Algorithms for Hotspot Computation on Trajectory Data

Joachim Gudmundsson
School of Information Technologies
University of Sydney, Australia
joachim.gudmundsson@sydney.edu.au

Marc van Kreveld
Dept. of Information and Computing Sciences
Utrecht University, The Netherlands
m.j.vankreveld@uu.nl

Frank Staals
Dept. of Information and Computing Sciences
Utrecht University, The Netherlands
f.staals@uu.nl

ABSTRACT

We study one of the basic tasks in moving object analysis, namely the location of *hotspots*. A hotspot is a (small) region in which an entity spends a significant amount of time. Finding such regions is useful in many applications, for example in segmentation, clustering, and locating popular places. We may be interested in locating a minimum size hotspot in which the entity spends a fixed amount of time, or locating a fixed size hotspot maximizing the time that the entity spends inside it. Furthermore, we can consider the total time, or the longest contiguous time the entity spends in the hotspot. We solve all four versions of the problem. For a square hotspot, we can solve the contiguous-time versions in $O(n \log n)$ time, where n is the number of trajectory vertices. The algorithms for the total-time versions are roughly quadratic. Finding a hotspot containing relatively the most time, compared to its size, takes $O(n^3)$ time. Even though we focus on a single moving entity, our algorithms immediately extend to multiple entities. Finally, we consider hotspots of different shape.

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

trajectory, moving entity, hotspot, geometric algorithms

1. INTRODUCTION

In the last decade there have been many developments in trajectory data and their analysis. Due to increasingly simple and precise tracking technologies, large data sets of trajectories have been collected and are being analyzed. Objects that are tracked are as diverse as pedestrians, birds, insects, mammals, cars, ships, and hurricanes.

Broadly one can classify the analytical tools into two classes; algorithms for clustering, segmentation, flock detection, and trajectory similarity, and visual analytics techniques used to assist domain experts to analyze the data. In the former class we can distinguish general-purpose algorithms, and algorithms developed for specific application-driven tasks, or even for specific data sets. We briefly review the tasks and approaches in general-purpose trajectory analysis.

Similarity. The similarity of two trajectories can be captured by a function that takes two trajectories and returns a scalar: the higher the value, the higher the similarity. Similarity is usually viewed as the inverse of distance. There are several common, general-purpose distance measures for trajectories, including the Hausdorff distance [2], Frechet distance [3], Dynamic Time Warping [22], time-focused distance [28], and edit distance [34].

Besides similarity of whole trajectories, one could also define and compute similarity of subtrajectories of two trajectories [11], or self-similarity of subtrajectories within a single trajectory [10].

Clustering. Clustering is the process of partitioning trajectories into a usually small number of groups so that within each group, the trajectories are similar, but across different groups, they are dissimilar. Trajectory clustering has been studied in [16, 24, 28], for example.

The use of similarity measures to assist in clustering is clear. Once a similarity measure is selected, many of the standard clustering methods for point data can be used for trajectory data as well, like single-linkage and complete-linkage clustering [19]. For other clustering methods like k-means and k-medoids, we also need a process that determines a specific, typical trajectory from a set of trajectories.

Segmentation. An analysis method that operates on a single trajectory is segmentation. Segmentation involves splitting a trajectory into a number of subtrajectories that have certain characteristics [6, 12, 35].

Motion patterns. When we are analyzing a set of trajectories and are interested in interactions like joint motion or leadership, we speak of motion patterns in trajectories [14, 17, 20]. Several definitions of flocking [8, 18] and

SIGSPATIAL '13 November 05 - 08, 2013, Orlando, FL, USA.

http://dx.doi.org/10.1145/2525314.2525359.

various other group motions [9, 23, 33] have been given and algorithms for these have been suggested.

Interesting places. From a collection of trajectories one can identify (small) places that are visited by many different trajectories, places that are used by a single moving entity for a long duration, or places where many trajectories change their movement behavior (e.g., they all pause), see [7, 26, 29, 32, 33]. Such locations are called hotspots, popular places, stationary regions, stay points, or stops.

