Efficient Detection of Patterns in 2D Trajectories of Moving Points — Source link ⧉

Joachim Gudmundsson, Marc van Kreveld, Bettina Speckmann

Institutions: NICTA, Utrecht University, Eindhoven University of Technology

Related papers:

- Finding REMO — Detecting Relative Motion Patterns in Geospatial Lifelines

- Reporting flock patterns

- Discovering relative motion patterns in groups of moving point objects

- Computing longest duration flocks in trajectory data

- On discovering moving clusters in spatio-temporal data

Efficient Detection of Patterns in 2D Trajectories of Moving Points

**Joachim Gudmundsson · Marc van Kreveld ·
Bettina Speckmann**

Abstract Moving point object data can be analyzed through the discovery of patterns in trajectories. We consider the computational efficiency of detecting four such spatio-temporal patterns, namely flock, leadership, convergence, and encounter, as defined by Laube et al., *Finding REMO—detecting relative motion patterns in geospatial lifelines*, 201–214, (2004). These patterns are large enough subgroups of the moving point objects that exhibit similar movement in the sense of direction, heading for the same location, and/or proximity. By the use of techniques from computational geometry, including approximation algorithms, we improve the running time bounds of existing algorithms to detect these patterns.

J. Gudmundsson
NICTA, Sydney, Australia
e-mail: joachim.gudmundsson@nicta.com.au

M. van Kreveld (✉)
Department of Information and Computing Sciences, Utrecht University,
Utrecht, The Netherlands
e-mail: marc@cs.uu.nl

B. Speckmann
Department of Mathematics and Computer Science, TU Eindhoven,
Eindhoven, The Netherlands
e-mail: speckman@win.tue.nl

1 Introduction

Moving point object data is becoming increasingly more available since the development of GPS and radio transmitters. One of the objectives of spatio-temporal data mining [16], [23] is to analyze such data sets for interesting patterns. For example, a group of caribou with radio collars gives rise to the positions of each caribou at a sequence of time steps. Analyzing this data gives insight into entity behavior, in particular, migration patterns [22]. The analysis of moving objects also has applications in sports (e.g., soccer players [12]) and in socio-economic geography [8].

There is ample research on data mining of moving objects (e.g., [13], [25], [27], [28], [30]) in particular, on the discovery of similar trajectories or clusters. Trajectories for moving points are also referred to as (geo)spatial lifelines. In general the input is a set of n moving point objects whose locations are known at t consecutive time steps, that is, the path of each moving object is a polygonal line that can self-intersect (see Fig. 1). For brevity, we will call moving point objects *entities* from now on.

The REMO framework (RElative MOtion) was developed by Laube and Imfeld [14] to define similar behavior in groups of entities. To this end, they define a collection of spatio-temporal patterns based on similar direction of motion or change of direction. These patterns are meaningful, for example, with respect to data that represents the movement of a caribou herd or data that represents change of political opinions in a space where dimensions represent left–right, liberal–conservative, and ecological–technocratic. Laube et al. [15] extended the framework by not only including direction of motion, but also location itself. They defined several spatio-temporal patterns, including *flock*, *leadership*, *convergence*, and *encounter*, which can occur for a subset of the entities at a given time step or time interval. They also give algorithms to compute these patterns efficiently. We formalize the patterns below.

We assume that the data to be analyzed consists of n entities, each with t locations at consecutive time steps. We also assume that the locations of the entities are known at the same time steps (concurrent observation), but we do not make any assumptions on the distance traveled in any time step for any entity. We will treat each time step separately. Hence, at each time step, we have to analyze a set of n points with a given motion direction and speed. The flock pattern describes entities moving in the same direction while being close to each other (see Fig. 1). We formalize "being
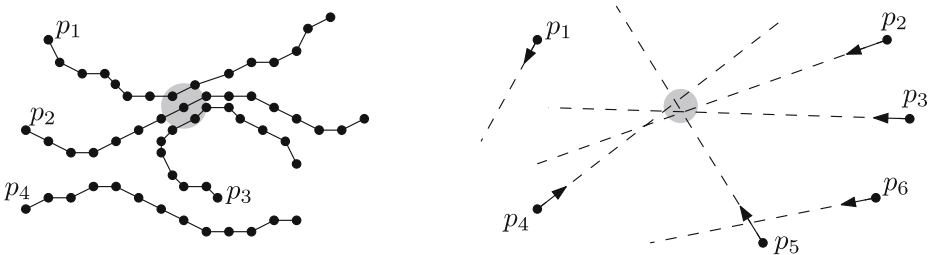
Fig. 1 *Left*, a flock pattern for p_1, p_2, p_3 at the eighth time step. It is also a leadership pattern with p_2 as the leader. *Right*, a convergence pattern if $m = 4$ for p_2, p_3, p_4, p_5

close" as being inside a circle of some specified radius r, whose position is initially not known. A set of entities can have many flock patterns and even one single entity can be involved in several flock patterns. The leadership pattern is similar to the flock pattern, except that one of the entities was already heading in the specified direction for some time before the flock pattern occurs. Convergence refers to moving to the same location, given that the direction of motion does not change. The entities need not arrive at the same time. Again, "same location" is formalized as a circle whose radius can be specified and whose position is unknown. Finally, encounter refers to moving to and meeting at the same location, so it is a convergence pattern where the entities arrive at the same time. In all cases we are looking for "interesting" patterns, which means that a large enough subgroup of all entities forms the pattern in a small enough region.

Formally, flock, leadership, convergence, and encounter patterns are defined for some given set of entities at a specified time step i. The time step i allows us to obtain the locations of the n entities from the input, and the directions and speeds of all entities can be determined from their locations at time steps i and $i + 1$. Alternatively, we can choose a longer time interval than one time step, which again allows us to determine a general direction of motion and an average speed. Hence, we can assume that location, direction and speed are known. To form a pattern, we require that a sufficient number of entities are involved, and denote the minimum number by m. The four patterns are defined as follows:

Flock	Parameters: $m > 1$ and $r > 0$. At least m entities are within a circular region of radius r and they move in the same direction.
Leadership	Parameters: $m > 1$, $r > 0$, and $s > 0$. At least m entities are within a circular region of radius r, they move in the same direction, and at least one of the entities was already heading in this direction for at least s time steps.
Convergence	Parameters: $m > 1$ and $r > 0$. At least m entities will pass through the same circular region of radius r (assuming they keep their direction).
Encounter	Parameters: $m > 1$ and $r > 0$. At least m entities will be simultaneously inside the same circular region of radius r (assuming they keep their speed and direction).

All four patterns are related to clustering, especially the flock pattern. In clustering it is common to allow clusters of arbitrary shape, but another choice is identifying groups of entities inside a fixed region type. For example, the Geographical Analysis Machine [18] considers circular regions that contain many points. Density measures for point sets also assume fixed regions [19]. Especially for convergence and encounter, fixed region types are natural, because these patterns are not realized yet. They are only extrapolated to occur later.

For each of the four patterns, we must specify what we want to find and report in a given data set. One possibility is simply to detect whether a pattern occurs. If so, we may want to report one example of such a pattern. Secondly, we may want to find all patterns that occur. Thirdly, we may want to report the largest size subset of entities that form a pattern. We refer to these pattern problems as *detect*, *find all*, and *find largest*.

