

Detecting Single File Movement

[Extended Abstract] *

K. Buchin
Dept. Computer Science
Utrecht University
The Netherlands
buchin@cs.uu.nl

M. Buchin
Dept. Computer Science
Utrecht University
The Netherlands
maike@cs.uu.nl

J. Gudmundsson
NICTA[†] Sydney
Australia
joachim.gudmundsson@
nicta.com.au

ABSTRACT

We study the problem of detecting a single file behavior in a set of trajectories. A group of entities is moving in single file if they are following each other, one behind the other. This movement pattern occurs often, among animals, humans, and vehicles. It is challenging to detect because it does not have a fixed layout.

In this paper we first model the notion of following behind, on which we base our definition of single file. We present efficient algorithms for detecting following behind and single file behaviors. We test and evaluate these algorithms on real and generated test data.

Categories and Subject Descriptors

H.4 [Information Systems Applications]: Miscellaneous—*Geographic Information Systems*; F.2.m [Analysis of Algorithms]: Miscellaneous

General Terms

Algorithms

Keywords

Trajectories, Movement Patterns

1. INTRODUCTION

Due to technological advances in location-aware devices, trajectory data is becoming more and more available. To analyze large amounts of trajectory data, automated methods are needed. An important analysis task is to find *movement patterns* in a set of trajectories. A frequently occurring

[†]NICTA is funded through the Australian Government's Backing Australia's Ability initiative, in part through the Australian Research Council.

*This research was funded by DMIST/NICTA and the Netherlands Organisation for Scientific Research (NWO) under BRICKS/FOCUS grant number 642.065.503.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

ACM GIS '08, November 5-7, 2008, Irvine, CA, USA
Copyright 2008 ACM ISBN 978-1-60558-323-5/08/11 ...\$5.00.

movement pattern is a *single file*, i.e., 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 [5].

A single file differs from other movement patterns in that it does not have a fixed layout and that it requires a relation between every pair of entities to be maintained at each time steps, as illustrated in Figure 1. This makes it particularly challenging to compute compared to many other movement patterns. For example, trendsetting [12] and leadership [2] patterns are similar to the single file pattern but only require the entities to follow the leader (or trendsetter). Another example is the flock pattern [8, 9, 11] which can be seen as a cluster of entities at each time step (either with a fixed subset or varying subset) and hence there is no special relationship between two entities in a flock that must be maintained.

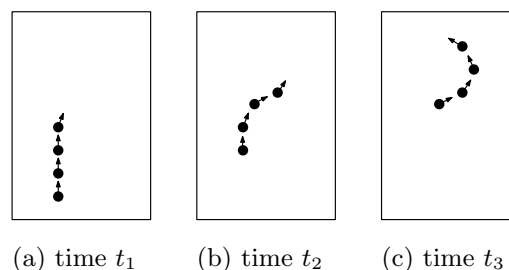


Figure 1: The layout of a single file typically changes over time.

The problem of detecting movement patterns in spatio-temporal data has recently received considerable attention from several research communities, e.g., geographic information science [14] (see also the survey [6]), data mining [15], data bases [10] and algorithms [4]. One of the first movement pattern studied [9, 10, 11] was moving clusters. It is not surprising that this was one of the first patterns to be studied since it is the spatio-temporal equivalence of a point cluster in the spatial setting. A moving cluster in a time interval T consists of at least m entities such that for every point in time within T there is a cluster of m entities. Note that the set of entities might be much larger than m , thus

entities may join and leave a cluster during the cluster’s lifetime. A moving cluster is sometimes also called a *variable subset flock*. Closely related to moving clusters is the *flock* pattern, or *fixed subset flock*. This problem has been studied in several papers [3, 7, 12]. Even though different papers use different definitions the main idea is that a flock consists of a fixed set of entities moving together as a cluster at all times.

More recently, further movement patterns have been studied. Jeung et al. [10] modified the definition of a flock to what they call a convoy. 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. Gudmundsson et al. [6] developed approaches to detect leadership patterns. They proposed a definition of a pattern to be the geometrical relation of one individual moving in front of its followers such that all the followers can ‘see’ the leader. They gave an $O(m^2n \log m)$ time algorithm to detect all leadership patterns of a pre-defined minimum length, among a set of m trajectories over n time-steps.

In this paper we discuss the automatic detection of a single file movement pattern. For this, we first develop a mathematical model for the notion of *following behind*. We then give algorithms for detecting a single file based on this model. These algorithms are also evaluated experimentally.

