that an entity travels a distance of at least $2\delta$ in a time interval of length $2\tau_{\min}$ and that it does not come close to a previous location before a time interval of length $2\tau_{\max}$ has elapsed. Note that the factor 2 in the definition is there only for technical reasons.

**Lemma 1.** *If entity $a_1$ or entity $a_2$ fulfils the travel condition, then they cannot both follow each other for a time interval of length $\tau_{\max}$.*

*Proof.* Assume the opposite, i.e., there exists $t \in T$ such that $a_1, a_2$ both follow each other for the time interval $[t, t+\tau_{\max}]$ and $a_1$ (or $a_2$) fulfil the travel condition. Then

$$\begin{aligned}
f_1(t+\tau_{\max}) \ &\in\ N_\delta(f_2[t, t+\tau_{\max}-\tau_{\min}]) \\
&\subseteq N_\delta(N_\delta(f_1[t-\tau_{\max}, t+\tau_{\max}-2\tau_{\min}])) \\
&= N_{2\delta}(f_1[t+\tau_{\max}-2\tau_{\max}, t+\tau_{\max}-2\tau_{\min}]) \\
&\quad \text{Contradiction!}
\end{aligned}$$

For the first line, we use that $a_1$ is following behind $a_2$ at time $(t+\tau_{\max})$. For the second line, we use that $a_2$ is following behind $a_1$ in the time interval $[t-\tau_{\max}, t+\tau_{\max}-2\tau_{\min}]$. In the third line, we get a contradiction to the travel condition of $a_1$. □

In practise, we may detect that $a_2$ is following $a_1$ as well as $a_1$ is following $a_2$. In this case, we might get a strict order by choosing a larger value for $\tau_{\min}$ or a smaller value for $\delta$. Or we may conclude that based on the input data we cannot decide if one is following the other. A further alternative is to use additional tests to check whether $a_1$ *is front of* $a_2$ at equal times.

Consider for instance the following model for *behind* and *in front of*. The line through $a_2$ orthogonal to its travel direction decomposes the plane into two half-spaces. We say $a_1$ *is front of* $a_2$, if $a_1$ is in the half-space in front of $a_2$. Likewise, we say that $a_1$ is *behind* $a_2$ if it is in the other half-space. Thus, to prevent false positives in the case that the travel condition does not hold, we can additionally test whether $a_1$ is in front of $a_2$ at equal times, or alternatively whether $a_2$ is behind $a_1$, or both. We do not include these tests in our general definition, because they fail in sharp turns. In our experiments, we will test the effect of adding these constraints.

## 4 Algorithms for single file movement

In this section we give algorithms for detecting single file behaviour using the constrained free space diagram that we defined for the following behind behaviour. Recall that a single file is a set of entities $a_1, \ldots, a_m$ where $a_i$ is following $a_{i+1}$ for $i = 1, \ldots, m-1$ during a time interval $[t_1, t_n]$. The problem of detecting a single file behaviour can be considered for several variants based on the following parameters:

- fixed or variable time interval,
- fixed or variable subset of entities, and
- fixed or variable order of the entities.

For fixed time, subset and order, we ask the question: Are the entities $a_1, \ldots, a_m$ moving in single file during the given time interval $[t_1, t_n]$ in the given order $1, \ldots, m$? In the case of a variable time interval, we are searching for subintervals of $[t_1, t_n]$ such that the entities are moving in single file during this subinterval. In the case of a variable subset of entities we are searching for subsets of the entities $a_1, \ldots, a_m$ which are moving in single file. In the case of a variable order, we are searching for a permutation $\pi$ of $1, \ldots, m$ such that the entities $a_{\pi(1)}, \ldots, a_{\pi(m)}$ are moving in single file in the order $\pi(1), \ldots, \pi(m)$.

We will give algorithms for all these variants. First, we give algorithms for the variants for two and then for several trajectories. The algorithms we give detect the (non-strict) following behaviour. If the travelling condition holds for the given trajectories, then this coincides with strict following. Otherwise the same algorithm can be run twice with swapped input trajectories to detect strict following.

## 4.1 Algorithms for two trajectories

We first give an algorithm for detecting the following behind behaviour of two trajectories for a fixed time interval and then generalise this algorithm for non-fixed time intervals.

**An algorithm for fixed time** According to Observation 1 we can detect whether one trajectory is following behind another during a fixed time interval by searching for a monotone path in the constrained free space diagram $F_C(f_1, f_2)$ of the trajectories. This is illustrated in Figure 6, which shows two trajectories $f_1, f_2$ and the constrained free space diagram $F_C(f_1, f_2)$ (here $\tau_{\min} = 1, \tau_{\max} = 2$). A monotone path in this strip corresponds to the reparameterisation $\sigma$ in our definition of following. More explicitly, the path is the graph of $\sigma$, i.e., it consists of the points $(t, \sigma(t))$. Let $a_1, a_2$ be entities with trajectories $f_1, f_2$ over the time intervals $[s_1, t_1]$ and $[s_2, t_2]$ as in Definition 2. We compute the *reachable free space* in the constrained free space diagram, i.e., all free space that is reachable by a monotone path from the point $(s_2, s_1)$. We do this in a similar manner as Alt and Godau [1995], which we briefly review here.