In the following sections we address the algorithmic problems of computing flock, leadership, convergence, and encounter patterns. Exact algorithms solving these

Table 1 Running time bounds for finding patterns; $\delta > 0$ is an arbitrarily small positive constant, ε is the relative approximation error, and M is the size of the largest pattern

Pattern	Exact (from [15])	Exact (new)	Approximate
Flock	$O(nm^2 + n \log n)$	–	$O(\frac{n}{\varepsilon^2} \log \frac{1}{\varepsilon} + n \log n)$ (radius)
Leadership	$O(ns + nm^2 + n \log n)$	–	$O(ns + \frac{1}{\varepsilon^2} n \log \frac{1}{\varepsilon} + n \log n)$ (radius)
Convergence	$O(n^2)$	– $O(n^3)$ (all)	$O(n^{2+\delta}/(\varepsilon m))$ (subset) $O(\frac{1}{\varepsilon} n^2 \log n)$ (radius)
Encounter	$O(n^4)$	$O((m + \log n)n^2)$ (detect) $O((M + \log n)n^2 \log M)$ (largest)	

In the "find all" problems, the time needed to report the output must be added.

problems were already given in [15] and here we improve the exact results only for the encounter pattern—albeit in three different ways (see Table 1). However, recall that our patterns always involve a "sufficiently large" group of entities being in or passing through a "sufficiently small" area which we formalize by using a threshold m for the number of entities and a radius r defining the circle that represents the area. Any exact values of m and r hardly have a special significance—20 caribou meeting in a circle with radius 50 m form as interesting a pattern as 19 caribou meeting in a circle with radius 51 m. Therefore the problem of computing these patterns is ideally suited for approximation algorithms.

Our results for one time step are listed in Table 1. When comparing the running times of the exact and the approximate algorithms, we observe that probably an order of magnitude can be saved due to approximation. The improvements for flock, leadership, and encounter are more significant than for convergence. The precise efficiency improvements cannot be given, because they also depend on the size of the input, the size of the patterns to be found, the quality of the approximation, and the implementation. The asymptotic notation of running times in Table 1 shows that for large data sets, the improvements are more significant than for smaller data sets.

The algorithms consider the time steps from τ_1 up to τ_t consecutively, and hence, t is a multiplicative factor in all time bounds in the table when we consider all starting times. Since the time steps can be considered in sequence, not all input data is needed in main memory at the same time. Therefore, we do not need to deal with secondary memory issues. We note that we look only for patterns defined by the given input locations and time steps, not for patterns defined by locations in between. This is referred to as the snapshot view of time [20]. If time is sampled sufficiently densely, this is no severe limitation. Furthermore, our algorithms do not assume that the extrapolated time instances for convergence and encounter are sampled, and we find these for any time instance.

In Section 2 we examine what it means for two patterns to be different and discuss a method to determine a set of sufficiently different patterns as a postprocessing step. In Sections 3, 4, and 5 we describe efficient approximation algorithms for all four patterns, where we let either the size of the region or the specified subset size deviate slightly from what is specified (see Table 1). In particular, approximating

the size of the region means that a region with a radius between r and $(1 + \varepsilon)r$ that contains at least m moving entities may or may not be reported as a pattern while a region with a radius of at most r that contains at least m entities will always be reported. Approximating the size of the subset, m, implies that we will find all patterns that involve at least $(1 + \varepsilon)m$ entities, we may or may not find patterns that involve between m and $(1 + \varepsilon)m$ entities, and we will not find patterns with less than m entities.

2 Different patterns and postprocessing

When reporting a pattern that was detected, the most complete information that can be supplied is giving the position of the pattern, in case of encounter also the time, and furthermore the complete subset of entities that form the pattern. For our patterns, there may be several circles with radius r that yield a pattern with the same subset of entities, and therefore it seems natural to define that two patterns are different if they involve a different subset. It is well-known that there can be $\Theta(n^2)$ combinatorially distinct ways to place a circle of radius r amidst n points in the plane, and therefore there can be $\Theta(n^2)$ flock patterns and leadership patterns involving different subsets of entities. Similarly, there can be $\Theta(n^2)$ convergence patterns and $\Theta(n^3)$ encounter patterns. Even if we require that the entities of one pattern may not be a subset of the entities of another pattern (that is, only *maximal* patterns should be reported), there may still be as many patterns, asymptotically. If an algorithm would output all patterns by also reporting all entities involved, the output size and therefore the time required can be $\Theta(n^3)$ for the flock, leadership, and convergence pattern, and $\Theta(n^4)$ for the encounter pattern.

Many of the patterns we find may be similar, for example, two patterns may share $m - 1$ entities and differ in only one. The question arises when we should consider patterns to be different. The issue of finding sufficiently different patterns is related to the concept of *interestingness* of a pattern [11], [26]. In our case, we wish to compute an *interesting set* of patterns, with the idea that in each such set, only one of two very similar patterns should appear. We treat the issue of different patterns objectively: only the data itself determines when a set of patterns is interesting; it is not user-dependent [26].

One could take the simple view that two patterns are different if they involve different subsets of entities. After finding all patterns, we can then identify patterns that are the same and report only one of each. This postprocessing step can be done using sorting. If we use a different definition of when two patterns are different, then a different postprocessing step should be used. Most of our algorithms are approximation algorithms on either the radius or the subset size. In these cases, the definition of different patterns is more complex. We cannot require reporting all different patterns, and even if we could, it would undo the efficiency savings that we get from computing approximate patterns. Intuitively, due to the approximation algorithm ideas, we often find one representative of a group of patterns that are similar.

We present a simple definition and a general algorithm for dealing with similar patterns. We define that two patterns are different if their circle centers are at distance at least ρ. If we choose $\rho < 2r$ then the circles of different patterns can

overlap, and two circles can contain exactly the same entities. If we choose $\rho > 2r$ then the circles are disjoint, and patterns cannot contain the same entities.

Assume that for some pattern, the *find all* algorithm returns a set of circles that contain a sufficient number of entities to form a pattern (or an approximate pattern). We construct a geometric graph $\mathcal{G}$ where the nodes represent the patterns that we found and where two nodes are connected by an arc if the centers of the circles are at distance at most ρ. An independent set $\mathcal{I}$ of nodes in $\mathcal{G}$ represents a set of different patterns by definition. A maximal independent set $\mathcal{I}$ has the property that for every node not in $\mathcal{I}$, there is a neighboring node in $\mathcal{I}$. When reporting a maximal independent set of nodes and their patterns, this implies that for any pattern not reported there is another pattern with its center within distance ρ that is reported. We remark that computing a *maximum* independent set in $\mathcal{G}$ is NP-hard, but we are only interested in a *maximal* independent set.

We will not construct $\mathcal{G}$ explicitly. If we find N patterns, $\mathcal{G}$ may have $\Theta(N^2)$ edges, and hence constructing $\mathcal{G}$ already takes more time than we want to spend. Instead, we can compute a maximal independent set incrementally. Assume that the patterns that are found have circle centers $c_1, \ldots, c_N$. Assume we have treated $c_1, \ldots, c_{i-1}$, and we have reported some maximal independent set $\mathcal{I}$ of them. We use a data structure $\mathcal{D}$ for the points in $\mathcal{I}$ that can answer closest point queries. To decide if we should report the pattern for circle center c_i we search in $\mathcal{D}$ to find the closest center from $\mathcal{I}$. If it is at distance at most ρ, then we discard c_i and continue with c_{i+1}. Otherwise, we report the pattern corresponding to circle center c_i and store c_i in $\mathcal{D}$. With the *logarithmic method* [3] applied to point location in the Voronoi diagram (Theorem 7.3.4.1 in [21]), we can process a sequence of at most N queries and N insertions in $O(N \log^2 N)$ time.

3 Flock and leadership

This section discusses the detection of the flock pattern and its extension, the leadership pattern. The leadership pattern is discussed only briefly, since its detection is a fairly straightforward extension of the flock algorithm.