1.1 Problem Definition

Let a spatio-temporal trajectory τ of a moving entity a be given by n time-space positions. That is, $\tau = ((t_1, p_1), \dots, (t_n, p_n))$, where $p_i \in \mathbb{R}^c$ for some $c > 0$ (typically $c = 2$ or $c = 3$) gives the position of entity a at time t_i for $i = 1, \dots, n$. We assume that in between time stamps t_i and t_{i+1} the entity a moves with constant speed along a straight line from p_i to p_{i+1} for $i = 1, \dots, n$. For detecting a single file behavior, we are given m spatio-temporal trajectories τ_1, \dots, τ_m of entities a_1, \dots, a_m .

We say that the entities a_1, \dots, a_m are moving in single file for a given time interval if during this time each entity a_{j+1} is following behind entity a_j for $j = 1, \dots, m - 1$ (see the next section for our definition of *following behind*).

The problem of detecting a single file behavior can be considered for several variants based on the following parameters:

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

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

Next we state our definition of following behind. Then, in Section 3, we develop algorithms for detecting single file behavior. The model and algorithms are experimentally evaluated in Section 4.

2. DEFINITION FOR FOLLOWING

We will define the notion of following behind using parameterizations and re-parameterizations of the trajectories. A natural parameterization of a trajectory τ at n time-space positions $((t_1, p_1), \dots, (t_n, p_n))$ is the function $f: [t_1, t_n] \rightarrow \mathbb{R}^c$ where $f(t)$ gives the position of a at time t . We use parameterizations 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 $f(t)$ moves along the trajectory from start to end. A re-parameterization of f is a continuous, bijective function $\sigma: [t_1, t_n] \rightarrow [t_1, t_n]$ with $\sigma(t_1) = t_1$. Note that this implies $\sigma(t_n) = t_n$, as well.

We will always use a to denote a moving entity, τ for its trajectory given as sequence of time-space-positions and f for the parameterized trajectory. For positions in space we use p, q and for points in time s, t . For a time interval we will use T . In case of multiple objects, we index all these variables. Also, we will always use m to denote the number of trajectories and n for the complexity of one trajectory (i.e., the number of its time-space-positions).

For the definition of following, we fix three parameters τ_{\min} , τ_{\max} , and $\delta \in \mathbb{R}$ with $\tau_{\min} < \tau_{\max}$. The parameters τ_{\min}, τ_{\max} specify minimum and maximum offsets in time and δ specifies a maximum offset in space. In the definition of following behind, we will re-parameterize one trajectory by a varying time difference in $[\tau_{\min}, \tau_{\max}]$, such that the spatial positions of the two trajectories differ by at most δ . The parameters $\tau_{\min}, \tau_{\max}, \delta$ shall be chosen depending on the application and exactness of the input and will be discussed in more detail in Section 4.

Consider the two trajectories of entities a_1, a_2 shown in Figure 2, where Figure 2(a) shows the trajectories in space, while Figure 2(b) indicates time as the vertical axis. Assume that a_i is given by the parameterized trajectory f_i over the time interval $[s_i, t_i]$, for $i = 1, 2$, where $s_1 < s_2 < t_1 < t_2$. Or both trajectories are given for the same time interval $[s_1, t_2]$, as indicated by dashed lines in the figure. 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 $i = 1, 2$. 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 $d(q_1, q_2) \leq \delta$

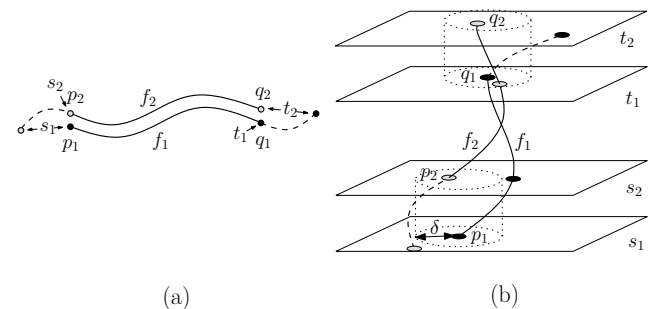


Figure 2: Trajectories of two entities moving in single file.

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 .

In this situation we have that at all times $s \in [s_2, t_1]$ entity a_2 is at a position which is δ -close to the position of a_1 at an earlier time $s' \in [s - \tau_{\max}, s - \tau_{\min}]$. This will be our definition of following behind, which can be formalized as follows.

DEFINITION 1. Let a_1 be an entity with parameterized trajectory f_1 over the time interval $[s_1, t_1]$ and let a_2 be an entity with parameterized 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 $\sigma: [s_1, t_1] \rightarrow [s_2, t_2]$ such that $\sigma(s_1) = s_2$ and

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

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

Our definition of following behind is closely related to a well-known distance measure for curves: the *Fréchet distance*. The Fréchet distance between curves can be illustrated by a man and a dog walking on the curves: assume the man walks on one curve and the dog on the other and the man holds the dog on a leash. Both may choose their speed and may stop but not walk backwards. Then the Fréchet distance is the shortest leash length that allows them to walk on the two curves from beginning to end.