In this paper we focus on the last analysis type, the detection of interesting places in trajectory data. In particular, we give algorithms to identify regions where a moving object spends a large amount of time. We refer to such a region as a *hotspot*, and model it as a square whose location is not known yet. We distinguish several versions of the problem of finding square-shaped hotspots. Extensions to different hotspot shapes are considered as well. The problems we consider are:

1. the size of the square is fixed and we wish to find the placement that maximizes the time the entity spent inside. Here we allow the entity to leave the region and return to it later; all visits count for the duration.
2. We are given a duration and want to determine the smallest square and its placement so that the entity is inside for at least the given duration.
3. We consider problem 1, but now we are interested in *contiguous* presence inside the square, so only one of the visits to the square counts.
4. The same for problem 2, and finally
5. we do not fix duration nor square size, but optimize a relative measure that is the ratio of the duration and the square side length.

We do not assume that the hotspots are pre-defined; instead, we adopt the more generic view that no contextual data is present. This is in contrast to Alvares et al. [4] and Chawla and Verhein [33], who assume that a set of places or a superimposed grid is given that pre-defines potential regions of interest. Our work is different from research that uses the stop-and-move model [4, 26, 31], because we do not attempt to segment a trajectory at changes in movement type, nor do we try to semantically annotate a trajectory. Our methods allow the time of multiple visits to be added to obtain a hotspot: three visits of an hour can be considered more important than a single visit of two hours. Finally, our results are also different from research on popular places [7, 32] because the number of entities that visits the region plays no role. Benkert et al. [7] do not incorporate the duration of visits, so the retrieved popular places could be transit places used by many entities. Tiwari and Kaushik [32] consider only contiguous-time visits and choose their time intervals based on the trajectory vertices only, not the locations in between. In contrast, we consider a trajectory to be a continuous space-time object and not a sampling of a discrete set of points in space-time. This implies that entering and leaving a hotspot is not restricted to vertices and that our results are less influenced by sampling rate and regularity. We solve the hotspot identification problem as an algorithmic optimization problem and provide running time bounds. We present our results for a single trajectory, but they immediately extend to the case of multiple trajectories.

Before formalizing the problem further, we discuss various situations where our methods are useful, besides hotspot location itself.

First-passage time: In animal ecology, first-passage time is defined as the time taken for an animal to cross a circle of a given radius that is centered at some start time on the trajectory. It can be used as a measure for search effort along a trajectory, and a long first-passage time may indicate that there is a significant amount of food in an area [15, 21]. First-passage time is closely related to contiguous-time visit to a region, although the presence of food may be indicated better by total visit time of one or more animals.

Segmentation: When a trajectory is segmented into semantically meaningful parts, it may be important to use more advanced trajectory properties than speed and heading, for example. Segmentation can also be assisted using environment data, but this requires such data to be acquired and processed. Even when environment data is not available, we can pre-process the trajectory and determine areas where the entity spends significant amounts of time. The entering and leaving of such regions can be a reason to segment a trajectory there. In this situation we require consecutive time within the area, so that crossing the area without halting does not lead to unnecessary segmentation.

Clustering: In some clustering applications we may be interested in the places where an entity spends some time, but not in the routes taken by the entity between these places. To identify the similarity of two trajectories in this situation we must identify these places and ignore the rest. Our algorithm can do this, after which a method like dynamic time warping on the distances between these relevant places is done.

An example of such a situation is migration. We may find similarity in resting places more relevant than the routes travelled between these places.

Visualization: Our methods can be used iteratively to enrich a trajectory data set with symbols that indicate that one or more entities spend much time there. For example, after locating a fixed-size square region where most time is spent, we can remove all pieces of trajectories inside that square, and iterate to find the next longest-visited square region. This is possible because our algorithms do not require trajectories to be contiguous or complete. We can then show the top-10 places in terms of duration of visits.