For flock detection, we are given a set of n moving entities as well as a radius r and the minimum size $m \leq n$ for a subset to form a pattern. As in [15] we first separate the input data into eight subsets according to their motion direction and then treat each subset separately. (Ideally, we should repeat the process with the eight subsets which we obtain after splitting the input according to motion directions that are rotated by $\pi/8$ degrees.) Laube et al. [15] propose an algorithm that is based on higher-order Voronoi diagrams with a running time of $O(nm^2 + n \log n)$. Finding all flock patterns, or the largest flock pattern, takes $O(n^2)$ time by computing the depth in an arrangement of circles. Detecting one flock pattern takes $O(nm + n \log n)$ time by incrementally constructing an arrangement of circles (de Berg, personal communication, 2005). An approximately largest flock pattern can be found in $O(\frac{n}{\varepsilon^2} \log^2 n)$ time for any constant $\varepsilon > 0$; this pattern is guaranteed to have at least $(1 - \varepsilon)M$ entities, where M is the size of the largest flock pattern [1].

We are presenting an approximation algorithm that approximates the size of the significant region and requires $O(\frac{n}{\varepsilon^2} \log \frac{1}{\varepsilon} + n \log n)$ time.

3.1 Approximating the radius

We will use a *quadtree* [24] as a building block for our algorithm. Let $S = \{p_1, \ldots, p_n\}$ be a set of n points in the plane contained in a square C of length ℓ. A quadtree $\mathcal{T}$ for S is recursively constructed as follows. The root of $\mathcal{T}$ corresponds to the square C. The root has four children corresponding to the four subsquares of C of side length $\ell/2$. The leaves of $\mathcal{T}$ are the nodes whose corresponding square contains exactly one point. Using a *compressed quadtree* [2] for $\mathcal{T}$ reduces its size to $O(n)$ by removing nodes not containing any points of S and eliminating nodes having only one child. A compressed quadtree for a set of n points in the plane can be constructed in $O(n \log n)$ time.

Now consider a subset S of the input as described earlier. We already know that all entities in S move in roughly the same direction so it remains to report all circles of radius r that contain the positions of at least m entities. That implies that we can treat S simply as a set of points in the plane for the remainder of this section.

We first construct a compressed quadtree $\mathcal{T}$ for S with the additional property that every non-empty square C corresponding to a node v has side length between $\frac{\varepsilon}{4}r$ and $\frac{\varepsilon}{8}r$. That is, we stop the recursion as soon as we reach a small enough side length; several points can be in any such square. We then build the arrangement $\mathcal{A}$ of all squares in $\mathcal{T}$ (see Fig. 2). $\mathcal{A}$ can be built from $\mathcal{T}$ in $O(n \log n)$ time. Each non-empty face/cell C of $\mathcal{A}$ stores information about the number of points of S within the cell, denoted by S_C.

A simple packing argument yields the following observation:

Observation 3.1 A disc D of radius $O(r)$ intersects $O(1/\varepsilon^2)$ cells of $\mathcal{A}$.

We now process the $O(n)$ non-empty cells in $\mathcal{A}$ one-by-one. Consider a non-empty cell C of $\mathcal{A}$, and denote the center of C by c. We traverse $\mathcal{A}$, starting at C, and find all cells of $\mathcal{A}$ within distance $(2 + \frac{\varepsilon}{4})r$ of c. By using a standard breadth-first search in the arrangement this can be done in time proportional to the number of cells reported, thus in $O(1/\varepsilon^2)$ time according to Observation 3.1. We then sort the reported cells
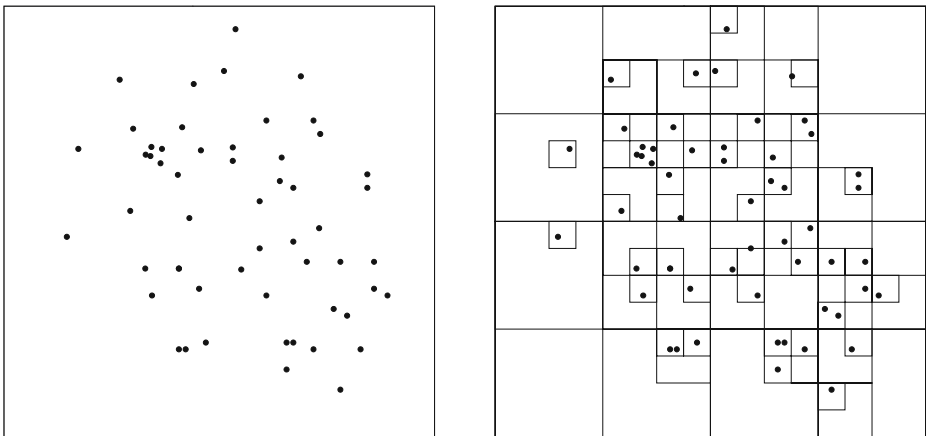
Fig. 2 The input set S contained in a *square* (*left*). The arrangement $\mathcal{A}$ of the *squares* obtained from the quadtree for S (*right*)
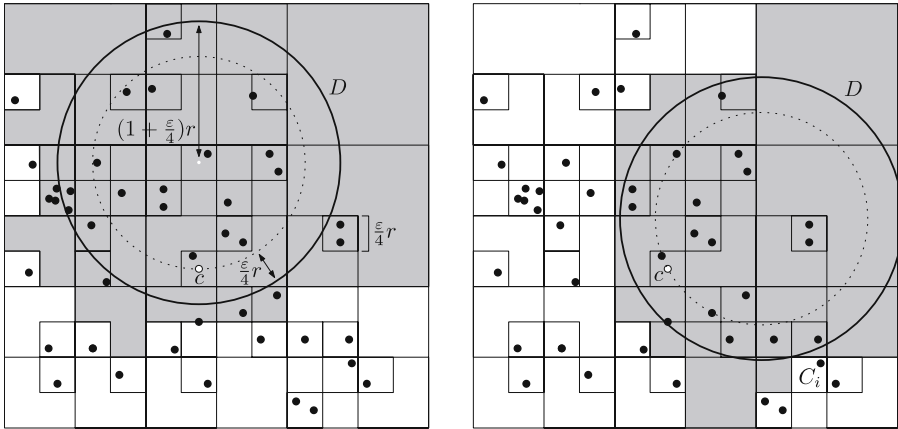
Fig. 3 Sweeping $\mathcal{A}$ with a *circle* D of radius $(1 + \frac{\varepsilon}{4})r$. Starting position of D (*left*), the non-empty cell C_i enters D (*right*)

into an event queue for a rotational plane sweep around c with a disc D of radius $(1 + \frac{\varepsilon}{4})r$ and with c fixed at distance $\frac{\varepsilon}{4}r$ from the boundary of D, as illustrated in Fig. 3. (Note that for the sake of illustration ε is chosen very large with respect to r in the figures.) Each non-empty cell C_i can cause at most two events since it can enter or leave D at most once. The event queue can be built in time $O(1/\varepsilon^2 \log 1/\varepsilon)$ by using a standard sorting algorithm.

Initially D is placed such that its bottom point is at distance $\frac{\varepsilon}{4}r$ from c (see Fig. 3 (left)). We compute the number of points of S which are contained in the cells of $\mathcal{A}$ that have a non-empty intersection with D. This number is denoted by S_D and can be computed in $O(1/\varepsilon^2)$ time since each cell contains information about the number of points within it. Now we rotate D clockwise around c and process events as they occur. If a non-empty cell C_i enters D then we increment S_D by S_{C_i}, if C_i leaves D then we decrement S_D by S_{C_i}. If $S_D \geq m$ then we report the disc D' with radius $(1 + \varepsilon)r$ centered at the center c_D of D. (Note that every non-empty cell of $\mathcal{A}$ that intersects D necessarily lies entirely within D'.)

