Efficient Detection of Motion Patterns in Spatio-Temporal Data Sets

Joachim Gudmundsson*
Dep. of Mathematics & Computer Science
TU Eindhoven
h.j.gudmundsson@tue.nl

Marc van Kreveld
Inst. of Information & Computing Sciences
Utrecht University
marc@cs.uu.nl

Bettina Speckmann
Dep. of Mathematics & Computer Science
TU Eindhoven
speckman@win.tue.nl

Abstract

Moving point object data can be analyzed through the discovery of patterns. We consider the computational efficiency of detecting four such spatio-temporal patterns, namely flock, leadership, convergence, and encounter, as defined by Laube et al., 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.

Categories and Subject Descriptors: F.2.2 [Analysis of Algorithms and Problem Complexity]: Nonnumerical Algorithms and Problems—*Geometrical problems and computations*

General Terms: Algorithms, Theory

Keywords: Spatio-temporal patterns, Moving objects, Geometric algorithms, Approximation algorithms

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 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 [15]. The analysis of moving objects also has applications in sports (e.g., soccer players [8]) and in socio-economic geography [5].

The general objective of spatio-temporal data mining [12, 16] is to discover interesting patterns in spatio-temporal data, which includes moving point object data. There is ample research on data mining of moving objects (e.g., [9, 18, 19, 21]) in particular on the discovery of similar trajectories or clusters. 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 Figure 1). For brevity, we call moving point objects *entities* from now on.

The REMO framework (RElative MOtion) was developed by Laube and Imfeld [10] 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. [11] extend the framework by not only including direction of motion, but also location itself. They define several spatio-temporal patterns, including *flock*, *leadership*, *convergence*, and *encounter*, and give algorithms to compute them efficiently. We formalize these patterns below.

Each pattern can occur for a subset of the entities at a given time step. The input consists of n entities, each with t locations at consecutive time steps and 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 Figure 1). We formalize "being 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 con-

*Supported by the Netherlands Organisation for Scientific Research (NWO).

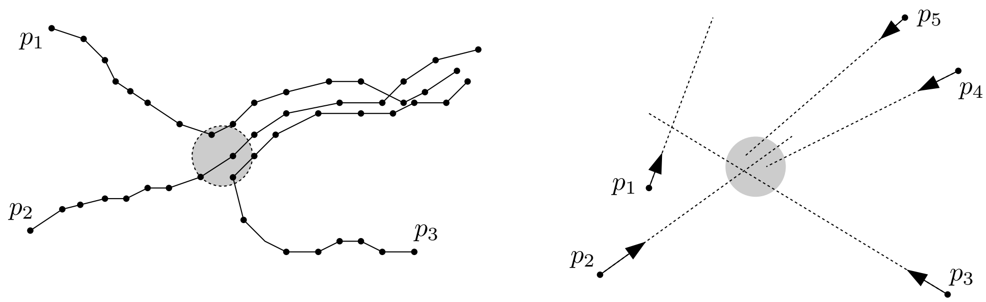

Figure 1: Left, a flock pattern at the eighth time step. It is also a leadership pattern with p_2 as the leader. Right, a convergence pattern for p_2, p_3, p_4, p_5.

vergence 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 meets in a small enough region. Formally, flock, leadership, convergence, and encounter patterns for some given set of entities with a position, direction, and speed 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 $\tau > 0$. 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 τ 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).

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 problems were already given in [11] 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 meters form as interesting a pattern as 19 caribou meeting in a circle with radius 51 meters. Therefore the problem of computing these patterns is ideally suited for approximation algorithms.

Recall that the input consists of locations of n entities at t consecutive time steps. We only look for patterns defined by these input locations and time steps, not for patterns defined by locations in between. This is referred to as the snapshot view of time [14]. If time is sampled sufficiently densely, this is no severe limitation. Furthermore, we concentrate on approximation algorithms with possibly slightly different radius and subgroup size. On the other hand, we will find the patterns convergence and encounter at any time instance, not only at the given time steps.