More explicitly, our definition can be reformulated using the *free space diagram*, a geometric data structure for computing the Fréchet distance of polygonal curves, which was introduced by Alt and Godau [1]. Given two parameterized curves f_1, f_2 and a value $\delta > 0$, their free space diagram is defined as

$$F_\delta(f_1, f_2) = \{(s, t) \mid d(f_1(s), f_2(t)) \leq \delta\}.$$

Thus, in the man and dog metaphor, the free space diagram corresponds to the pairs of positions of man and dog at which a leash of length δ suffices. See Figure 3 for an example of a free space diagram.

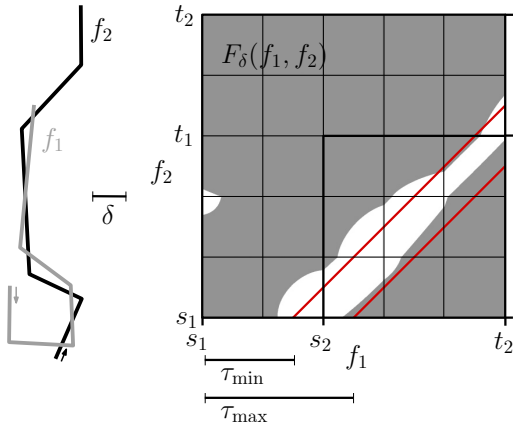


Figure 3: Two parameterized trajectories f_1, f_2 over the time interval $[s_1, t_2]$, a value $\delta > 0$, and the free space diagram $F_\delta(f_1, f_2)$ with $[\tau_{\min}, \tau_{\max}]$ -strip.

For polygonal curves the free space diagram consists of $n_1 \cdot n_2$ cells where n_1, n_2 are the complexities of f_1, f_2 , respectively. A cell is the free space diagram of two line segments and the free space inside a cell is convex. It therefore usually suffices to compute the cell boundaries of the free space diagram which can be computed in constant time each. We will consider only a part of the free space diagram, which we call the $[\tau_{\min}, \tau_{\max}]$ -strip defined as

$$[\tau_{\min}, \tau_{\max}]\text{-strip} := \{(s, t) \mid t - s \in [\tau_{\min}, \tau_{\max}]\}.$$

See again Figure 3 for an illustration. Note that in the free space diagram for the larger time interval $[s_1, t_2]$ the $[\tau_{\min}, \tau_{\max}]$ -strip lies below the diagonal, whereas it lies around the diagonal in the free space diagram for the time intervals $[s_1, t_1]$ and $[s_2, t_2]$. Since we only need to consider the $[\tau_{\min}, \tau_{\max}]$ -strip of the free space diagram our algorithms will have a linear run-time, in contrast to (known) algorithms for computing the Fréchet distance, which have a quadratic run-time.

A re-parameterization σ of one of the curves corresponds to a monotone path in the free space diagram, i.e., a path that only increases in both the horizontal and vertical direction. More explicitly, the monotone path is the graph of σ , i.e., it consists of all points $(t, \sigma(t))$. In the man and dog metaphor the monotonicity is guaranteed by the constrained that man and dog may not walk backwards. By the correspondence between re-parameterizations and monotone paths the following observation holds.

OBSERVATION 1. Entity a_2 is following behind a_1 in the time interval $[s_2, t_1]$ if the $[\tau_{\min}, \tau_{\max}]$ -strip of the free space $F_\delta(f_1, f_2)$ contains a monotone path from (s_1, s_2) to (t_1, t_2) .

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., τ_{\min} is too small or δ 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 in Figure 4. Figure 4(a) clearly shows one entity following behind another and Figure 4(b) shows two entities moving beside each other. However, where does this change, i.e., what about Figure 4(c)?

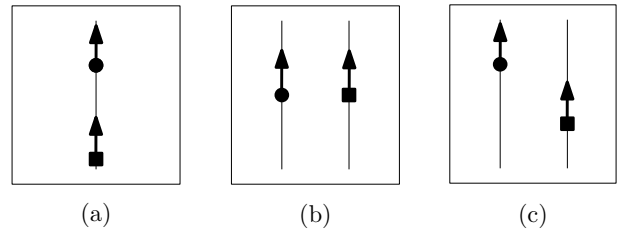


Figure 4: Different moving patterns: (a) following behind, (b) beside each other, and (c) “following beside”.

We therefore consider the following strict order of following.

DEFINITION 2. *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 .*

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. To see that this is a reasonable definition, we give the 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 δ in the lemma can be set to zero.