Problem Statement. A trajectory $\mathcal{T}$ is a continuous function, mapping a time interval onto points in the plane. By scaling we can assume without loss of generality that the time interval is $[0, 1]$. We write $\mathcal{T}[t]$ for the point on $\mathcal{T}$ at time $t \in [0, 1]$. Furthermore, we denote the *subtrajectory* from time s to time t, with $s \leq t \in [0, 1]$, by $\mathcal{T}[s, t]$. In the remainder of this paper, we consider only piecewise linear trajectories. Thus, the image of $\mathcal{T}$ consists of n line segments, the *edges*, in the plane. The end points of these edges are the *vertices* of $\mathcal{T}$. By general position, we assume that there are no two vertices that have the same x-coordinate or the same y-coordinate. Our results do not depend on

this assumption but the description is considerably easier because of it.

With slight abuse of notation we use $\mathcal{T}$ to denote the trajectory as well as the set of edges $\{e_1, .., e_n\}$ of trajectory. An edge e between points u and v, denoted $e = \overline{uv}$, has length $\|e\|$. We use the same notation for line segments and (sub)trajectories.

Let $\mathcal{H} \subset \mathbb{R}^2$ denote the axis-parallel square hotspot with center c and side length $2r$. We refer to r as the *radius* of $\mathcal{H}$. The boundary of $\mathcal{H}$ is denoted by $\partial\mathcal{H}$. The function $\Upsilon(\mathcal{H}) = \sum_{e\in\mathcal{T}} \|e \cap \mathcal{H}\|$ describes the total length of the trajectory inside $\mathcal{H}$. Similarly, $\Phi(\mathcal{H}) = \max\{\|\mathcal{T}[s,t]\| \mid s,t \in [0,1] \wedge \mathcal{T}[s,t] \subseteq \mathcal{H}\}$ denotes the length of the longest (contiguous) subtrajectory in $\mathcal{H}$, and $\Psi(\mathcal{H}) = \Upsilon(\mathcal{H})/2r$ denotes the *relative* trajectory length in $\mathcal{H}$. We may write $\Upsilon(c,r) = \Upsilon(\mathcal{H})$, and $\Upsilon(c) = \Upsilon(c,r)$ if c and/or r is clear from the context. We do the same for Φ and Ψ. We can now formalize the five problems that we study as follows.

1. Given r, maximize $\Upsilon(\cdot, r)$ over all square placements.
2. Given L, find a smallest hotspot $\mathcal{H}^*$ with $\Upsilon(\mathcal{H}^*) \geq L$ over all square placements and sizes.
3. Given r, maximize $\Phi(\cdot, r)$ over all square placements.
4. Given L, find a smallest hotspot $\mathcal{H}^*$ with $\Phi(\mathcal{H}^*) \geq L$, over all square placements and sizes.
5. Find a hotspot $\mathcal{H}^*$ that maximizes Ψ, over all square placements and sizes.

We describe our algorithms by considering length rather than duration because it is more intuitive and easier to describe. Adapting the algorithms to consider duration is straightforward, however. We simply observe that all lemmas and algorithms still hold for *weighted* edges, and we can use the Euclidean and duration lengths of an edge to assign a weight to each edge so that maximizing weighted length inside $\mathcal{H}$ is the same as maximizing duration inside $\mathcal{H}$.

Results and Organization. Our algorithms that solve the above problems all use a key property of Υ, namely that it is a linear function in c and r. We prove this in Section 2, and discuss how the shape of $\mathcal{H}$ affects this property. In Section 3 we study problems 1 and 2, and show that we can solve them in $O(n^2)$ time and $O(n^2 \log^2 n)$ time, respectively. We focus on the contiguous-length variants, problems 3 and 4, in Section 4. We show that we can solve both problems in $O(n \log n)$ time. In Section 5 we present an algorithm to maximize the relative trajectory length Ψ that lies in $\mathcal{H}$, that is, we solve problem 5. This algorithm uses $O(n^3)$ time. We review some extensions, like multiple entities and different shapes for $\mathcal{H}$, in Section 6.

2. PRELIMINARIES

We start by showing that Υ (and also Φ) is a piecewise linear function, which is key in our algorithms.