The constrained free space diagram can be computed column by column, and each column can be computed cell by cell from bottom to top. We call the reachable upper and right cell boundaries the *outgoing* boundaries of the cell, and the reachable lower and left cell boundaries its *ingoing* boundaries. We process only cells with non-empty ingoing boundary. At the beginning all ingoing boundaries are empty. For the lower-left cell we set the ingoing boundary to be the point $(s_2, s_1)$, if this point lies in free space. If $(s_2, s_1)$ is not in free space, the whole reachable free space is empty and the algorithm terminates. Then we compute for each reachable cell its upper and right cell boundary of reachable free space based on the previously computed lower and left cell boundary of reachable free space. For computing the outgoing boundaries, we first compute the free space boundaries. These can then be easily modified to the reachable cell boundaries as follows (cf. Figure 8).

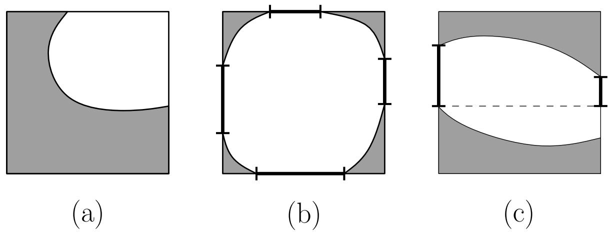

**Fig. 8.** Computing the outgoing cell boundaries based on ingoing cell boundaries. The outgoing cell boundaries are empty in (a), equal to the ingoing cell boundaries in (b), and modified from the free space boundaries in (c).

We do not process a cell, if both ingoing boundaries are empty, because then so are the outgoing boundaries (Figure 8 (a)). Now assume that the reachable bottom cell boundary is empty and the reachable left cell boundary is non-empty. Then the whole top cell boundary is reachable and the right cell boundary is reachable upward from the starting point of the left cell boundary (Figure 8 (c)). The case where the reachable bottom cell boundary is non-empty and the reachable left boundary is empty is analogous. If the two ingoing boundaries are both non-empty, then the outgoing boundaries equal the free space boundaries (Figure 8 (b)). For reachable cells which are on the boundary of the $[\tau_{\min}, \tau_{\max}]$-strip and therefore not completely contained in the strip, we also intersect the outgoing boundaries with the line $t - \tau_{\min}$ or $t - \tau_{\max}$, respectively.

We maintain an initially empty queue of cells that need to be processed. Now, if a cell has a right non-empty outgoing boundary, we add the cell to the right of it to the queue of cells that need to be processed. If a cell has an upper non-empty outgoing boundary, we process the cell above it next. We also check the queue for this cell and possibly remove it. If the upper outgoing boundary is empty, we process the next cell in the queue. At any time the queue only stores cells from two neighboring rows.

After computing the reachable free space in the $[\tau_{\min}, \tau_{\max}]$-strip in this way, we see if the point $(t_1, t_2)$ is reachable. This algorithm can be modified slightly to handle the case where only the start time for either $f_1$ or $f_2$ and the end time for either $f_1$ or $f_2$ is given. For instance, if both trajectories $f_1, f_2$ are given over the same time interval $[s_1, t_2]$, the algorithm can be modified to determine whether $s_2, t_1$ exist such that $a_2$ is following $a_1$ in the time interval $[s_2, t_1]$. For this, the algorithm would check whether the free space reachable from $[s_1 + \tau_{\min}, s_1 + \tau_{\max}] \times s_1$ contains any point in $t_2 \times [t_2 - \tau_{\max}, t_2 - \tau_{\min}]$.

The time and space requirements of the algorithm depend on the number of non-empty cells of the constrained free space diagram. In the following we use $k_{\text{avg}}$ and $k_{\max}$ to denote the average and maximum number of non-empty cells intersected by the $[\tau_{\min}, \tau_{\max}]$-strip per column of the constrained free space diagram. In the case of uniform time sampling $k_{\text{avg}}$ and $k_{\max}$ are bounded by $(\lceil \tau_{\max} - \tau_{\min} \rceil + i)$ with $i \in \{1, 2\}$.

**Theorem 1.** *For two entities with trajectories of complexity n each, we can determine in $O(nk_{avg})$ time using $O(k_{max})$ space whether one entity is following behind the other in a fixed time interval.*

*Proof.* The correctness of the algorithm follows directly from Observation 1. The running time is linear in the number of processed cells. These are the non-empty cells of the $[\tau_{\min}, \tau_{\max}]$-strip. The space needed corresponds to the maximum size of the queue. Since the queue never stores cells from more than two columns this is in $O(k_{\max})$. □

**An algorithm for variable time** For a variable time interval, we again search for monotone paths in the constrained free space diagram. Now, however, we are searching for paths which do not necessarily span the whole strip. An example is shown in Figure 9. For this, we sweep the $[\tau_{\min}, \tau_{\max}]$-strip with a similar approach as Buchin et al. [2008]. As a sweep line we use a vertical line which moves along the vertices of $f_1$, i.e., it moves from column boundary to column boundary. We again compute the free space in the $[\tau_{\min}, \tau_{\max}]$-strip column by column and in each column cell by cell from bottom to top. In contrast to the previous algorithm, we compute the entire free space in the $[\tau_{\min}, \tau_{\max}]$-strip and not only the free space reachable from a certain point or interval. Furthermore, we label all cell boundaries of the free space with the smallest $x$-value from where the interval is reachable by a monotone path in the $[\tau_{\min}, \tau_{\max}]$-strip. We propagate this information cell by cell. At each event of the sweep, i.e., for each column, we check whether a maximal monotone path ends in this column. The duration of the path is stored in the edge label. Next we describe how to compute the labelled cell boundaries and then how to check which maximal paths end in a column.