DEFINITION 3. *An entity a with parameterized trajectory f fulfills the travel condition in the time interval T if*

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

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

For the travel condition we defined the neighborhood of a region to be all points which have a small distance to the region based on the distance measure $d(\cdot, \cdot)$. See Figure 5 for an example. Intuitively, the travel condition says that an entity travels a distance of at least 2δ 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 has only technical reasons.

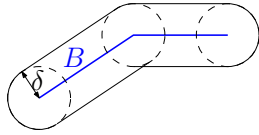


Figure 5: δ -Neighborhood of a polygonal line B .

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

PROOF. Assume they can although a_1 fulfills the travel condition. That is, there exists $t \in T$ such that a_1, a_2 both follow each other for the time interval $[t, t + \tau_{\max}]$. Then

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

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 . \square

In practice, we may detect that both 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 τ_{\min} or a smaller value for δ . Or we conclude that based on the input data we cannot decide if one is following the other.

3. ALGORITHMS

In this section we give algorithms for detecting single file behavior based on our definition of following behind. The algorithms we give detect the (non-strict) following behavior. If the traveling 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. We give algorithms first for two and then for several trajectories.

3.1 Two trajectories

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

An algorithm for fixed time.

By 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 $[\tau_{\min}, \tau_{\max}]$ -strip of the free space diagram of the trajectories. 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 1. We compute the *reachable free space* in the $[\tau_{\min}, \tau_{\max}]$ -strip, i.e., all free space that is reachable from the point (s_2, s_1) . We do this by the same technique as in [1], which we briefly review here.

The $[\tau_{\min}, \tau_{\max}]$ -strip can be computed column by column and each column can be computed cell by cell from bottom to top. First, we set all left cell boundaries of the first column and the bottom cell boundary of the first cell to empty except for the point (s_2, s_1) . Then we compute for each 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. We will 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. 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 6).

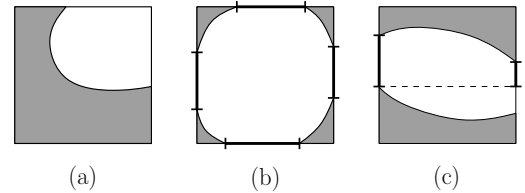


Figure 6: Computing the outgoing cell boundaries based on ingoing cell boundaries. The outgoing cell boundaries are empty in (a), equal to the ingoing cell boundaries in (b), and modified from the free space boundaries in (c).

If both ingoing boundaries are empty, then so are the outgoing boundaries (Figure 6(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 6(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 6 (b)). For 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. 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 free space cells intersected by the $[\tau_{\min}, \tau_{\max}]$ -strip. In the following we will always use k_{avg} and k_{max} to denote the average and maximum number of cells intersected by the $[\tau_{\min}, \tau_{\max}]$ -strip per row or column of the free space diagram. In the case of uniform time sampling k_{avg} and k_{max} equal $(\lceil \tau_{\max} - \tau_{\min} \rceil + i)$ with $i \in \{1, 2\}$.

THEOREM 1. *For two trajectories of complexity n each, we can determine using $O(nk_{\text{avg}})$ time and $O(n+k_{\text{max}})$ space whether one trajectory is following behind the other in a fixed time interval.*

PROOF. The correctness of the algorithm follows directly from Observation 1. The run-time is linear in the size of the $[\tau_{\min}, \tau_{\max}]$ -strip of the free space diagram that we need to compute. \square

An algorithm for variable time.

For a variable time interval, we again search for monotone paths in the $[\tau_{\min}, \tau_{\max}]$ -strip of the free space diagram. Now, however, we are searching for paths which do not necessarily span the whole strip. An example is shown in Figure 7. For this, we sweep the $[\tau_{\min}, \tau_{\max}]$ -strip with a similar approach as in [4]. As 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 labeled cell boundaries and then how to check which maximal paths end in a column.

In each cell, we need to compute the labeled upper and right cell boundary based on the previously computed labeled lower and left cell boundary. For the first column, the bottom-most and all left cell boundaries are empty except

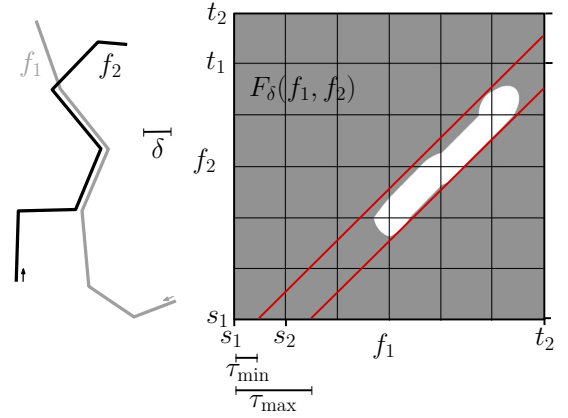


Figure 7: Two parameterized trajectories f_1, f_2 over the time interval $[s_1, t_2]$, a value $\delta > 0$, and the free space diagram $F_\delta(f_1, f_2)$ with $[\tau_{\min}, \tau_{\max}]$ -strip. Here, the two entities are following each other only for a part of $[\tau_{\min}, \tau_{\max}]$ -strip.

for the point (s_2, s_1) which is labeled s_1 . In the case where only the start point s_1 of f_1 is given, all non-empty left cell boundaries are labeled 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 labeled by their respective starting points.