LEMMA 1. *Consider a square hotspot $\mathcal{H}$ with center c and radius r. The function Υ is piecewise linear in c and r. The break points of Υ correspond to hotspots such that: (i) a vertex of $\mathcal{T}$ lies on a side of $\mathcal{H}$, or (ii) a corner of $\mathcal{H}$ lies on an edge of $\mathcal{T}$.*

PROOF. It is easy to see that Υ is piecewise, its pieces depending (only) on which trajectory edges intersect the boundary $\partial\mathcal{H}$ of $\mathcal{H}$, and how they do so. Let E denote the set of edges that intersect $\partial\mathcal{H}$. When a side of $\partial\mathcal{H}$ crosses a trajectory vertex the set E changes, and when a corner of $\mathcal{H}$ crosses a trajectory edge, it changes how an edge $e \in E$ intersects $\partial\mathcal{H}$. Hence, the break points of Υ are of type (i) and (ii). It remains to show that Υ is linear in c and r.

Since $\mathcal{H}$ is convex, the intersection between an edge $e = \overline{uv} \in \mathcal{T}$ and $\mathcal{H}$ is single contiguous line segment, a singleton point, or it is empty. Fix the set of sides of $\mathcal{H}$ intersected by e, let $p_\lambda = (1-\lambda)u + \lambda v$, and let α and β be functions such that $\overline{p_{\alpha(c,r)}p_{\beta(c,r)}} = e \cap \mathcal{H}$. We can then express the *contribution* of e, that is, the length of the part of e that lies inside of $\mathcal{H}$, as $\Upsilon_e(c,r) = \|e \cap \mathcal{H}\| = \|e\|\,(\beta(c,r) - \alpha(c,r))$. It is now an easy exercise to show that α and β are linear functions in c and r. It follows that $\|e \cap \mathcal{H}\|$ is piecewise linear in c and r, and hence Υ is piecewise linear as well. □

A function γ is a *simple* linear function if it is of the form $\gamma(x) = ax + b$, for $a, b \in \mathbb{R}$. Since each piece of Υ is a simple linear function, Υ has a description of constant size that we can evaluate and update in $O(1)$ time. This is still the case when $\mathcal{H}$ is convex and polygonal. However, when $\mathcal{H}$ has a curved boundary the intersection points of a single edge are no longer linear functions of c and r. For example, in case $\mathcal{H}$ is a disk, these intersection points are described by an equation of the form $\sqrt{\gamma(c,r)}$, where γ is a quadratic function in c and r. This means that on a single piece, Υ is the sum of square roots. A description of this function has linear size, and linear time is required to evaluate it. Worse still is that we can no longer analytically compute the roots of its derivative, which is needed for maximization. For this reason, we focus on square hotspots.

3. TOTAL LENGTH

3.1 Fixed Radius

When the radius r of $\mathcal{H}$ is fixed, we can compute a placement that maximizes Υ as follows (see [27] for a very similar approach). Let c be the center of $\mathcal{H}$, and consider the parameter space of c. We compute a subdivision $\mathcal{S}$ of the parameter space such that inside each cell, Υ is a simple linear function. It follows that maxima can occur only at vertices of this subdivision. For each cell C, we compute this function and evaluate it at the vertices of C. We can then just select the maximum over all cells. It remains to bound the number of cells in $\mathcal{S}$.

Each edge of $\mathcal{T}$ yields $O(1)$ line segments in $\mathcal{S}$. Thus $\mathcal{S}$ is the arrangement of $O(n)$ line segments and can be constructed in $O(n^2)$ time. See Fig. 1 for an illustration of $\mathcal{S}$.

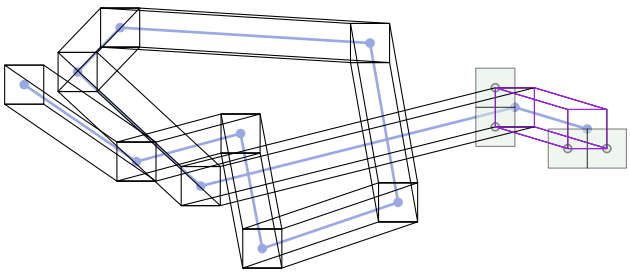

Figure 1: A trajectory and the corresponding subdivision $\mathcal{S}$. The line segments corresponding to a single edge e are drawn in purple.