In each cell, we need to compute the labelled upper and right cell boundary based on the previously computed labelled lower and left cell boundary. For the first column, the bottom-most and all left cell boundaries are empty except for the point $(s_2, s_1)$ which is labelled $s_1$. In the case where only the start point $s_1$ of $f_1$ is given, all non-empty left cell boundaries are labelled with $s_1$. Similarly, if only the start point $s_2$ of $f_2$ is given, then all left cell boundaries of the first column are empty, but we add the non-empty horizontal cell boundaries in $[s_2 + \tau_{\min}, s_2 + \tau_{\max}]$, which are labelled by their respective starting points.

Similar to the previous algorithm, we call the labelled bottom and left cell boundaries of a cell its *ingoing* boundaries and the labelled upper and right cell boundary its *outgoing* boundaries. In each cell the free space may *start, continue,* or *end*. Typical situations for these three cases are shown in

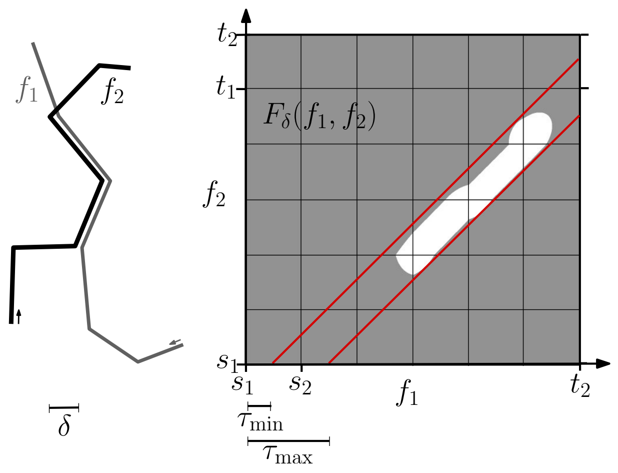

**Fig. 9.** Two parameterised trajectories $f_1, f_2$ over the time interval $[s_1, t_2]$, a value $\delta > 0$, and the free space diagram $F_\delta(f_1, f_2)$ with $[\tau_{\min}, \tau_{\max}]$-strip. Here, the two entities are following each other only for a part of $[\tau_{\min}, \tau_{\max}]$-strip.

Figure 10 (a)-(c). If a cell has empty ingoing and non-empty outgoing cell boundaries, then free space starts in this cell (Figure 10 (a)). Analogously, if the ingoing boundaries are non-empty and the outgoing boundaries are empty, then free space ends in this cell (Figure 10 (c)). If both the in- and outgoing boundaries are non-empty, free space continues (Figure 10 (b)). Also, free space may start or end as well as continue in a cell (Figure 10 (d)).

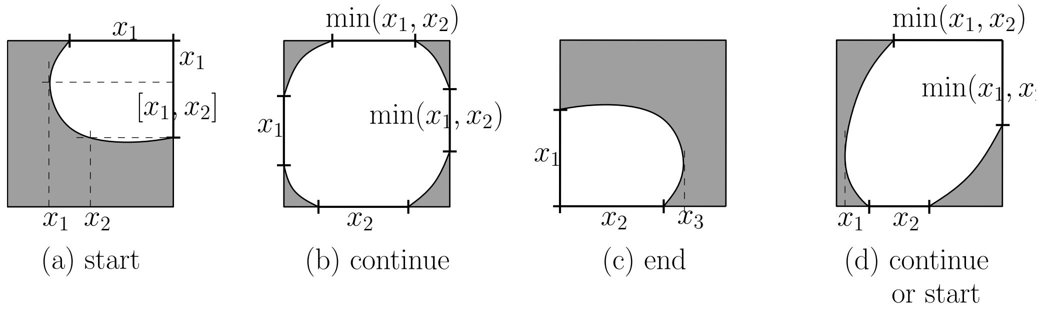

**Fig. 10.** Examples for free space (a) starting, (b) continuing, or (c) ending in a cell. This may also occur mixed as in (d).

The outgoing boundaries are the free space boundaries with labels based on the ingoing boundaries. The labels are illustrated in Figure 10. For each cell we compute its free space boundaries and then label the boundaries based on the ingoing boundaries. If free space starts or ends, then we also compute the maximal or minimal $x$-value in free space of that cell, e.g., $x_1$ in Figure 10 (a) and $x_3$ in Figure 10 (c). If free space continues and both ingoing boundaries are non-empty, we propagate the smaller $x$-value of the ingoing edges, as in Figure 10 (c). In a cell where free space starts or continues, we may have to split an interval once, as in Figure 10 (a), to give it two distinct labels. The splits propagate to the right in the same column. However, since we only add one split per cell, we get at most $2k_{\max}$ subintervals per cell.

While computing a column, we maintain the longest maximal path continuing or ending in this column. We do this by maintaining the starting $x$-value of this path and the information whether it is an ending or continuing path. These values are initiated in the first cell of the column containing continuing or ending free space. In later cells with continuing or ending free space we update these values if the new starting $x$-value is smaller. After computing a column, we output the maximal path found if this is a path ending in this column.

**Theorem 2.** *For two trajectories of complexity n each, we can determine in $O(nk_{avg}^2)$ time using $O(n+k_{max}^2)$ space during which time intervals one trajectory is following behind the other.*

*Proof.* The correctness follows again from Observation 1. The running time is linear in the number of event points of the sweep, since the cost at each event point is constant. □

## 4.2 Algorithms for several trajectories

We now give algorithms for detecting single file behaviour in a set of trajectories.