It remains to show that our circular sweeps do indeed find all patterns. Consider a set F of entities that form a flock pattern. There exists a disc D_F of radius r that contains F and whose boundary passes through a point $p \in S$, as shown in Fig. 4. Consider the cell C of $\mathcal{A}$ containing p and let c be its center. Since C is non-empty we will perform a circular sweep around c. At some point during this sweep the center c_D of D will necessarily lie on the line through c and the center of c_F of D_F. The triangle inequality then implies that c_D lies within a circle of radius $\frac{\varepsilon}{4}r$ around c_F and therefore D_F is completely contained in D. This means that F is contained in D and so $S_D \geq m$.

Theorem 1 *Given a set of n moving entities, a radius r, the minimum size $m \leq n$ for a subset to form a pattern, and a positive constant ε. Using a $(1 + \varepsilon)$-approximation with respect to the radius of the flocking pattern in 2D, one can compute:*

1. *The existence of flock patterns in $O(\frac{n}{\varepsilon^2} \log \frac{1}{\varepsilon} + n \log n)$ time.*

Fig. 4 The *center c_D* of *D* lies on the line through *c* and *c_F*
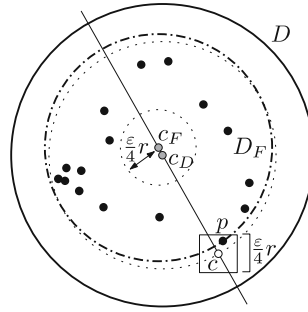
2. *All flock patterns in $O(\frac{n}{\varepsilon^2} \log \frac{1}{\varepsilon} + n \log n + N)$ time, where N is the size of the output.*

Proof The two claims follow from the fact that there are $O(n)$ non-empty cells, and the event queue for each cell can be built and processed in $O(\frac{1}{\varepsilon^2} \log \frac{1}{\varepsilon})$ time. Building the quadtree requires $O(n \log n)$ time, thus the theorem follows.   □

3.2 Different patterns and postprocessing

The flock patterns that we compute consist of a circle of radius $(1 + \frac{\varepsilon}{4}) \cdot r$ that contains a set of $O(\frac{1}{\varepsilon^2})$ grid squares that together contain at least *m* points. Since we rotate about centers of grid squares, we will find the same set of $O(\frac{1}{\varepsilon^2})$ grid squares during at most $O(\frac{1}{\varepsilon^3})$ sweeps. We can prove a better bound.

Lemma 1 *The algorithm computes at most $O(\frac{1}{\varepsilon})$ circles that contain the same set of non-empty grid squares.*

Proof Let *A* be any set of non-empty grid cells. During a sweep around the center *c* of a cell *C*, we find *A* at most once, namely at the event where it first appears. Cell *C* must also be non-empty, otherwise we would not have used *c* as a sweep center. A packing argument shows that there can be $O(\frac{1}{\varepsilon})$ non-empty cells whose center is used for a sweep and for which the circle contains all cells of *A* but not any other non-empty cell.   □

The $O(\frac{1}{\varepsilon})$ circles that contain the same non-empty cells as in the lemma are slightly different. Therefore, they may contain different entities if we take the ones in cells that intersect the circles too, as in Fig. 5. We may accept that the same pattern may be found $O(\frac{1}{\varepsilon})$ times. If not, we must detect patterns that potentially are the same, and we should do this without considering all points in all patterns for efficiency reasons. We will only report one circle from a group whose circles contain the same set of non-empty grid squares. This implies that for every pattern that we do not report, there is another pattern that shares at least *m* entities with it.

Since the circles of a group are not precisely the same, we cannot sort all circles lexicographically on (x, y, r), the coordinates of the center and the radius, to remove reoccurring patterns. Instead, for each circle that we find during the algorithm, we compute the smallest enclosing circle of the set *A* of non-empty grid squares (in fact,

Fig. 5 Two discs that contain the same *non-empty squares* (*dark grey*), but D_1 contains one fewer entity than D_2


D_1 D_2
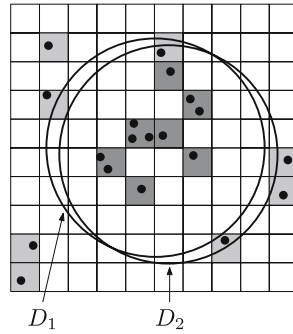
of their corner vertices). Although A can have size $O(\frac{1}{\varepsilon^2})$, we can easily adapt the detection algorithm so that it reports a circle together with the leftmost and rightmost non-empty grid cell of every row that lies inside the circle. We simply maintain it during the sweep. The smallest enclosing circle for these $O(\frac{1}{\varepsilon})$ grid cells is the same as for A. Smallest enclosing circle computation takes time linear in the number of points. Now all circles that contain the same non-empty grid cells have the same coordinates and we can sort lexicographically on (x, y, r) to remove duplicates. In total, the $O(n/\varepsilon^2)$ circles for flock patterns each require a smallest enclosing circle computation. The additional time needed is $O(\frac{n}{\varepsilon^3} + \frac{n}{\varepsilon^2} \log n)$.

3.3 Adaptations for the leadership pattern

To detect or find all leadership patterns we are given an additional parameter s that prescribes during how many time steps the leader was already moving in the specified direction. We modify the flock pattern algorithm to find leadership patterns as follows. Before starting flock detection, we decide for each entity whether it can be a leader, that is, whether that entity was already heading in the same direction during the previous $s - 1$ time steps. This takes $O(ns)$ time. For each grid cell obtained from our compressed quadtree, we also store whether it contains a leader. During the sweep we maintain whether the circle D contains some leader. When a flock pattern with a leader is discovered, it is a leadership pattern. We conclude that the time bounds in the theorem above also hold for the leadership pattern if we add an additional $O(ns)$ time term. Note that to find leadership patterns for all t time steps of the input data, only $O(nt)$ additional time is needed (and not $O(ns \cdot t)$ time).

4 Convergence

In this section we discuss the detection of the convergence pattern. Again, we are given a set of n moving entities as well as a time step τ_i, a radius r, and the minimum size $m \leq n$ for a subset to form a pattern. For each entity, we take its locations p_i and p_{i+1} at time steps τ_i and τ_{i+1}, respectively, to determine a half-line that starts at p_i and passes through p_{i+1} (see Fig. 6). Alternatively, we can choose to take a larger number of time steps to determine a half-line that represents the extrapolated future locations. This may be more reliable. In [15] Laube et al. show how to use this representation to detect convergence patterns in $O(n^2)$ time. They compute the
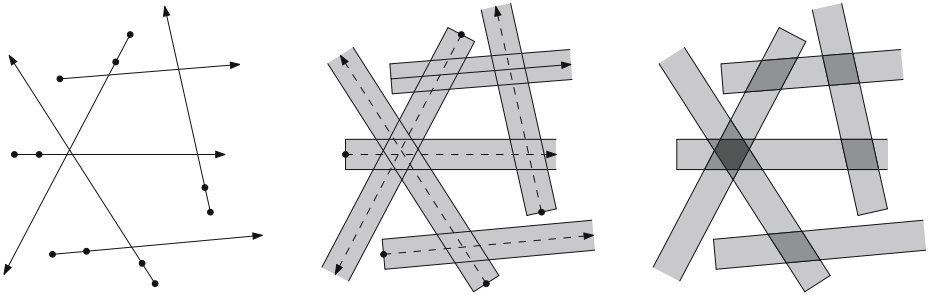
Fig. 6 *Half-lines* (*left*) converted into *half-strips* of a given width (*middle*), such that for each cell the number of *half-strips* covering it can be determined (*right*)

arrangement formed by the thickened half-lines which are turned into half-strips of width $2r$. For each of the $O(n^2)$ cells of the arrangement they compute the number of half-strips that cover it (see Fig. 6), and report each cell that is covered by at least m half-strips.