Similar to the previous algorithm, we will call the labeled bottom and left cell boundaries of a cell its *ingoing* boundaries and the labeled upper and right cell boundary its *outgoing* boundaries. In each cell free space may *start*, *continue*, or *end*. Typical situations for these three cases are shown in Figure 8 (a),(b),(c). If a cell has empty ingoing and non-empty outgoing cell boundaries, then free space starts in this cell (Figure 8 (a)). Analogously, if the ingoing boundaries are non-empty and the outgoing boundaries are empty, then free space ends in this cell (Figure 8 (c)). If both the in- and outgoing boundaries are non-empty, free space continues (Figure 8 (b)). Also, free space may start or end as well as continue in this case (Figure 8 (d)).

The outgoing boundaries are the free space boundaries with labels based on the ingoing boundaries. The labels are illustrated in Figure 8. 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 8 (a) and x_3 in Figure 8 (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 8 (c). In a cell where free space starts or continues, we may have to split an interval once, as in Figure 8 (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 $2k$ 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 continu-

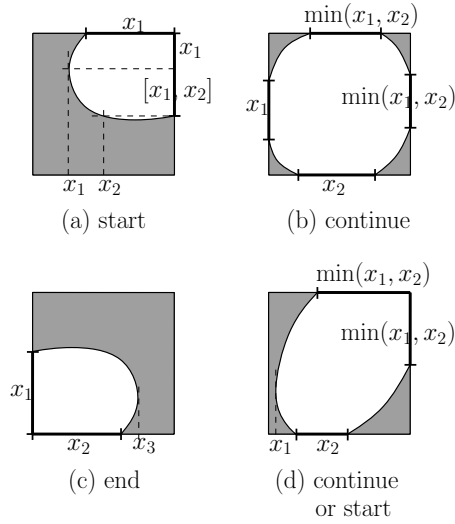


Figure 8: Examples for free space (a) starting, (b) continuing, or (c) ending in a cell. This may also occur mixed as in (d).

ing 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 and $O(n + k_{max}^2)$ space during which time intervals one trajectory is following behind the other.

PROOF. The correctness follows again from Observation 1. The run-time is linear in the number of event points of the sweep, since the cost at each event point is constant. \square

3.2 Several trajectories

We now consider single file behavior 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 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 and $O(nm + k_{max})$ space whether they are walking 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 walking in single file using the algorithm for two trajectories over a fixed time interval. We represent this information as a directed graph, as shown in Figure 9(a). That is, we build the graph $G = (V, E)$ of vertices $V = \{a_1, \dots, a_m\}$ and directed edges $E = \{(a_i, a_j) \mid a_i \text{ is following } a_j, i, j \in$

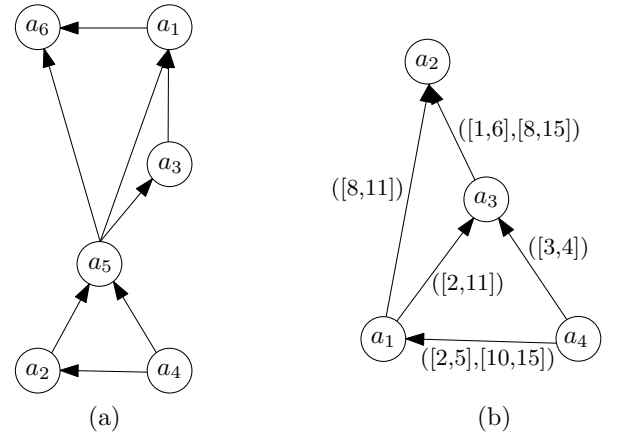


Figure 9: (a) The path $a_4a_2a_5a_3a_1a_6$ in G corresponds to a single file with the same entities. (b) In the variable time case the graph G represents all single file patterns, e.g., entities $a_4a_1a_3a_2$ form a single in the time interval $[2, 5]$ and entities $a_4a_1a_2$ form a single file in the time interval $[8, 11]$.