**An algorithm for fixed time, number and order** For fixed time and ordered sequence of trajectories, we can simply check each incident pair in the sequence using the algorithm for two trajectories and fixed time interval. From Theorem 1 we get the following corollary.

**Corollary 1.** *For m trajectories of complexity n each, we can check in $O(mnk_{avg})$ time using $O(nm+k_{max})$ space whether they are moving in single file for a given order and time.*

**An algorithm for fixed time, variable order and subset** For each pair of trajectories, we can check whether they are moving in single file using the algorithm for two trajectories over a fixed time interval. We represent this information as a directed acyclic graph, as shown in Figure 11(a). That is, we build the graph $G=(V,E)$ of vertices $V=\{a_1,\ldots,a_m\}$ and directed edges $E=\{(a_i,a_j)\,|\,a_i$ is following $a_j,\ i,j\in\{1,\ldots,m\})\}$. Each maximal directed path in this graph corresponds to a maximal single file. In particular, the single file consisting of the most entities corresponds to the longest path in the graph. We can find all these paths in the graph in time linear in the size of the graph, i.e., $O(m^2)$, using for instance depth-first-search.

**Theorem 3.** *For m trajectories of complexity n each, we can detect in $O(m^2nk_{avg})$ time using $O(nm+m^2+k_{max})$ space all single file behaviours for a given time interval.*

*Proof.* The correctness follows from the correctness of the algorithm for two trajectories, i.e., Theorem 1, and by the definition of the graph $G$. The time needed to construct $G$ is $m^2$ times the time needed for the algorithm to handle two trajectories, that is $O(m^2nk_{\text{avg}})$. □

If we want to report all maximal directed paths in the graph this can be done in $O(m^2)$ time.

**An algorithm for variable time, order and subset** For each pair of trajectories, we determine during which time intervals they are following each other using the corresponding algorithm for two trajectories. We represent this as a directed, labelled graph, where each edge is labelled by the corresponding time intervals, see Figure 11(b). That is, we have the directed edge $(a_i,a_j)$ in the graph if $a_i$ is following $a_j$ during some time interval. This edge is labelled with all time intervals during which $a_i$ is following $a_j$. Thus, the space requirement of the graph is now $O(m^2n)$, since each edge may be labelled with up to $n$

time intervals. For any given time interval, we can search for paths in the corresponding subgraph, that is, we use only the edges labelled with this time interval. Note that the full graph may be cyclic, but for each fixed time interval it is acyclic for the strict following behaviour or assuming that all entities fulfil the travel condition.

Based on the graph $G$ we can answer several different single file queries. It therefore makes sense to split these two steps of the algorithm: computing $G$ and searching for paths in $G$. In fact, $G$ gives a compact representation of all single file behaviours and we can use it as a data structure for answering single file queries in time linear in the size of $G$, which is $O(m^2)$.

If we want to report all single file behaviours which occur we can do this (naïvely) by searching the graph for all $O(n^2)$ possible time intervals.

**Theorem 4.** *For m trajectories of complexity n each, we can construct in $O(m^2 n k_{avg}^2)$ time using $O(m^2 n + k_{max})$ space a data structure which can answer single file queries for an arbitrary time interval in $O(m^2)$ time.*

*Proof.* As in the previous proof, the correctness and running time follow from the correctness and running time of the algorithm for two trajectories and the graph searching. □

If we want to search for all single file behaviours at variable times in a set of trajectories, we have to query $O(n^3)$ possible starting times, since each of the $O(n^2)$ pairs of trajectories has $O(n)$ possible starting times for the following behind behaviour. In practise, a more efficient approach is to query only for $O(n)$ uniformly sampled starting points over the whole time interval. The error of this approach is only twice the length of the uniform time interval.

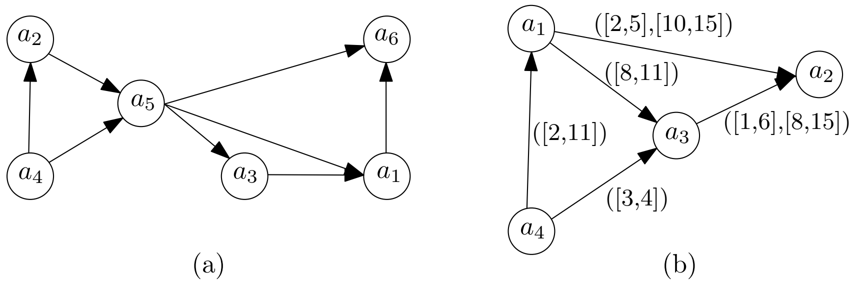

**Fig. 11.** (a) The path $a_4a_2a_5a_3a_1a_6$ in $G$ corresponds to a single file with these entities. (b) In the variable time case the graph $G$ represents all single file patterns, e.g., entities $a_4a_1a_3a_2$ form a single in the time interval $[8,11]$ and entities $a_4a_1a_2$ form a single file in the time interval $[2,5]$.

## 4.3 Experiments

We tested the effectiveness of our model and the efficiency of our algorithms for detecting single file movement on a set of real and generated data. The running times and effectiveness of all our algorithms are directly based on the running times and effectiveness of the algorithms for two trajectories. We therefore performed most of the tests on two trajectories.