If $r = 0$, that is, the region of interest consist of only a single point, then the dual of the convergence problem (where lines are turned into points and vice versa) can be expressed as follows. Given a set of n points in the plane, test whether there is a line that passes through at least m points. For this special case Guibas et al. [9] show how to report all lines containing at least m points in time $O(\min\{\frac{n^2}{m} \log \frac{n}{m}, n^2\})$. Furthermore, Erickson [7] shows that the problem of deciding if any three lines have a common intersection point has a lower bound of $\Omega(n^2)$ time in a particular model of computation which in addition to standard operations also allows sidedness queries.

We are presenting an approximation algorithm that approximates the minimum size of a subset to form a pattern. Our algorithm reports a set of N circles of radius r that are each visited by at least $(1 + \varepsilon)m$ moving entities, and takes $O(n^{2+\delta}/(\varepsilon m) + N)$ time for any constant $\delta > 0$. A circle that is visited by less than m entities will not be reported, while a circle visited by at least m entities and less than $(1 + \varepsilon)m$ entities may, or may not, be reported. For any circle with at least $(1 + \varepsilon)m$ entities that is not reported, a similar pattern that differs by at most εm entities is reported.

4.1 Approximating the subset size

We are using the representation of the problem proposed in [15], that is, we study the arrangement of the thickened half-lines of width $2r$ described above. The approximation algorithm is a simple divide-and-conquer algorithm using the well-known *cutting lemma* which we state here for completeness.

For a given set of lines L and a parameter s, we seek a partition of the plane into a set of h (possibly unbounded) triangles $\Delta_1, \ldots, \Delta_h$ such that the interior of each triangle Δ_i is intersected by at most n/s lines of L. A partitioning of the plane with this property is called a $1/s$-cutting of the arrangement $\mathcal{A}(L)$. Chazelle proved the following lemma:

Lemma 2 *(Cutting lemma [5]) A $1/s$-cutting of $\mathcal{A}(L)$ that uses $\Theta(s^2)$ triangles can be computed in $O(ns)$ time.*

Now consider the set H of the n half-strips of width $2r$ in the plane, and the set L of $3n$ lines supporting the edges and half-lines that bound the half-strips. Initially we construct a triangle Δ that contains all intersections between the half-strips. The number of half-strips that completely cover Δ, denoted by $|\Delta|$, is zero.

In a generic step of our algorithm we receive a triangle Δ as input. If the number of lines from L intersecting Δ is greater than εm then we apply the cutting lemma with the parameter s to partition Δ into $h = O(s^2)$ smaller triangles $\Delta_1, \ldots, \Delta_h$. For each triangle Δ_i we compute $|\Delta_i|$ by adding $|\Delta|$ to the number of half-strips that intersect Δ and cover Δ_i. Hence, if n half-strips intersect Δ then we can compute $|\Delta_i|$ for $1 \le i \le h$ in $O(ns^2)$ time.

If the number of lines from L intersecting Δ is at most εm, so is the number of half-strips from H. If $|\Delta| \ge m$, then any disc of radius r and center within Δ contains at least m entities and forms an approximate convergence pattern; it is therefore reported by choosing any point inside it as the disc center. If $|\Delta| < m$, then any disc of radius r and center within Δ contains less than $m + \varepsilon m$ entities and we do not report these patterns.

Theorem 2 *Given a set of n moving entities, a radius r, the minimum size $m \le n$ for a subset to form a pattern, and a positive constant ε. Using a $(1 + \varepsilon)$-approximation with respect to the minimum size of a subset to form a convergence pattern, one can compute:*

1. *The existence of convergence patterns in $O(n^{2+\delta}/(\varepsilon m))$ time, for any constant $\delta > 0$.*
2. *The approximate largest convergence pattern in $O(n^{2+\delta}/(\varepsilon m))$ time, for any constant $\delta > 0$.*
3. *All convergence patterns in $O(n^{2+\delta}/(\varepsilon m) + N)$ time, where N is the size of the output and δ is any positive constant.*

Proof Using the cutting lemma the time complexity can be described by the following recurrence: $T(n) = O(ns^2) + h \cdot T(n/s)$ if $n \ge \varepsilon m$, and $T(n) = O(n)$ if $n \le \varepsilon m$. For any $\delta > 0$, we can choose $h = O(s^2)$ to be some constant, and apply the master theorem [6] to prove that the recurrence solves to $O(n^{2+\delta}/(\varepsilon m))$. □

4.2 Different patterns and postprocessing

The algorithm that we presented determines a set of disjoint triangles such that: (1) each point in some triangle lies inside at least m half-strips, and (2) two points in the same triangle lie in the same half-strips with the possible exception of at most εm of them. The triangles can have widely varying diameter: from much smaller to much larger than r. (the latter can happen when m or more half-strips are nearly parallel). For every triangle, we choose one point inside that is the center of a circle containing at least m entities. Consequently, for every pattern with at least $(1 + \varepsilon)m$ entities that is not reported, there is a pattern with at least m entities the same which is reported. This pattern need not be spatially close to the not reported pattern, however.

It can be the case that two points in different triangles lie in the same subset of half-strips. Therefore, we might report different circles which in fact contain the same entities. These circles can even be very far apart. To assure that the same pattern is reported only once, we have to look at all entities of all patterns that we found explicitly, which can take $O(Nn)$ time for N patterns consisting of $O(n)$ entities each.

As an alternative to guarantee different patterns, we can apply the exact $O(n^2)$ time algorithm of Laube et al. [15] to find all $N = O(n^2)$ patterns with at least m entities, and then apply the independent set formulation on the centers of the circles as in Section 2. This yields the property that every not reported pattern is spatially close to some reported pattern. The running time is $O(n^2 + N \log^2 N)$.

5 Encounter

This section discusses the encounter pattern. Assume that a set of n entities is given, as well as a time step τ_j, a radius r and the minimum size $m \leq n$ of a subset required to form a pattern. We consider how to report all patterns, consisting of some location specified by a point p and radius r, a time τ, and a subset $S' \subseteq S, |S'| \geq m$, of entities that are within distance r from p at time τ, where $\tau > \tau_j$.

We model the problem as a 3D geometric problem by adding time as the third dimension to the position of each entity. We create a half-line for each entity, starting at the plane $z = \tau_j$ and extending upwards. The half-line starts at the location of the entity at time τ_j and contains a point $(x, y, \tau_{j'})$, where (x, y) is the known location at time step $\tau_{j'}$ with $j' > j$. The slope of the half-line with respect to the horizontal plane represents speed.

5.1 Exact: Find all

For an exact algorithm that finds all patterns, we start out with the set of half-lines just described. For any entity p_i, the region of all points that are within distance r from p_i at some moment in time is represented by a cylinder-like region, such that every cross-section with a horizontal plane is a disc of radius r.

The subdivision of space induced by the n cylinders consists of $O(n^3)$ cells, which is a tight bound in the worst case. For any point inside a cell, the subset of cylinders containing that point is the same, and hence, also the subset of entities that are within distance r at a given time. If the cell is inside at least m cylinders, then it represents a pattern.

One way to convert this idea into an algorithm is the following. Take the cylinder-like region of one entity p_i and call it C_i. All other cylinders $C_1, \ldots, C_{i-1}$, $C_{i+1}, \ldots, C_n$ intersect it in saddle-like curves. Build the arrangement of these curves on the boundary of the cylinder C_i. It has quadratic complexity and can be constructed in quadratic expected time [10], [17]. More precisely, the running time is $O(n \log n + k)$ expected, where k is the number of intersection points of the curves on C_i. We then traverse the arrangement on C_i and determine for every cell (a curved 2D facet) how many cylinders contain it. Using the fact that two adjacent cells have a count that differs by only 1, we can fill in the numbers in $O(n + k)$ time. We add one more, for C_i itself, so that the arrangement represents the counts for the 3D cells inside the cylinder C_i. We do this for all cylinders, resulting in an $O(n^2 \log n + K) = O(n^3)$ expected time bound, where K is the total size of all arrangements that were built. The storage requirements for the algorithm are $O(n + k_{\max}) = O(n^2)$.