$\{1, \dots, m\}$). 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 and $O(nm + m^2 + k_{max})$ space all single file behaviors 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. This is $O(m^2nk_{avg})$. Finding all maximal, directed paths in the graph can be done $O(m^2)$ time. \square

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, labeled graph, where each edge is labeled by the corresponding time intervals, see Figure 9(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 labeled 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 labeled 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 labeled 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 behavior or assuming that all entities fulfill 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 behaviors and we can use it as a data structure for answering single file queries in time linear in the size of G , which is $O(m^2)$.

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

PROOF. As in the previous proof, the correctness and run-time follow from the correctness and run-time of the algorithm for two trajectories and the graph searching. \square

If we want to search for all single file behaviors 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 $O(n)$ starting times for the following behind behavior. In practice, 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.

4. EXPERIMENTS

We tested the effectiveness of our model and the efficiency of our algorithms on a set of real and generated data. The run-times and effectiveness of all our algorithms are directly based on the run-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 free space as described in Section 3.1 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 behavior 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. Else we use the next time stamp after the current start point as new start point. This simplified algorithm will not find all maximal monotone paths as the algorithm in Section 3.1 does, i.e., the simplified algorithm will not find overlapping monotone paths. 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 will 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.

4.1 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

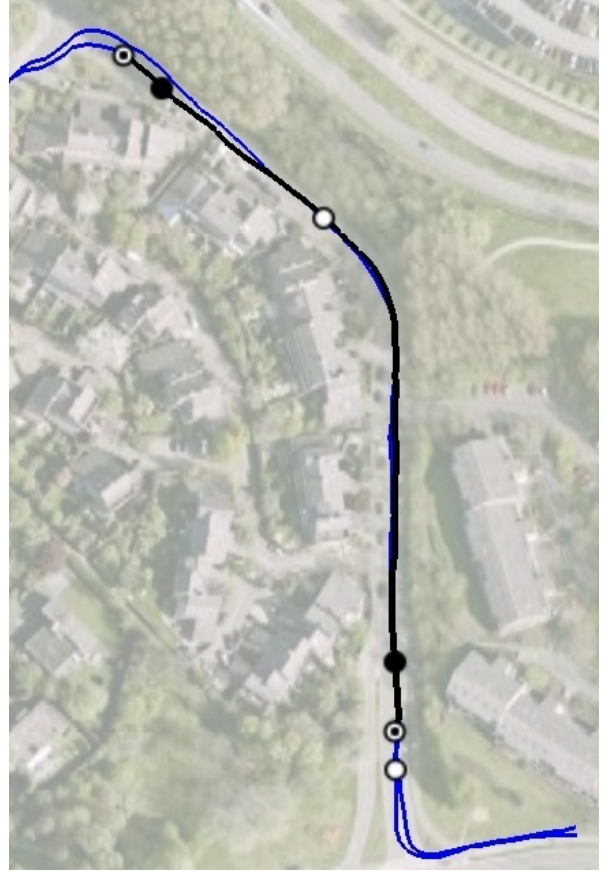


Figure 10: A following behavior in the test on real data for $\tau_{min} = 4$ and $\delta = 12$.

between the cyclists was about 2 to 4 seconds. This relatively large distance was chosen because of the considerably 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 (predictably) 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 one meter is used. This enables better results in detecting single file behavior. 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 behavior.

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 behaviors. Figure 10 shows an example of a following behavior 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 behavior. The next black dot marks the signal of the follower that the following behavior has been established. The circled dot in between marks the start of the detected following behavior. At the end of the follower behavior, again first the leader signaled the ending, which was confirmed by the follower. The circled dot marks the end of the detected

τ_{\min}	δ						
	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 behavior in dependence on δ and τ_{\min} is shown.

following behavior, which in this case is slightly too late.

The results for real data are shown in Table 1. For different values of δ and τ_{\min} , the percentage of correctly detected following and non-following behavior is shown. For each (δ, τ_{\min}) pair the top value gives the percentage of time intervals in which a following behavior was correctly detected. The bottom value gives the percentage of time intervals in which a non-following behavior 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 behavior were not classified and did not contribute to the percentages of correctly detected intervals. For instance in Figure 10 all time intervals of following were correctly identified, and a few intervals of non-following were incorrectly identified as following. In Table 1 the bold values indicate for each δ value the smallest value of τ_{\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 δ -neighborhood of a point after approximately 4 seconds. Note that for values of δ and τ_{\min} where the travel condition is likely to be fulfilled, the percentages of correctly identified non-following behaviors is large.

4.2 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 by moving 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 τ . 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 11.