Let Υ_C denote the function Υ restricted to cell C. In each cell of $\mathcal{S}$ we can compute Υ_C and its maximum from scratch. This takes $O(n)$ time. However, for neighboring cells C and D, Υ_C and Υ_D can differ in only two ways: (i) the set of contributing edges has increased or decreased by one or two edges, or (ii) an edge intersects a different side of $\mathcal{H}$. So, we can compute a function $\Delta_{C,D}$ that describes these changes in constant time. We then have $\Upsilon_D(c) = \Upsilon_C(c) + \Delta_{C,D}(c)$, and thus we can compute Υ_D from Υ_C in constant time (as in [27]). Since $\mathcal{S}$ consists of $O(n^2)$ cells we conclude:

THEOREM 2. *Given a radius r, we can find a hotspot $\mathcal{H}^*$ with radius r that maximizes Υ in $O(n^2)$ time.*

3.2 Fixed Length

Now we are given a threshold L, and we need to find the center c^* and the radius r^* of a minimum size hotspot $\mathcal{H}^*$ with $\Upsilon(c^*, r^*) \geq L$.

The general idea is to use Megiddo's parametric search technique [25]. We briefly sketch this technique here. A more detailed explanation can be found in [1, 25].

Consider a decision algorithm $\mathcal{A}_s$ that, for a given radius r, can determine if there is a center c for which $\Upsilon(c, r) \geq L$ in T_s time. The parametric search technique will run this algorithm with the unknown optimum value r^*. When the technique encounters comparisons involving the radius r^*—in the form of "is this small polynomial equation $p(r^*)$ smaller, greater or equal to zero?"—it computes the roots of p, and runs the decision algorithm $\mathcal{A}_s$ on each of those roots. This gives us a range of radii that must contain r^*. As the technique progresses this range keeps shrinking. In the end, it is either empty, in case the problem was unfeasible, or r^* is the lower endpoint of the interval.

The running time required for this approach depends heavily on how often we have to call the decision algorithm $\mathcal{A}_s$. If we can delay executing a call to $\mathcal{A}_s$ until we have many of them, then we can handle this batch faster by using binary search. That is, we can process m calls to $\mathcal{A}_s$ in $O(T_s \log m)$ time, rather than $O(T_s m)$ time. Since we call $\mathcal{A}_s$ for each comparison involving r^*, this means we need to find batches of independent comparisons that we can process at once. A standard way to achieve this is to replace the (sequential) decision algorithm that we run on r^* by a parallel decision algorithm $\mathcal{A}_p$. If $\mathcal{A}_p$ uses P processors, then executing a single step on each processor yields P independent comparisons in total. We now emulate running $\mathcal{A}_p$ in lockstep. For each step we spend $O(P + T_s \log P)$ time. Thus, if $\mathcal{A}_p$ requires T_p steps, then it takes $O(T_p P + T_p T_s \log P)$ time in total to find r^*.

We can use our algorithm from the previous section as the sequential decision algorithm $\mathcal{A}_s$, by simply computing the maximal length L^* in a hotspot of radius r and checking if $L^* \geq L$. So all that remains is to construct an analogous parallel decision algorithm (which will show the existence of an efficient sequential algorithm to compute r^*).

A Parallel Decision Algorithm. Let r be the given radius. We use the same approach as in the previous section: we construct the subdivision $\mathcal{S}$, and traverse all cells C to compute Υ_C and its maximum. Constructing $\mathcal{S}$ and its dual graph $\mathcal{G}$ takes $O(\log \log^* n)$ time, using $O(n^2 / \log n)$ processors [5]. The graph $\mathcal{G}$ has a node v for every face F_v of $\mathcal{S}$, and an arc (u, v) for each pair of adjacent faces F_u and F_v.