The cubic running time is not particularly efficient, but the quadratic working storage requirement is an even bigger problem. Below we present an $O(n^3 \log n)$ time algorithm that uses only linear storage.

Consider a pair of half-lines ℓ_i and ℓ_j from the set. At any time, there are at most two circles with radius r that have a point $p_i \in \ell_i$ and a point $p_j \in \ell_j$ on the boundary (see Fig. 7 (left)). Such circles exist at any time when the (horizontal) distance between ℓ_i and ℓ_j is less than $2r$. There is only one interval in time where this occurs for ℓ_i and ℓ_j. Let us consider one of the two discs, and the swept volume it makes in the relevant time interval. This volume is a cylinder-like shape with a curved central axis. The pair of lines ℓ_i and ℓ_j gives rise to two such volumes which we denote by $V_{ij,1}$ and $V_{ij,2}$ (see Fig. 7 (right)).

Let $V = V_{ij,1}$; later we will test $V_{ij,2}$ in the same way. Any third half-line ℓ_h can intersect V in at most two disjoint time intervals. When at least $m - 2$ intervals intersect V at the same time, we have found a pattern. This will happen for the first time at an endpoint of an interval, which is an intersection point of some half-line with V.

The algorithm to find all patterns is as follows. For any two half-lines ℓ_i and ℓ_j (for entities p_i and p_j), compute the volumes $V_{ij,1}$ and $V_{ij,2}$. For each of these, compute all intervals of intersections with the other half-lines and consider only the time-interval. We sort the endpoints of the intervals by time, and traverse them in increasing order of time. Every endpoint of an interval is an addition of an entity to a subset within radius r, or the removal of an entity from a subset. We can report all subsets of size at least m.

The algorithm takes $O(n^3 \log n)$ time to detect all patterns, and $O(n^4)$ time in the worst case if we report all patterns explicitly, with the whole subset involved. If we only report the time and place of a pattern, we spend $O(n^3)$ time on reporting, and the algorithm takes $O(n^3 \log n)$ time overall.

Theorem 3 *All encounter patterns involving at least m entities can be found in $O(n^3 \log n + N)$ time using linear storage, where N is the time needed to report the output. The patterns can also be found in $O(n^3 + N)$ expected time using $O(n^2)$ storage.*

Alternatively, we can identify the first subset in which p_i and p_j participate. Since we find the first pattern for p_i and every other entity, we can select the first one of these to get the first pattern for p_i. Hence, we can find the first pattern for every entity using $O(n^3 \log n)$ time overall.

Fig. 7 Two discs of radius
r through two points (*left*).
Swept volume $V = V_{ij,1}$ (*right*)
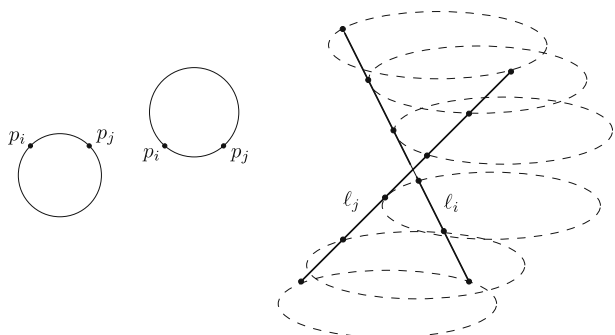
5.2 Exact: Detect

Next we show that we can detect a pattern in $O(mn^2 \log n)$ time. If m is considerably smaller than n, then this method is more efficient than using an adaptation of the algorithm that finds all patterns. We again make use of cylinders centered at each one of the half-lines. This time the intersection of a cylinder with a horizontal plane is a disc of radius $2r$. Many half-lines may have an interval of intersection with a cylinder. However, $m - 1$ half-lines that intersect a cylinder at the same time need not form a pattern, because a cylinder now has twice the radius. Our method is based on the following:

Lemma 3 *Let ℓ_i be a half-line for entity p_i, and let V_i be the region of points that are within distance $2r$ of p_i at some time. If at least $7m$ half-lines intersect V_i at the same time, then a pattern of size at least m exists (this pattern need not include p_i itself).*

Proof Follows by a packing argument. A disc of radius $2r$ can be covered by seven discs of radius r. By the pigeon-hole principle, one of the radius-r discs must be intersected by at least $7m/7 = m$ half-lines [29]. □

Globally, our detection algorithm works as follows. For every entity p_i, consider its half-line ℓ_i and cylinder V_i as defined above. For all other entities, compute the interval of intersection and consider the time-dimension. Sort the endpoints of the interval by time and traverse the endpoints as before. If we discover that at some moment in time there are at least $7m$ half-lines in V_i, then we stop and report that a pattern exists. If we have tested all entities and have not discovered a pattern yet, we use a different algorithm that makes use of the fact that for any cylinder, at most $O(m)$ half-lines can intersect it at the same time. In fact, the algorithm is similar to the previous, $O(n^3 \log n)$ time algorithm for all patterns, but it is initialized differently. Observe that so far, we have spent only $O(n^2 \log n)$ time.

Consider one cylinder V_i and the time intervals $I_1, \ldots, I_{n'}$ of half-lines that intersect V_i. Consider the endpoints sorted by time, which we have already done. For every interval I_j, define the subset $overlap(I_j) \subseteq \{I_1, \ldots, I_{j-1}, I_{j+1}, \ldots, I_{n'}\}$ of intervals that have a non-empty overlap with I_j.

Lemma 4 *Given a cylinder V_i for which at no time there are $7m$ or more half-lines inside, all subsets* overlap(I_j) *together have size $O(mn)$.*

Proof When two intervals I_j and I_h overlap, then they occur in the subset of each other. We charge both occurrences to the smaller size subset, that is, if $|overlap(I_j)| \leq |overlap(I_h)|$, then we charge both occurrences to I_j and otherwise we charge both to I_h.

Consider the interval with the smallest overlap subset and assume without loss of generality that it is I_j. We may assume that no interval I_h is properly contained in I_j, otherwise we take that interval as the smallest instead. We claim that $|overlap(I_j)| < 14m$. Assume the contrary for a contradiction. Observe that all intervals in $overlap(I_j)$ contain the left or the right endpoint of I_j (or both). Hence, the left or the right endpoint is covered by at least $7m$ intervals. But by assumption, there cannot be $7m$ half-lines in V_i at the same time. We conclude that $|overlap(I_j)| < 14m$. We charge

all overlaps that include I_j to I_j and since we charge twice per overlap, I_j is charged at most $28m$. Now we remove I_j from all subsets $overlap(..)$ in which it occurs and we repeat the argument until no intervals remain. Every interval is charged $O(m)$ times. Since there are up to $n - 1$ intervals $I_1, \ldots, I_{n'}$, we charge $O(mn)$ in total. $\square$

Our algorithm computes all subsets $overlap(..)$ in $O(mn)$ time, finds the smallest one, and runs the exact, all patterns algorithm on p_i and p_j and the subset $overlap(I_j)$. If we do not find a pattern, then we remove I_j from all other subsets and continue with the interval with the next smallest $overlap(..)$ subset. If we maintain appropriate pointers, then we can perform these updates in $O(m)$ time and find the next smallest in $O(\log n)$ time. Specifically, we store an overlap subset $overlap(I_j)$ by a counter and a pointer to a doubly-linked list. If I_h is in $overlap(I_j)$, it is a list element. Also, I_j is a list element in the list for $overlap(I_h)$. We create cross-pointers between them. Every list element also stores a back-pointer to the (representation of) $overlap(I_j)$. The counters store the number of elements in the lists.