For variable time intervals we implemented and tested a simplified algorithm. Instead of sweeping the constrained free space diagram, as described in Section 4.1, we loop over possible starting points. We begin with the first possible start point. For each start point, we traverse the $[\tau_{\min}, \tau_{\max}]$-strip for the longest monotone path starting at the current start point. We report this as single file behaviour if the duration is larger than a chosen threshold and the entities move at least a chosen distance from the current start point. If it is reported, we continue by using the next time stamp after the end point of the path as new starting point. Otherwise we use the next time stamp after the current start point as new start point. This simplified algorithm does not find all maximal monotone paths as the algorithm in Section 4.1 does, i.e., the simplified algorithm does not find a monotone path overlapping in time with an already reported path. It does, however, sweep the whole $[\tau_{\min}, \tau_{\max}]$-strip and gives a classification of each time interval in *following* or *not following*. We use this classification for testing the effectiveness of our model.

We tested the effectiveness of our model using a small set of real data. For testing the efficiency of the algorithms we generated a larger set of artificial data. The time measurements of the experiments were performed on an Intel(R) Pentium(3) 3CPU 3.00GHz 1GB RAM using the Visual C++ 2008 compiler.

**Test on real data** The set of real data was generated by two cyclists carrying data loggers. The data loggers used an MTK chipset with 32 channel tracking. In our experiments, errors of up to 30 meters occurred. Time-space positions were logged every second while the cyclists moved with an average speed of about 3 to 4 meters per second. The average time difference between the cyclists was about 2 to 4 seconds. This relatively large distance was chosen because of the large errors in the logged data. For higher accuracy, the data was manually post-processed s.t. it contained only errors of up to approximately 15 meters. We note that in the future, and already now in many applications, trajectory data with higher accuracy will be available. For example in sports, trajectory data with errors less than half a meter is used. This enables better results in detecting single file behaviour. The set of real data consisted of five tests, each consisting of two tracks with between 690 and 990 data points each. The tests contained intervals of following and non-following behaviour.

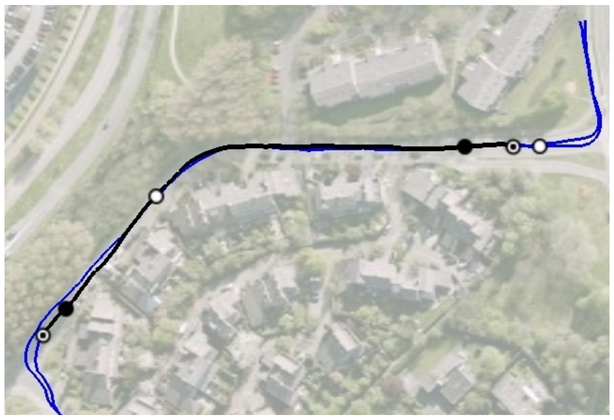

**Fig. 12.** A following behaviour in the test on real data for $\tau_{\min} = 4$ and $\delta = 12$. The white/black dots indicate extra waypoints set by the leader/follower indicating start and end of the following behaviour. The circled dot shows the detected following behaviour.

To test the effectiveness of the model, the cyclists set extra waypoints while generating the tracks, which indicated starting and ending points of all following behaviours. Figure 12 shows an example of a following behaviour in one of the tests. The trajectories start in the lower right corner. The dots all lie on the followers trajectory. The first white dot marks the signal of the leader for beginning the following behaviour. The next black dot marks the signal of the follower that the following behaviour has been established. The circled dot in between marks the start of the detected following behaviour. At the end of the follower behaviour, again first the leader signalled the ending, which was confirmed by the follower. The circled dot marks the end of the detected following behaviour, which in this case is slightly too late.

The results for real data are shown in Table 1. We here set $\tau_{\max} = 10$ meters, the minimum duration for a following pattern to 20 seconds, and the minimum distance to be travelled during that time to 40 meters. Note that the exact choice of these values had little effect on the detection rate. For different values of $\delta$ and $\tau_{\min}$, the percentage of correctly detected following and non-following behaviour is shown. For each $(\delta, \tau_{\min})$ pair the top value gives the percentage of time intervals in which a following behaviour was correctly detected. The bottom value gives the percentage of time intervals in which a non-following behaviour was correctly detected. The correctness was based on the waypoints set by the cyclists. For this, all time intervals between the leader and the follower signal for the start and end of a following behaviour were not classified and did not contribute to the percentages of correctly detected intervals. For instance in Figure 12 all time intervals of following were correctly identified, and a few intervals of non-following were incorrectly identified as following.

| $\tau_{\min}$ | $\delta$ | | | | | | |
|---|---|---|---|---|---|---|---|
| | 6 | 8 | 10 | 12 | 14 | 16 | 18 |
| 1 | 84.3 | 94.3 | 97.1 | 99.3 | 99.5 | 99.9 | 100 |
| | 73.3 | 64.1 | 55.6 | 48.0 | 44.3 | 37.6 | 30.3 |
| 2 | **80.7** | 90.0 | 94.0 | 96.7 | 98.0 | 99.3 | 99.5 |
| | **91.4** | 79.3 | 65.2 | 59.4 | 54.4 | 45.2 | 39.8 |
| 3 | 72.4 | **83.9** | 89.5 | 92.1 | 93.6 | 96.8 | 98.1 |
| | 97.6 | **92.0** | 85.5 | 73.3 | 63.9 | 57.1 | 50.8 |
| 4 | 58.7 | 71.3 | **81.0** | **85.6** | 90.1 | 92.4 | 93.7 |
| | 99.3 | 95.0 | **92.0** | **89.4** | 83.4 | 69.0 | 61.5 |
| 5 | 34.7 | 56.2 | 67.2 | 74.8 | **82.1** | 85.7 | 90.9 |
| | 99.7 | 96.5 | 95.2 | 92.3 | **91.7** | 88.7 | 77.5 |
| 6 | 14.4 | 31.3 | 42.6 | 58.8 | 68.1 | **75.1** | **83.2** |
| | 98.3 | 96.9 | 96.7 | 95.8 | 93.2 | **92.5** | **90.1** |

**Table 1.** Test on real data. The percentage of correctly detected following behaviour in dependence on $\delta$ and $\tau_{\min}$ is shown. The top and bottom values give the percentage of correctly detected following and non-following behaviour, respectively. The bold values indicate where the travel condition is likely to be fulfilled.