Next, we compute a spanning tree of $\mathcal{G}$ and its Euler tour $\mathcal{E} = \varepsilon_0, .., \varepsilon_m$, with $m = O(n^2)$. This takes $O(\log n)$ time using $O(n^2)$ processors [30]. Now let Υ_i denote the function Υ in node ε_i, and let $\Delta_{i,j}$ be the difference function between Υ_i and Υ_j, that is, $\Delta_{i,j}(c) = \Upsilon_j(c) - \Upsilon_i(c)$. For two consecutive nodes ε_i and ε_{i+1} we can compute this function $\Delta_{i,i+1}$ without having computed Υ_i and Υ_{i+1} since we know which trajectory edge starts/stops to intersect $\mathcal{H}$ (or which edge now intersects a different side of $\mathcal{H}$). Once we have $\Delta_{i,j}$ and $\Delta_{j,\ell}$ we can obtain $\Delta_{i,\ell}$ by combining the two functions. So, the main idea is to compute function Υ_0, and all functions $\Delta_{0,i}$ to obtain the functions Υ_i. We now show that this can be done using $O(n^2)$ processors in $O(\log n)$ time.

LEMMA 3. *Given $O(n)$ processors, we can compute the function Υ for a given node v of $\mathcal{G}$ in $O(\log n)$ time.*

PROOF. We use one processor for each trajectory edge e. Each processor computes the function $\Upsilon_{ev}(c) = \|e \cap \mathcal{H}\|$. We then add these functions together in pairs of two, and repeat this process until we have one function representing Υ. This takes $O(\log n)$ steps. □

LEMMA 4. *Given $O(m)$ processors, with $m \geq n$, and a path $\mathcal{E}$ of length m, we can compute all functions Υ_i in $O(\log m)$ time.*

PROOF. By Lemma 3 we can compute Υ_0 in $O(\log n) = O(\log m)$ time. We now show how to compute all functions $\Delta_{0,i}$ in $O(\log m)$ time in total. We then use one processor for each i to compute Υ_i from Υ_0 and $\Delta_{0,i}$ in constant time.

We represent $\mathcal{E}$ as a balanced binary tree T. Each node $v_{i,j}$ in T represents a subpath $\varepsilon_i, .., \varepsilon_j$. The leaves of T simply correspond to the singleton paths ε_i. We now compute the functions $\Delta_{i,j}$ for each internal node in the tree. Using $O(m)$ processors, starting with one in each leaf, this takes $O(\log m)$ time. For a given leaf ε_i, we can compute $\Delta_{0,i}$ by traversing T from the root to ε_i: we sum the functions stored at the left child of each node v we encounter, and add this to the function stored at the leaf of ε_i. This takes $O(\log m)$ time. Since we have $O(m)$ processors, we can compute all functions $\Delta_{0,i}$ in parallel. □

By Lemma 4 we can compute Υ_i for all nodes in $\mathcal{E}$ in $O(\log m) = O(\log n^2) = O(\log n)$ time using $O(n^2)$ processors. Once we have all functions Υ_i, we can use the same tactic to compute the global maximum in $O(\log n)$ time. Thus, we have:

THEOREM 5. *Given a threshold L, a radius r, and $O(n^2)$ processors, we can decide if there is a hotspot $\mathcal{H}$ with center c and radius r such that $\Upsilon(c, r) \geq L$ in $O(\log n)$ time.*

Computing a Minimum Size Hotspot. We use the above decision algorithm together with the parametric search technique. We stress that this yields a sequential algorithm to compute $\mathcal{H}^*$, since we emulate the execution of the parallel algorithm during the parametric search. In general parametric search takes $O(PT_P + T_P T_S \log P)$ time. From Theorem 2 we have $T_S = O(n^2)$, and from Theorem 5 we have $T_P = O(\log n)$ and $P = O(n^2)$. Plugging in these results gives us:

THEOREM 6. *Given a threshold L, we can find a minimum size hotspot $\mathcal{H}^*$ with center c^* and radius r^* such that $\Upsilon(c^*, r^*) \geq L$ in $O(n^2 \log^2 n)$ time.*

4. CONTIGUOUS LENGTH

In this section we focus on finding a hotspot containing a longest contiguous subtrajectory.