When we treat $overlap(I_j)$, we traverse its list, use the cross-pointers to access all occurrences of I_j in other lists $overlap(I_h)$, delete this occurrence, and we use the back-pointer to decrease the counter of $overlap(I_h)$ by one.

Finally, all subsets are stored in a Fibonnacci Heap [6] on the counters (current size of the subset). We can extract the minimum from a Fibonnacci Heap in $O(\log n)$ time. When we decrease a counter by one, we perform a Decrease-Key on that subset, which takes $O(1)$ amortized time. For V_i and all intervals, we spend $O(mn + n \log n)$ time, and for all cylinders this is $O((m + \log n) \cdot n^2)$ time.

Theorem 4 *Detection of the existence of some encounter pattern involving at least m entities from a set of n entities can be done in $O((m + \log n) \cdot n^2)$ time.*

5.3 Exact: Find largest

We can use the detection algorithm to search for the largest pattern, which is the largest subset of entities that are expected to come within a disc of radius r. Let M be the (unknown) size of this largest subset. We first guess $m = 2$ and run the detection algorithm. If a pattern is detected, we know that $M \geq m$, we set m to be $2m$ and repeat (run the detection algorithm). As soon as detection fails for some m, we know that $m/2 \leq M < m$. Using a binary search in this interval, we determine the exact value of M.

The detection algorithm is called $O(\log m) = O(\log M)$ times, and hence the total running time is $O((M + \log n) \cdot n^2 \log M)$.

Theorem 5 *The largest subset of entities that are involved in an encounter pattern can be determined in $O((M + \log n) \cdot n^2 \log M)$ time.*

5.4 Approximating the radius

The cubic time algorithms to find all patterns, or find the first pattern for each entity, are rather time consuming. If we let go of the precise value of r, the radius of the disc needed to form a pattern, then we can obtain a near-quadratic time algorithm. The value of r need only be relaxed slightly: choose any constant value $\varepsilon > 0$, then we

will be sure to find a pattern consisting of m entities if they lie in a region of radius at most r, and we may or may not find any pattern with a region of radius between r and $(1 + \varepsilon) \cdot r$. The algorithm runs in $O((n^2 \log n)/\varepsilon)$ time.

The general idea is that the number of cylinders that we are going to process will be $4/\varepsilon$ per half-line. Let an entity p_i and its half-line ℓ_i be given. Consider the cylinder-like shape C_i with ℓ_i as the center and such that every cross-section with a horizontal plane is a disc of radius r. Place $4/\varepsilon$ evenly spaced markers, denoted $v_1, \ldots, v_{4/\varepsilon}$, on some cross-section boundary, and consider the half-lines $\ell_{i,1}, \ldots, \ell_{i,4/\varepsilon}$ containing $v_1, \ldots, v_{4/\varepsilon}$ and in the boundary of C_i (they are parallel to ℓ_i). For each half-line $\ell_{i,j}$ through v_j we define the cylinder-like shape $C_{i,j}$ such that every cross-section is a disc of radius $(1 + \varepsilon)r$ (see Fig. 8 (left)). Each cylinder $C_{i,j}$ is processed in the same way as described for the exact problem: we determine the time intervals where other half-lines intersect $C_{i,j}$, and find subsets of size at least $m - 1$. The time complexity is, just as before, $O(n \log n)$ per cylinder. Since the number of cylinders is $O(n/\varepsilon)$ we get a total running time of $O(\frac{n^2}{\varepsilon} \log n)$.

No region of radius $(1 + \varepsilon)r$ with less than m entities is reported by the algorithm, hence it suffices to prove that the algorithm returns all regions of radius r with at least m entities. This is proven by showing that every horizontal disc D of radius r and a half-line ℓ_i intersecting its perimeter must lie entirely within one of the cylinders of ℓ_i that is processed. Consider D and let $C_{i,j}$ be the cylinder treated for ℓ_i that is closest to D, that is, whose center v_j is closest to the center c_D of D. Let D' be the horizontal disc of $C_{i,j}$ at the same moment of time as D, see Fig. 8 (right). We need to show that $D \subset D'$. Note that the angle between the two horizontal segments from the centers of D and D' to ℓ_i is bounded by $\varepsilon\pi/4$, and the distance between the centers of D and D' is at most $2r \sin(\varepsilon\pi/8) < r\varepsilon\pi/4 < \varepsilon r$. Since the radius of D' is $(1 + \varepsilon)r$ it follows that D must lie within D'. We have proven the following result.

Theorem 6 *Given a set of n moving entities, a radius r, the minimum size $m \leq n$ for a subset to form a pattern, and a positive constant ε. Using a $(1 + \varepsilon)$-approximation with respect to the radius of the encounter pattern in 2D, one can compute:*

1. *The existence of approximate encounter patterns in $O(\frac{n^2}{\varepsilon} \log n)$ time.*
2. *The approximate largest encounter pattern in $O(\frac{n^2}{\varepsilon} \log n)$ time.*
3. *All approximate encounter patterns in $O(\frac{n^2}{\varepsilon} \log n + N)$ time, where N is the size of the output.*

Fig. 8 The cylinder $C_{i,j}$ for p_i (*left*). The *center* c_D of D is closest to v_j (*right*)
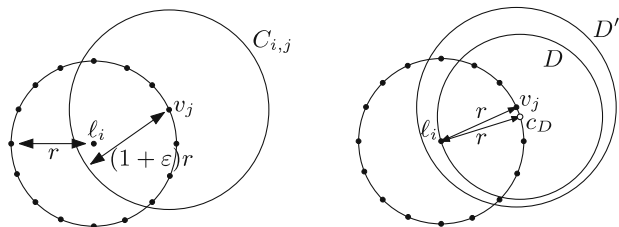
5.5 Different patterns and postprocessing

Both the exact and the approximate *find all* problem for encounter can find the same pattern many times. We cannot directly use the definition and technique from Section 2 for sufficiently different patterns, because patterns are represented by horizontal circles in three-dimensional space. Circles at the same x, y-location but at different time instances may represent completely different patterns.

One option is to use a three-dimensional definition of different patterns: two patterns are different if the distance of the centers of their defining circles is at least some value ρ. The value ρ is a distance in the time–space, and therefore we have to find a suitable scaling of the time–axis, depending on the data that is analyzed. For this reason, the three-dimensional definition of different patterns is less attractive than the two-dimensional version. Still, it is a relatively simple extension and it is unclear at present how to define different patterns appropriately.

To identify different patterns in a set of N retrieved patterns we can use the approach of computing a maximal independent set in the graph $\mathcal{G}$, as before. This time we need a three-dimensional nearest neighbor query structure. This would lead to an $\Omega(N^2)$ time bound, because the three-dimensional Voronoi diagram has quadratic complexity in the worst case. It is more practical to use an approximate nearest neighbor query structure [2], [4], which, when combined with the logarithmic method [3], [21], gives an $O(N \log^2 N)$ time algorithm. For any constant $\delta > 0$, any two patterns reported have circle centers that are at distance at least ρ, while for every pattern that was not reported, there is a pattern within distance $(1 + \delta)\rho$ that is reported.

6 Conclusion

In this paper we described efficient approximation algorithms to compute four spatio-temporal patterns, namely flock, leadership, convergence, and encounter. Approximation algorithms—a technique frequently used in computational geometry— are ideally suited for the algorithmic problems arising from these patterns. The approximation algorithms presented in this paper are significantly faster than their exact counterparts. During interactive detection of patterns, such efficiency savings are important.