For the time measurements we generated follower-leader tracks of up to 10 million data points. Figure 12 shows the time measurements in dependency on the number of data points for different values of k_{avg} . Since the trajectories were uniformly sampled we have $k_{\text{avg}} = \tau_{\max} - \tau_{\min}$. As expected,

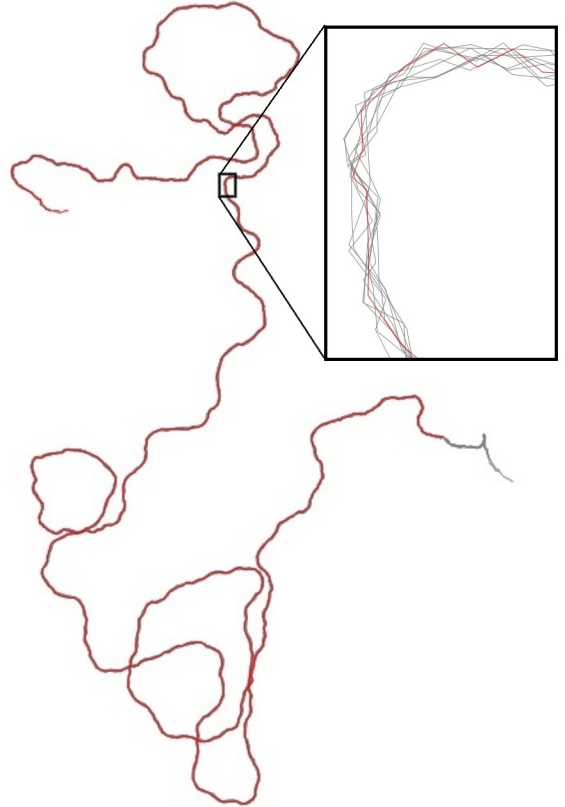


Figure 11: Example of a generated leader and several follower tracks.

the run-times are linear in the number of data points. The dependency of the run-time on k_{avg} is better than the worst case linear bound, since for large k_{avg} only a small part of the $[\tau_{\min}, \tau_{\max}]$ -strip is non-empty.

4.3 Choosing the Parameters

As our experiments confirm, choosing the values of δ , τ_{\min} , and τ_{\max} is crucial. The value of δ should be chosen such that in a following behavior the paths of the entities deviate by less than δ except for outliers. The deviation can come from the measurement or because the entities simply are not following exactly the same path. Next, τ_{\min} should be chosen as the smallest time difference between two entities in a single-file. It should if possible be at least the time needed to travel a distance of δ , otherwise even two identical trajectories would be classified as following. If a large δ is chosen, τ_{\min} can also be chosen larger. Finally, τ_{\max} is chosen as the largest time difference between two entities in a single-file. The choice of τ_{\max} is often not as important as the choice of the other two parameters.

Choosing δ or τ_{\max} too small or τ_{\min} too large increases the number of unrecognized following behaviors. The contrary, i.e., choosing δ or τ_{\max} too large or τ_{\min} too small increases the number of incorrectly identified following behaviors. Choosing a large τ_{\max} in a single file of $k > 2$ entities leads to the situation illustrated in Figure 13. Not only the correct following behind behaviors $a_{i+1} \rightarrow a_i$ are

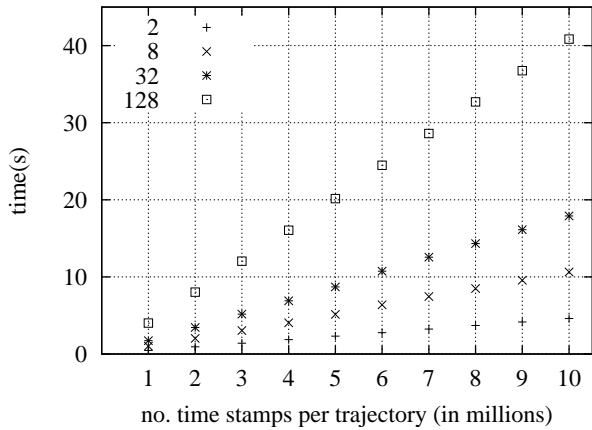


Figure 12: Run-time of the algorithm on generated data for $\tau_{\max} - \tau_{\min} = 2, 8, 32, 128$.

detected ($i = 1, \dots, m$) but also several following behind patterns $a_i \rightarrow a_j$ for $1 \leq i < j \leq m$. Although these are correct following behind behaviors, these are typically not of interest, in particular as they are implied by the following behind behaviors $a_{i+1} \rightarrow a_i$ for $i = 1, \dots, m$.

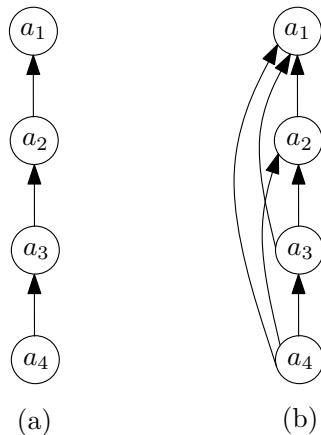


Figure 13: (a) Only the single file $a_1 a_2 a_3 a_4$ is detected. (b) Further following behaviors, e.g., $a_4 \rightarrow a_2$, are detected.