Recall that $\mathcal{T}[s,t]$ is the subtrajectory between s and t. With some abuse of notation we will also write $\mathcal{T}[p,q] = \mathcal{T}[t_p, t_q]$ for the subtrajectory between points $p = \mathcal{T}[t_p]$ and $q = \mathcal{T}[t_q]$.

4.1 Fixed Radius

Let $\mathcal{T}^* = \mathcal{T}[p,q]$ be a longest subtrajectory contained in any hotspot of radius r. We will state and prove properties of $\mathcal{T}^*$ that will help us to compute it efficiently. First we make the simple observation that the starting point p of $\mathcal{T}^*$ will be on the boundary of $\mathcal{H}^*$, with the one exception when $\mathcal{T}^*$ starts at the start of the trajectory $\mathcal{T}$. This exception is easy to handle in $O(n)$ time, so we ignore it from now on and assume that $\mathcal{T}^*$ starts at a point p on $\partial\mathcal{H}^*$.

LEMMA 7. *There is a hotspot $\mathcal{H}^*$ that maximizes Φ for which at least one vertex of $\mathcal{T}$ lies on the boundary of $\mathcal{H}^*$.*

PROOF. Proof by contradiction. Assume that $\mathcal{H}^*$ maximizes Φ, and there is no hotspot $\mathcal{H}$ with $\Phi(\mathcal{H}) \geq \Phi(\mathcal{H}^*)$ with a vertex on $\partial\mathcal{H}$. Since $\mathcal{H}^*$ is optimal, the longest contiguous subtrajectory $\mathcal{T}^* = \mathcal{T}[p,q]$ in $\mathcal{H}^*$ must touch two opposing sides of $\mathcal{H}^*$. Assume without loss of generality that these sides are horizontal.

Let $\mathcal{H}' = \mathcal{H}^*$ and let $\mathcal{T}'$ be the subtrajectory of $\mathcal{T} \cap \mathcal{H}'$ that is (initially) $\mathcal{T}^*$. It is easy to see that we can shift $\mathcal{H}'$ horizontally—while keeping $\mathcal{T}^*$ inside it—until either a vertex lies on $\partial\mathcal{H}'$ or p lies on a corner of $\mathcal{H}'$. In the former case we immediately obtain a contradiction. In the latter case, translate $\mathcal{H}'$ while keeping the starting point p' of $\mathcal{T}'$ on the same corner of $\mathcal{H}'$ (moving the starting point of $\mathcal{T}'$ earlier or later). Let $\phi(t)$ denote the length of $\mathcal{T}'$ as a function of the starting time $t = t_{p'}$ of $\mathcal{T}'$. Function ϕ has break points when p' or q' crosses a vertex or when $\mathcal{H}'$ gets a vertex of $\mathcal{T}'$ on its boundary. Since ϕ is (piecewise) linear, the translation of $\mathcal{H}'$ in at least one direction does not decrease the length of $\mathcal{T}'$ until it gets a vertex on its boundary, a contradiction. This completes the proof. □

COROLLARY 8. *For starting point $p = (p_x, p_y)$ of $\mathcal{T}^*$, and some vertex $v = (v_x, v_y)$ of $\mathcal{T}$ we have that (i) $p_x \in \{v_x - r, v_x, v_x + r\}$ or $p_y \in \{v_y - r, v_y, v_y + r\}$, and (ii) $v \in \mathcal{T}^*$.*

We now examine all vertices and all six cases for these vertices, knowing by Corollary 8 that one of these will give us p, the starting point of $\mathcal{T}^*$ with the property that the corresponding $\mathcal{H}^*$ has a vertex of $\mathcal{T}$ on its boundary.