Alternatively, our results imply that larger sets of data can be analyzed in the same amount of time. We observe from Table 1 that our algorithms are especially fast if the quality of the approximation is not so important, that is, the subset size, or radius of the region, is allowed to be, say, up to 20% off from the specified size or radius.

We have also presented a possible solution to the problem of detecting many patterns that are very similar. However, improvements or better solutions to this problem may exist. Any particular definition of the difference of two patterns will lead to an algorithmic problem of efficiently computing a set of different patterns that is "close" to all occurring patterns. More research to address this topic is needed.

References

1. B. Aronov and S. Har-Peled. "On approximating the depth and related problems," in *Proc. of 16th ACM-SIAM Symposium on Discrete Algorithms*, pp. 886–894, 2005.
2. S. Arya, D.M. Mount, N.S. Netanyahu, R. Silverman, and A.Y. Wu. "An optimal algorithm for approximate nearest neighbor searching fixed dimensions," *Journal of the ACM*, Vol. 45(6):891–923, 1998.
3. J.L. Bentley. "Decomposable searching problems," *Information Processing Letters*, Vol. 8:244–251, 1979.
4. T.M. Chan. "Approximate nearest neighbor queries revisited," *Discrete and Computational Geometry*, Vol. 20(3):359–373, 1998.
5. B. Chazelle. "Cutting hyperplanes for divide-and-conquer," *Discrete and Computational Geometry*, Vol. 9(2):145–158, 1993.
6. T.H. Cormen, C.E. Leiserson, R.L. Rivest, and C. Stein. Introduction to Algorithms. 2nd edition, MIT: Cambridge, MA, 2001.
7. J. Erickson. "New lower bounds for convex hull problems in odd dimensions," *SIAM Journal of Computing*, Vol. 28(4):1198–1214, 1999.
8. A.U. Frank, J.F. Raper, and J.-P. Cheylan (Ed.). *Life and Motion of Spatial Socio-economic Units*, 353–373, Taylor & Francis: New York, 2001.
9. L.J. Guibas, M.H. Overmars, and J.-M. Robert. "The exact fitting problem for points," *Computational Geometry—Theory and Applications*, Vol. 6:215–230, 1996.
10. D. Halperin. "Arrangements," in J.E. Goodman and J. O'Rourke (Eds.), *Handbook of Discrete and Computational Geometry*, Chapter 24, 529–562, Chapman & Hall: London/CRC: Boca Raton, FL, 2004.
11. J. Han and M. Kamber. Data Mining: Concepts and Techniques. Morgan Kaufmann, 2001.
12. S. Iwase and H. Saito. "Tracking soccer player using multiple views," in *Proc. of the IAPR Workshop on Machine Vision Applications (MVA02), Nara, Japan*, pp. 102–105, 2002.
13. G. Kollios, S. Sclaroff, and M. Betke. "Motion mining: Discovering spatio-temporal patterns in databases of human motion," in *Proc. of the ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery, Santa Barbara, CA*, pp. 25–32, 2001.
14. P. Laube and S. Imfeld. "Analyzing relative motion within groups of trackable moving point objects," in *GIScience 2002, no. 2478, Lecture Notes in Computer Science*, pp. 132–144, Springer, Berlin Heidelberg New York, 2002.
15. P. Laube, M. van Kreveld, and S. Imfeld. "Finding REMO—Detecting relative motion patterns in geospatial lifelines," in *Developments in Spatial Data Handling: Proc. of the 11th International Symposium on Spatial Data Handling, Leicester, London*, pp. 201–214, Springer, Berlin Heidelberg New York, 2004.
16. H.J. Miller and J. Han, (Eds.). *Geographic Data Mining and Knowledge Discovery*, 372, Taylor & Francis:London, 2001.
17. K. Mulmuley. Computational Geometry: An Introduction Through Randomized Algorithms. Prentice Hall, 1994.
18. S. Openshaw, M. Charlton, C. Wymer, and A. Craft. "Developing a mark 1 geographical analysis machine for the automated analysis of point data sets," *International Journal of Geographical Information Systems*, Vol. 1:335–358, 1987.
19. D. O'Sullivan and D.J. Unwin. Geographic Information Analysis. Wiley, 2003.
20. T. Ott and F. Swiaczny. Time-integrative Geographic Information Systems. Springer, 2001.
21. M.H. Overmars. "The design of dynamic data structures," in *Lecture Notes Computer Science, Vol. 156*, Springer, Berlin Heidelberg New York, 1983.
22. Porcupine caribou herd satellite collar project. http://www.taiga.net/satellite/.
23. J. Roddick, K. Hornsby, and M. Spiliopoulou. "An updated bibliography of temporal, spatial, and spatio-temporal data mining research," in *TSDM 2000, no. 2007, Lecture Notes in Artificial Intelligence,* pp. 147–163. Springer, Berlin Heidelberg New York, 2001.
24. H. Samet. The Design and Analysis of Spatial Data Structures. Addison-Wesley, 1990.
25. C.-B. Shim and J.-W.Chang. "A new similar trajectory retrieval scheme using k-warping distance algorithm for moving objects," in *Proc. of the 4th International Conference on Advances in Web-age Information Management, (WAIM 2003), no. 2762, Lecture Notes in Computer Science,* pp. 433–444, Springer, Berlin Heidelberg New York, 2003.
26. A. Silberschatz and A. Tuzhilin. "What makes patterns interesting in knowledge discovery systems," *IEEE Transactions on Knowledge and Data Engineering*, Vol. 8:970–974, 1996.

27. G. Sinha and D.M. Mark. "Measuring similarity between geospatial lifelines in studies of environmental health," *Journal of Geographical Systems*, Vol. 7:115–136, 2005.
28. N. Sumpter and A.J. Bulpitt. "Learning spatio-temporal patterns for predicting object behaviour," *Image Vision and Computing*, Vol. 18(9):697–704, 2000.
29. G. Fejes Tóth. "Packing and covering," in J.E. Goodman and J. O'Rourke, (Eds.), *Handbook of Discrete and Computational Geometry*, Chapter 2, 2nd edition, pp. 25–52, Chapman & Hall: London/CRC: Boca Raton, 2004.
30. M. Vlachos, G. Kollios, and D. Gunopulos. "Discovering similar multidimensional trajectories," in *Proc. of the 18th International Conference on Data Engineering (ICDE'02), San Jose, CA*, pp. 673–684, 2002.

Joachim Gudmundsson received his Ph.D. in computer science from Lund University in Sweden. During 2001-2004 he was a postdoctoral researcher at Utrecht University and at the Technical University of Eindhoven in the Netherlands. Since 2005 he works as a senior researcher at National ICT Australia Ltd in Sydney, and is currently heading the DMiST-project (Data Mining in Spatio-Temporal sets) at NICTA. His research interests are computational geometry and approximation algorithms.

Marc van Kreveld   received his Ph.D. in computer science from Utrecht University in the Netherlands. He was a postdoctoral researcher at the school of computer science of McGill University, Montreal, Canada, from 1992 until 1993. Since 1993 he works at the department of information and computing sciences of Utrecht University, currently as an associate professor. His main research interests are computational geometry and algorithms for GIS.

Bettina Speckmann received her Ph.D. in computer science from the University of British Columbia, Vancouver, Canada, in 2001. She was a postdoctoral researcher at the institute of theoretical computer science at ETH Zurich from 2001 until 2003. Since 2003 she is an assistant professor at the department of mathematics and computer science of the TU Eindhoven, the Netherlands. Her main research interests are computational geometry and algorithms for automated cartography.