In Table 1 the bold values indicate for each $\delta$ value the smallest value of $\tau_{\min}$ for which the travel condition is likely to be fulfilled. For instance for $\delta = 12$ meters and assuming a velocity of 3 meters per second, the trajectory is expected to leave the $\delta$-neighbourhood of a point after approximately 4 seconds. Note that for values of $\delta$ and $\tau_{\min}$ where the travel condition is likely to be fulfilled, the percentages of correctly identified non-following behaviours is large.

In the cases, where the travel condition does not hold the percentage of correctly recognized non-following behaviour drops considerably. The reason for this is that in such a case following behaviour might be detected for an entity, if it is not even behind the other entity, but beside it. As discussed in Section 3.5, we can in such a situation additionally test, whether for the duration of the possible pattern the leader is in front of the follower. Recall that the three discussed models were that 1.) B is in the half-plane in front of A, 2.) A is in the half-plane behind B, and 3.) both 1.) and 2.) hold. For instance then for $\tau_{\min} = 2$ and $\delta = 12$ the percentages for the three models are 90.9/80.9, 90.2/80.3, and 89.7/81.8. So all three models prevent false positives in this situation. In a case where the travel condition is fulfilled like

$\tau_{\min} = 2$ and $\delta = 6$, the models do not influence the detection rate much, the percentages are 80.1/93.4, 78.5/93.1, and 78.5/93.5.

**Test on generated data** The artificial data was generated by first generating a leader track and based on this follower tracks. The leader track was generated as a constrained random walk Turchin [1998]. It moves from a given starting point for $n$ time stamps. In each step, the next point is generated randomly using given bounds on the velocity, acceleration, and turning angle. The follower track is generated from the leader track by offsetting the leader track by a given time difference $\tau$. Also, all distances are offset at random by at most a given distance $\delta/2$. An example of a leader and several follower tracks generated in this way is shown in Figure 13.

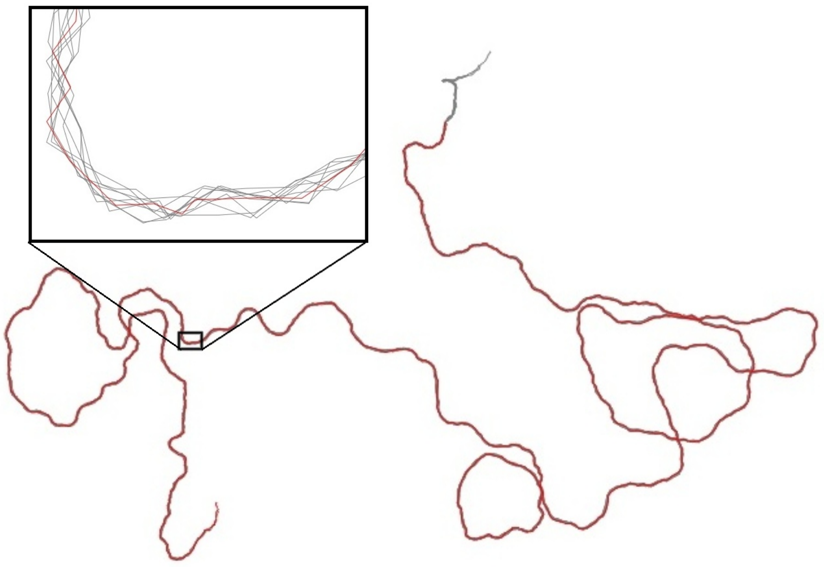

**Fig. 13.** Example of a generated leader and several follower tracks.

For the time measurements we generated follower/leader tracks of up to 10 million data points. Figure 14 shows the time measurements in dependency on the number of data points for different values of $k_{\text{avg}}$. Since the trajectories were uniformly sampled we have $k_{\text{avg}} = \tau_{\max} - \tau_{\min}$. As expected, the running times are linear in the number of data points. The dependency of the running time on $k_{\text{avg}}$ is better than the worst case linear bound, since for large $k_{\text{avg}}$ only a small part of the $[\tau_{\min}, \tau_{\max}]$-strip is non-empty.

**Choosing the parameters** As our experiments confirm, choosing the values of $\delta, \tau_{\min}$, and $\tau_{\max}$ is crucial. Choosing $\delta$ or $\tau_{\max}$ too small or $\tau_{\min}$ too large increases the number of unrecognised following behaviours. And choosing $\delta$ or $\tau_{\max}$ too large or $\tau_{\min}$ too small increases the number of incorrectly identified following behaviours. Choosing a large $\tau_{\max}$ in a single file of $k > 2$ entities leads to the situation illustrated in Figure 15(b). Not only the correct following behind behaviours $a_{i+1} \rightarrow a_i$ are detected ($i = 1, \ldots, m$) but also several following behind patterns $a_i \rightarrow a_j$ for $1 \leq i < j \leq m$. Although these are correct following behind behaviours, these are typically not of interest, in particular as they are implied by the following behind behaviours $a_{i+1} \rightarrow a_i$ for $i = 1, \ldots, m$.