In the next three sections we describe efficient approximation algorithms for all four patterns where we let either the

Pattern	Exact (from [11])	Exact (new)	Approximate
Flock	$O(nm^2 + n \log n)$	-	$O(\frac{1}{\varepsilon^2} n \log \frac{1}{\varepsilon} + n \log n)$ (radius)
Leadership	$O(n\tau + nm^2 + n \log n)$	-	$O(n\tau + \frac{1}{\varepsilon^2} n \log \frac{1}{\varepsilon} + n \log n)$ (radius)
Convergence	$O(n^2)$	-	$O(n^{2+\delta}/(\varepsilon m))$ (subset)
Encounter	$O(n^4)$	$O(n^3)$ (all) $O((m + \log n)n^2)$ (detect) $O((M + \log n)n^2 \log M)$ (largest)	$O(\frac{1}{\varepsilon} n^2 \log n)$ (radius)

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

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 Flock and leadership

This section discusses the flock pattern and its extension, the leadership pattern. The leadership pattern is discussed only briefly at the end of the section, 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 [11] 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. [11] propose an algorithm that is based on higher-order Voronoi diagrams with a running time of $O(nm^2 + n\log n)$. 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.

2.1 Approximating the radius

We will use a *quadtree* [17] 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* [1] 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.

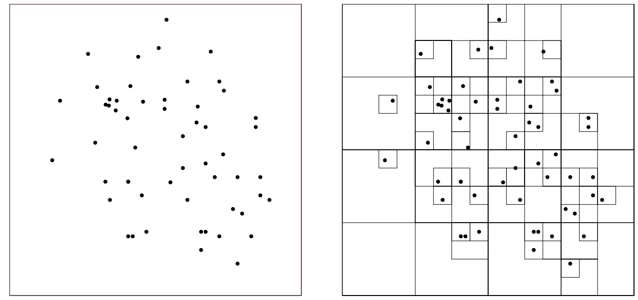

Figure 3: (a) The input set S contained in a square. (b) The arrangement $\mathcal{A}$ of the squares obtained from the quadtree for S.

Now consider a subset S of the input as described earlier. We already know that all entities in S move into 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 ν has side length less or equal to $\frac{\varepsilon}{4}r$. That is, we stop the recursion as soon as we reach a small enough side length. We then build the arrangement $\mathcal{A}$ of all squares in $\mathcal{T}$ (see Fig. 3). $\mathcal{A}$ 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 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

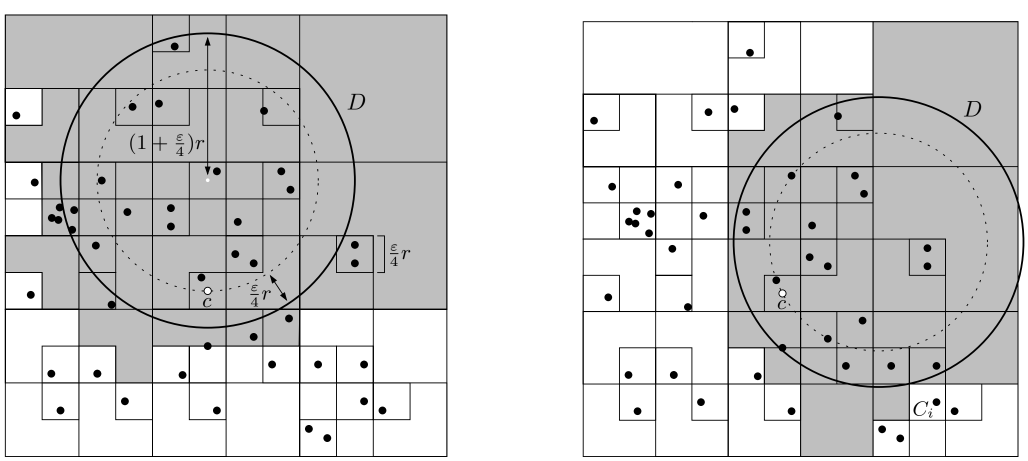

Figure 2: Sweeping $\mathcal{A}$ with a circle D of radius $(1+\frac{\varepsilon}{4})r$. Starting position of D (left), the non-empty cell C_i enters D (right).