5. DISCUSSION

Our main contribution is defining a model for single file patterns by using the notion of following behind, which we show allows for efficient computation. Our initial experiments show that our model is effective.

Alternative definitions for following behind might use other definitions for the similarity of the re-parameterized trajectories in Definition 1. In particular, other distance measures may be used for $d()$, e.g., a different L_r -metric. A difficulty in modeling the notion of following is the potential overlap of the offsets in time and space. One might also try to incorporate directions into the model of following, e.g., “ a_2 is

always walking in the direction of the position of a_1 at a previous time”. However, this only works if a_2 is walking (more or less) straight. If a_2 moves in zig-zags (e.g., due to errors in gps), then this criterion is likely to fail.

Further information about the entities or the geographical space can be used to support the detection of single-file behaviors. For instance, one restriction for single file behaviors that occurs in applications is that the leader is known. This can be used to eliminate false positives and limit the number of single file behaviors. Also, if the entities are moving on a known transport network, a single-file behavior might simply mean that the entities follow the same route on the network with a suitable time difference. Instead of using δ -closeness one might require the entities to reach the same parts of the network within a certain time frame.

A further contribution of this paper is the use of a *constrained free space diagram*, i.e., the $[\tau_{\min}, \tau_{\max}]$ -strip. This idea is novel and interesting for two reasons: restricting the free space adds information to the monotone path found in the restriction (in our case a time difference between τ_{\min} and τ_{\max}) and the run-time of the algorithm is reduced (in our case from quadratic to linear). This idea may have further applications, for instance in trajectory simplification.

6. REFERENCES

- [1] H. Alt and M. Godau. Computing the Fréchet distance between two polygonal curves. *International Journal on Computational Geometry and Applications*, 5:75–91, 1995.
- [2] 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.
- [3] M. Benkert, J. Gudmundsson, F. Hübner and T. Wolle. Reporting flock patterns. *Computational Geometry - Theory and Applications*, 41(3):11–125, 2008.
- [4] K. Buchin, M. Buchin, J. Gudmundsson, J. Luo and M. Löffler. Detecting commuting patterns by clustering subtrajectories. *Proceedings of the 19th International Symposium on Algorithms and Computation (ISAAC)*, 2008.
- [5] F. E. Fish. Energetics of swimming and flying in formation. *Comments on Theoretical Biology* 5: 283–304, 1999.
- [6] J. Gudmundsson, P. Laube and T. Wolle. Movement patterns in spatio-temporal data. In *Encyclopedia of GIS*, Eds. S. Shekhar and H. Xiong, Springer, 2008.
- [7] J. Gudmundsson, M. van Kreveld and B. Speckmann. Efficient detection of motion patterns in spatio-temporal data sets. *GeoInformatica*, 11(2):195–215, 2007.
- [8] J. Gudmundsson and M. van Kreveld. Computing longest duration flocks in spatio-temporal data. *Proceedings of the 16th International Conference on Advances in Geographic Information Systems (ACM GIS)*, 2006.
- [9] C. S. Jensen, D. Lin, and B. C. Ooi. Continuous clustering of moving objects. *IEEE Transactions on Data Engineering*, 19(9):1161–1174, 2007.

- [10] H. Jeung, M. L. Yiu, X. Zhou, C. S. Jensen and H. T. Shen. Discovery of convoys in trajectory databases. Proceedings of the 34th International Conference on Very Large Data Bases (VLDB), 2008.
- [11] P. Kalnis, N. Mamoulis and S. Bakiras. On discovering moving clusters in spatio-temporal data. Proceedings of the 9th International Symposium on Advances in Spatial and Temporal Databases (SSTD), pp. 364–381, 2005.
- [12] P. Laube, M. van Kreveld and S. Imfeld. Finding REMO – detecting relative motion patterns in geospatial lifelines. Proceedings of the 11th International Symposium on Spatial Data Handling, pp. 201–214, 2004.
- [13] Y. Li, J. Han and J. Yang. Clustering moving objects. Proceedings of the 10th ACM SIGKDD International Conference on Knowledge discovery and data mining, pp. 617–622, 2004.
- [14] T. Shirabe. Correlation analysis of discrete motions. Proceedings of the 4th International Conference on Geographic Information Science (GIScience), pp. 370–382, 2006.
- [15] F. Verhein and S. Chawla. Mining spatio-temporal association rules, sources, sinks, stationary regions and thoroughfares in object mobility databases. Proceedings of the 11th International Conference on Database Systems for Advanced Applications (DASFAA), pp. 187–201, 2006.