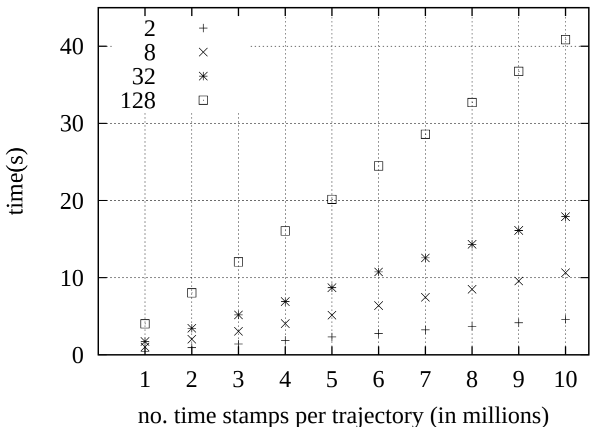

**Fig. 14.** The running times of the algorithm on generated data for $\tau_{\max} - \tau_{\min} = 2, 8, 32, 128$.

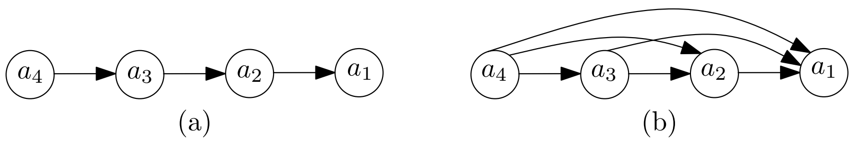

**Fig. 15.** (a) Only the single file $a_1a_2a_3a_4$ is detected. (b) Further following behaviours, e.g., $a_4 \rightarrow a_2$, are detected.

## 5 Conclusion

The main contribution of this paper is the introduction of the *constrained free space diagram*, an extension of the geometric *free space diagram* [Alt and Godau, 1995] which has been used extensively in curve analysis. We show that the constrained free space diagram can be used to model many common constraints such as distance, difference in time, speed or direction.

The constrained free space diagram is a general tool and can be used for many applications without much modification. However, to make it effective requires expert domain knowledge, and to make it efficient requires a good understanding of the constrained free space diagram.

To make it effective it is crucial to choose the right constraints. Furthermore, it is important, that the parameters are set appropriately otherwise one would either detect too many false positives, or no patterns at all. In our experiments we determined suitable parameters based on knowledge about the trajectories and adjusted them empirically. It would be interesting to investigate whether this process can be semi-automated. A possible approach is to evaluate the interestingness of the pattern for various settings of the parameters on synthetic data [Laube and Purves, 2006].

To make the algorithm efficient one needs to consider the structure in more detail. In general the time and space complexity for a free-space diagram is $n^2$ times the complexity of one cell, which can be very large for big instances. In the constrained case this can in many cases be improved upon by taking the constraints into consideration when building the constrained free space diagram. For example, for *following behind* the number of cells considered by our algorithm is $O(nk_{\text{avg}})$, where $k_{\text{avg}}$ is the average number of "active" cells which in many applications is very small compared to $n$.

We illustrated the utility of this structure by giving several examples among them most prominently the single file pattern. For detecting the single file pattern we give the first algorithms, to the best of our knowledge. The initial experiments are promising and we believe the experiments prove the usefulness of the constrained free space diagram for this application. However, the experiments also show that other approaches might be suitable for detecting single file movements. During this research we tried several other approaches, mainly focussing on modifying existing definitions of flocks, convoys and leadership patterns. All of these approaches assume that entities follow each other closely in time and space. In contrast, our approach can easily handle time gaps.

In specific applications further information may be available, that we can use to speed up the algorithm. For example, if the leader or other positions in the order are known, then this limits the number of single file behaviours, which in turn decreases the computation time. We define the single file pattern as a sequence of following behind patterns, which requires checking every pair of trajectories. Is this necessary or is there a reasonable definition of a single file pattern for which checking a linear number of pairs would suffice?

## Acknowledgements

K. Buchin was funded by the Netherlands Organisation for Scientific Research (NWO) under project number 639.022.707 and BRICKS/FOCUS grant number 642.065.503. M. Buchin was funded by the German Research Foundation (DFG) under grant number BU 2419/1-1 and by the Netherlands Organisation for Scientific Research (NWO) under BRICKS/FOCUS grant number 642.065.503. National ICT Australia (NICTA) is funded by the Australian Government as represented by the Department of Broadband, Communications and the Digital Economy and the Australian Research Council through the ICT Centre of Excellence program.

# Bibliography

P.K. Agarwal, S. Har-Peled, N.H. Mustafa, and Y. Wang. Near-linear time approximation algorithms for curve simplification. *Algorithmica*, 42(3-4):203–219, 2005.

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. In *Proc. 21st International Conference on Very Large Data Bases (VLDB)*, pages 490–501, 1995.

H. Alt and M. Godau. Computing the Fréchet distance between two polygonal curves. *International Journal on Computational Geometry and Applications*, 5:75–91, 1995.

H. Alt and L. Guibas. Discrete geometric shapes: Matching, interpolation, and approximation. In J.-R. Sack and J. Urrutia, editors, *Handbook of Computational Geometry*, pages 121–153. Elsevier, 1999.