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 1. We then sort the reported cells 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. 2. (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. 2 (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'.)

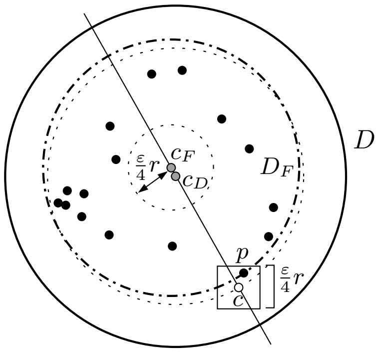

Figure 4: The center c_D of D lies on the line through c and c_F.

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 2 *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. *proof of the existence of flock patterns in $O(\frac{n}{\varepsilon^2}\log\frac{1}{\varepsilon} + n\log n)$ time.*
2. *all flock patterns in $O(\frac{n}{\varepsilon^2}\log\frac{1}{\varepsilon} + n\log n + N)$ time, where N is the number of reported flock patterns.*

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. □

To detect or find all leadership patterns we are given an additional parameter τ 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 $\tau - 1$ time steps. This takes $O(n\tau)$ 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 $O(n\tau)$ 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 $O(n\tau \cdot t)$ time).

3 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 radius r and the minimum size $m \leq n$ for a subset to form a pattern. If we draw a line from the current position of each entity that corresponds to its direction then we create a set of directed half-lines (see for example Fig. 1 (right)). In [11] Laube et al. show how to use this representation to detect convergence patterns in $O(n^2)$. They compute the 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 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. [6] 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 [4] shows that the problem of deciding if any three lines have a common intersection point has a lower bound of $\Omega(n^2)$ 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 all N regions that are visited by at least $(1+\varepsilon)m$ moving entities in $O(n^{2+\delta}/(\varepsilon m) + N)$ time for any constant $\delta > 0$. A region that is visited by less than m entities will not be reported, while a region visited by at least m entities and less than $(1+\varepsilon)m$ entities may, or may not, be reported.

3.1 Approximating the subset size

We are using the representation of the problem proposed in [11], 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 t (possibly unbounded) triangles $\Delta_1, \ldots, \Delta_t$ 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 3 (CUTTING LEMMA [2]) *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 $t = O(s^2)$ smaller triangles $\Delta_1, \ldots, \Delta_t$. 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 $1 \le i \le t$ in $O(ns^2)$ time.

Otherwise, the number of lines from L intersecting Δ is less than or equal to εm and so is the number of half-strips from H. If $|\Delta| \ge m$, then a disc of radius r and center within Δ forms an approximate convergence pattern and is therefore reported. Note that any disc of radius r and center within Δ forms an approximate convergence pattern, but only one is reported.

THEOREM 4 *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. *a proof of 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 number of reported convergence patterns and $\delta > 0$ is a constant.*

PROOF. Using the cutting lemma we get the following recursion: $T(n) = ns^2 + t \cdot T(n/s)$. If we set $t = O(s^2)$ to be a constant we have $T(n) = O(1)$ in the case when $n \le \varepsilon m$. Now we apply the master theorem [3] which solves the recursion to $O(n^{2+\delta}/(\varepsilon m))$, for any constant $\delta > 0$. □

4 Encounter

This section discusses the encounter pattern. Assume that a set of n entities is given, as well as a radius r and the minimum size $m \le n$ 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 t, and a subset $S' \subseteq S$ of entities that are within distance r from p at time t.

We model the problem as a 3D geometric problem by adding time as the third dimension to the position of each entity. This creates a half-line for each entity, starting at the plane $z = t_0$ and extending upwards. The slope of the half-lines with respect to the horizontal plane represents speed in this representation, and the point (x_i, y_i, t_i) on a half-line for the i-th entity tells that the i-th entity is expected to be at position (x_i, y_i) at time t_i.

4.1 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 [7, 13]. 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 $O(n^2 \log +K) = O(n^3)$ 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 only uses linear storage. We make use of the following observation:

OBSERVATION 5 *If a subset $S' \subseteq S$ of entities at time t' consists of at least m entities, then the pattern also occurs for a subset $S'' \subset S$ where $S' \subseteq S''$, at a time $t'' \le t'$ for which at least three entities define a circle of radius exactly r.*

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

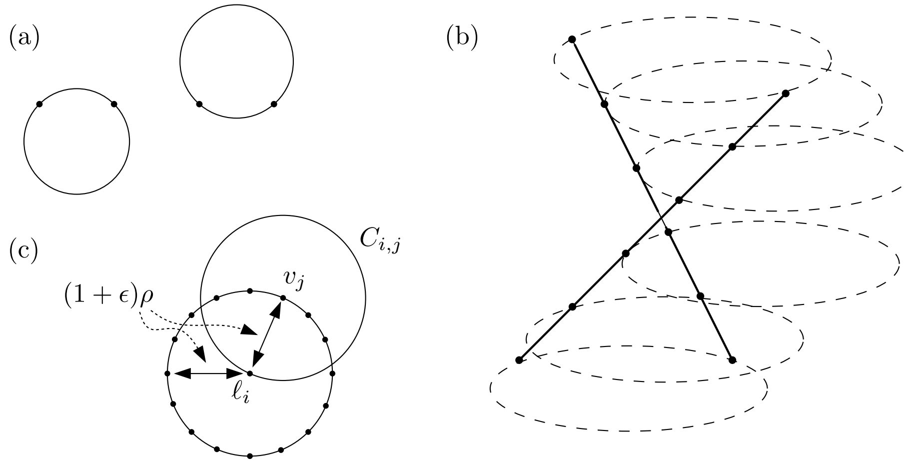

Figure 5: (a) Two discs of radius r through two points. (b) Swept volume $V = V_{ij,1}$. (c) The cylinder $C_{i,j}$ for p_i.

point of ℓ_i and a point of ℓ_j on the boundary (see Figure 5 (a)). 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 Figure 5 (b)).

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 6 *All encounter patterns of 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.

4.2 Exact: detect

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

Lemma 7 *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 at most 7 discs of radius r. By the pigeonhole principle, one of the radius-r discs must be intersected by at least $7m/7 = m$ half-lines [20]. □

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 8 *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 $|overlap(I_j)| < |overlap(I_h)|$, 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. □

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 [3] 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 9 *Detection of the existence of some encounter pattern of m entities from a set of n entities can be done in $O((m+\log n)\cdot n^2)$ time.*

4.3 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 10 *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.*

4.4 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 obtain a pattern, then we can obtain a near-quadratic time algorithm. The value of r need only be relaxed slightly: choose any constant value $\epsilon > 0$, 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 $(1+\epsilon)\cdot r$. The algorithm runs in $O((n^2 \log n)/\epsilon)$ 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 Figure 5 (c)). 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 intersecting its perimeter must lie entirely within one of the cylinders of ℓ that is processed. Consider D and let $C_{i,j}$ be the cylinder treated for ℓ_i that is closest to D, i.e., whose center is closest to the center of D. Finally, let D' be the horizontal disc of $C_{i,j}$ at the same moment of time as D. 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 $\epsilon\pi/4$, and the distance between the centers of D and D' is at most $2r\sin(\epsilon\pi/8) < r\epsilon\pi/4 < \epsilon r$. Since the radius of D' is $(1+\varepsilon)r$ it follows that D must lie within D', hence, we have proven the following theorem.

THEOREM 11 *An $(1+\varepsilon)$-approximation of the radius of the 2D encounter problem can be computed in time $O(\frac{n^2}{\varepsilon}\log n + N)$, where N is the time needed to report the patterns.*

Therefore we can conclude:

THEOREM 12 *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. *proof of the existence of 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 encounter patterns in $O(\frac{n^2}{\varepsilon} \log n + N)$ time, where N is the number of reported encounter patterns.*

5 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.