H. Alt, B. Behrends, and J. Blömer. Approximate matching of polygonal shapes. *Annals of Mathematics and Artificial Intelligence*, 3–4:251–265, 1995.

H. Alt, C. Knauer, and C. Wenk. Comparison of distance measures for planar curves. *Algorithmica*, 38 (2):45–58, 2004.

M. Andersson, J. Gudmundsson, P. Laube, and T. Wolle. Reporting leaders and followers among trajectories of moving point objects. *GeoInformatica*, 12(4):497–528, 2008.

M. Benkert, B. Djordjevic, J. Gudmundsson, and T. Wolle. Finding popular places. In *Proc. 18th International Symposium on Algorithms and Computation (ISAAC)*, pages 776–787, 2007.

M. Benkert, J. Gudmundsson, F. Hübner, and T. Wolle. Reporting flock patterns. *Computational Geometry – Theory & Applications*, 41(3):111–125, 2008.

K. Buchin, M. Buchin, J. Gudmundsson, J. Luo, and M. Löffler. Detecting commuting patterns by clustering subtrajectories. In *Proc. 19th International Symposium on Algorithms and Computation (ISAAC)*, pages 644–655, 2008.

K. Buchin, M. Buchin, and Y. Wang. Exact algorithms for partial curve matching via the Fréchet distance. In *Proc. 19th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 645–654, 2009a.

Kevin Buchin, Maike Buchin, Marc Van Kreveld, and Jun Luo. Finding long and similar parts of trajectories. In *Proc. 17th Annu. ACM Int. Symp. on Advances in Geographic Information Systems (ACM GIS'09)*, 2009b. To appear.

A.F. Cook IV and C. Wenk. Geodesic Fréchet distance with polygonal obstacles. Technical Report CS-TR-2008-0010, University of Texas at San Antonio, 2008a.

A.F. Cook IV and C. Wenk. Min-link shortest path maps and Fréchet distance. Technical Report CS-TR-2008-0011, University of Texas at San Antonio, 2008b.

Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proc. 2nd International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 226–231, 1996.

F. E. Fish. Energetics of swimming and flying in formation. *Comments on Theoretical Biolog*, 5:283–304, 1999.

J. Gudmundsson and M. van Kreveld. Computing longest duration flocks in trajectory data. In *14th ACM International Symposium on Geographic Information Systems (ACM GIS)*, pages 35–42, 2006.

J. Gudmundsson, M. van Kreveld, and B. Speckmann. Efficient detection of motion patterns in spatio-temporal data sets. *GeoInformatica*, 11(2):195–215, 2007.

C. S. Jensen, D. Lin, and B. C. Ooi. Continuous clustering of moving object. *IEEE Transactions on Data Engineering*, 19(9):1161–1174, 2007.

H. Jeung, M. L. Yiu, X. Zhou, C. S. Jensen, and H. T. Shen. Discovery of convoys in trajectory databases. *Proc. VLDB Endowment*, 1(1):1068–1080, 2008.

P. Kalnis, N. Mamoulis, and S. Bakiras. On discovering moving clusters in spatio-temporal data. In *Proc. 9th International Symposium on Advances in Spatial and Temporal Databases (SSTD)*, pages 364–381, 2005.

K. Kalpakis, D. Gada, and V. Puttagunta. Distance measures for effective clustering of ARIMA time-series. In *Proc. 1st IEEE Internat. Conference on Data Mining*, pages 273–280, 2001.

P. Laube, M. van Kreveld, and S. Imfeld. Finding REMO – detecting relative motion patterns in geospatial lifelines. In *Proc. 11th International Symposium on Spatial Data Handling (SDH)*, pages 201–214, 2004.

Patrick Laube and Ross S. Purves. An approach to evaluating motion pattern detection techniques in spatio-temporal data. *Computers, Environment and Urban Systems*, 30(3):347–374, 2006.

J.-G. Lee, J. Han, and K.-Y. Whang. Trajectory clustering: a partition-and-group framework. In *Proc. ACM SIGMOD International Conference on Management of Data*, pages 593–604, 2007.

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.

G. Trajcevski, H. Ding, P. Scheuermann, R. Tamassia, , and D. Vaccaro. Dynamics-aware similarity of moving objects trajectories. In *Proc. 15th Annu. ACM Int. Symp. on Advances in Geographic Information Systems (ACM GIS'07)*, pages 75–82, 2007.

Y. Tremblay, S. A. Shaffer, S. L. Fowler, C. E. Kuhn, B. I. McDonald, M. J. Weise, C.-A. Bost, H. Weimerskirch, D. E. Crocker, M. E. Goebel, and D. P. Costa. Interpolation of animal tracking data in a fluid environment. *Journal of Experimental Biology*, 209:128–140, 2006.

P. Turchin. *Quantitative Analysis of Movement: Measuring and Modelling Population Redistribution in Animals*. Sinauer Associates, 1998.

R. C. Veltkamp and M. Hagedoorn. State-of-the-art in shape matching. In M. Lew, editor, *Principles of Visual Information Retrieval*, pages 87–119. Springer, 2001.

F. Verhein and S. Chawla. Mining spatio-temporal association rules, sources, sinks, stationary regions and thoroughfares in object mobility databases. In *Proc. 11th International Conference on Database Systems for Advanced Applications (DASFAA)*, pages 187–201, 2006.

M. Vlachos, D. Gunopulos, and G. Kollios. Discovering similar multidimensional trajectories. In *Proc. 18th Internat. Conference on Data Engineering*, pages 673–684, 2002.