References

[1] 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*, 45(6):891–923, 1998.

[2] B. Chazelle. Cutting hyperplanes for divide-and-conquer. *Discrete and Computational Geometry*, 9(2):145–158, 1993.

[3] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. MIT Press, Cambridge, MA, 2nd edition, 2001.

[4] J. Erickson. New lower bounds for convex hull problems in odd dimensions. *SIAM Journal of Computing*, 28:1198–1214, 1999.

[5] A.U. Frank, J.F. Raper, and J.-P. Cheylan, editors. *Life and motion of spatial socio-economic units*. Taylor & Francis, London, 2001.

[6] L. J. Guibas, M. H. Overmars, and J.-M. Robert. The exact fitting problem for points. *Computational Geometry - Theory and Applications*, 6:215–230, 1996.

[7] D. Halperin. Arrangements. In J.E. Goodman and J. O'Rourke, editors, *Handbook of Discrete and Computational Geometry*, chapter 24, pages 529–562. Chapman & Hall/CRC, Boca Raton, 2nd edition, 2004.

[8] S. Iwase and H. Saito. Tracking soccer player using multiple views. In *Proc. IAPR Workshop on Machine Vision Applications (MVA02)*, 2002.

[9] G. Kollios, S. Sclaroff, and M. Betke. Motion mining: discovering spatio-temporal patterns in databases of human motion. In *Proc. ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery*, 2001.

[10] P. Laube and S. Imfeld. Analyzing relative motion within groups of trackable moving point objects. In *GIScience 2002*, number 2478 in Lect. Notes in Computer Science, pages 132–144, Berlin, 2002. Springer.

[11] P. Laube, M. van Kreveld, and S. Imfeld. Finding REMO – detecting relative motion patterns in geospatial lifelines. In *Proc. 11th Int. Symp. on Spatial Data Handling*, 2004. To appear.

[12] H.J. Miller and J. Han, editors. *Geographic Data Mining and Knowledge Discovery*. Taylor & Francis, London, 2001.

[13] K. Mulmuley. *Computational Geometry: An Introduction Through Randomized Algorithms*. Prentice Hall, Englewood Cliffs, NJ, 1994.

[14] T. Ott and F. Swiaczny. *Time-Integrative Geographic Information Systems*. Springer, Berlin, 2001.

[15] Porcupine caribou herd satellite collar project. `http://www.taiga.net/satellite/`.

[16] J. Roddick, K. Hornsby, and M. Spiliopoulou. An updated bibliography of temporal, spatial, and spatio-temporal data mining research. In *TSDM 2000*, number 2007 in Lect. Notes in Artificial Intelligence, pages 147–163, Berlin, 2001. Springer.

[17] H. Samet. *The design and analysis of spatial data structures*. Addison-Wesley, 1990.

[18] C.-B. Shim and Jae-Woo Chang. A new similar trajectory retrieval scheme using k-warping distance algorithm for moving objects. In *Proc. 4th Int. Conf. on Advances in Web-Age Information Management, (WAIM 2003)*, number 2762 in Lect. Notes in Computer Science, pages 433–444, 2003.

[19] N. Sumpter and A.J. Bulpitt. Learning spatio-temporal patterns for predicting object behaviour. *Image Vision Comput.*, 18:697–704, 2000.

[20] G. Fejes Tóth. Packing and covering. In J.E. Goodman and J. O'Rourke, editors, *Handbook of Discrete and Computational Geometry*, chapter 2, pages 25–52. Chapman & Hall/CRC, Boca Raton, 2nd edition, 2004.

[21] M. Vlachos, G. Kollios, and D. Gunopulos. Discovering similar multidimensional trajectories. In *Proc. 18th Int. Conf. on Data Engineering (ICDE'02)*, 2002.