Let v be some vertex of $\mathcal{T}$ and assume we are testing the case where $p_y = v_y - r$; all other cases are symmetric or handled analogously. In this case, the unknown point p lies on the bottom side of $\mathcal{H}^*$ and v lies on the top side. Furthermore, t_p is earlier than t_v, and t_p is the last point before v with y-coordinate $v_y - r$. Assume we have the last point before v with y-coordinate $v_y - r$ and call it p'. If $[t_{p'}, t_v]$ contains any point with y-coordinate greater than v_y, then p' is not a candidate for the start p of the optimal subtrajectory $\mathcal{T}^*$, otherwise we proceed to compute a maximal time interval I starting at p' such that I has its y-coordinates in the range $[v_y - r, v_y]$. Suppose that we have $I = [p', q']$. Then we test whether I satisfies the condition that the x-extent of $\mathcal{T}[p', q']$ is at most r, and in that case $\mathcal{T}[p', q']$ is a candidate to be $\mathcal{T}^*$. After testing all vertices v and all six cases, we choose the longest one as $\mathcal{T}^*$.

We need to find an efficient way to implement the query for p', the query for q', and the test whether $\mathcal{T}[p', q']$ has an x-extent of at most r. Suppose we define a polygonal line whose horizontal axis is time and its vertical axis is the y-coordinate, see Fig. 2. Assume that we have a horizontal decomposition of this polyline, preprocessed for efficient planar point location. Effectively this means that we can perform horizontal ray shooting queries in $O(\log n)$ time. A horizontal ray shooting query to the left from a point $(\hat{t}, \hat{y})$ asks for the most recent time before $\hat{t}$ where the trajectory has y-coordinate $\hat{y}$. Similarly, a ray shooting query to the right asks for the earliest time after $\hat{t}$ where the trajectory has y-coordinate $\hat{y}$. Hence, to determine p' given v, we perform a ray shooting query to the left from the point $(t_v, v_y - r)$, giving us $(t_{p'}, p'_y)$ and therefore p'. Then we perform a ray shooting query to the right from $(t_{p'}, p'_y + r) = (t_{p'}, v_y)$. If the ray hits an edge before t_v then p' is not a candidate start point and we stop. If the ray hits v and $\mathcal{T}$ extends upwards after v, then $[t_{p'}, t_v]$ is a candidate time interval that could give $\mathcal{T}^*$. Otherwise, v is a local maximum of the y-coordinate and we perform two ray shooting queries to the right: from (t_v, v_y) and from $(t_v, v_y - r)$. The earliest time of the two answers gives us q' and thus the interval I to test.

Finally, we test whether the x-extent of I is at most r. This can be done with an augmented binary search tree that stores the edges of $\mathcal{T}$ ordered along $\mathcal{T}$ in its leaves. Every internal node is augmented with two values: the minimum and maximum occurring x-coordinate of the edges in the leaves below it. Since this is a standard application of tree augmentation, we describe it no further and note that interval I can be queried in $O(\log n)$ time for its x-extent [13].

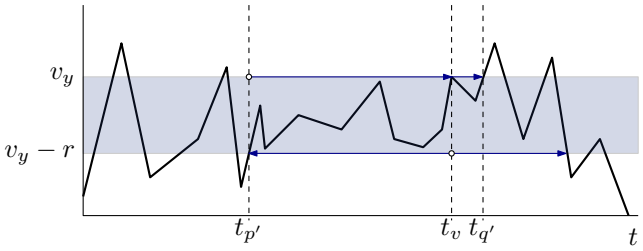

Figure 2: Determining the earliest starting time $t_{p'}$, and the latest ending time $t_{q'}$ such that $\mathcal{T}[p', q']$ lies in the slab $[v_y - r, v_y]$.

Running Time Analysis. In the preprocessing we build the ray shooting structures (for (t,y) and (t,x)) and augmented search trees (for x and for y) in $O(n \log n)$ time. Then for every vertex v, we can find candidate intervals in $O(\log n)$ time. We conclude:

THEOREM 9. *Given a radius r, we can find a hotspot $\mathcal{H}^*$ with radius r that maximizes Φ in $O(n \log n)$ time.*

4.2 Fixed Length

We are given a threshold L on the minimum required trajectory length in the hotspot, and we want to find a smallest hotspot $\mathcal{H}^*$ that contains a subtrajectory of length L.

Let $\phi(t)$ be the minimum radius of a hotspot $\mathcal{H}$ containing a subtrajectory $\mathcal{T}[p,q]$ of length L that enters $\mathcal{H}$, and thus

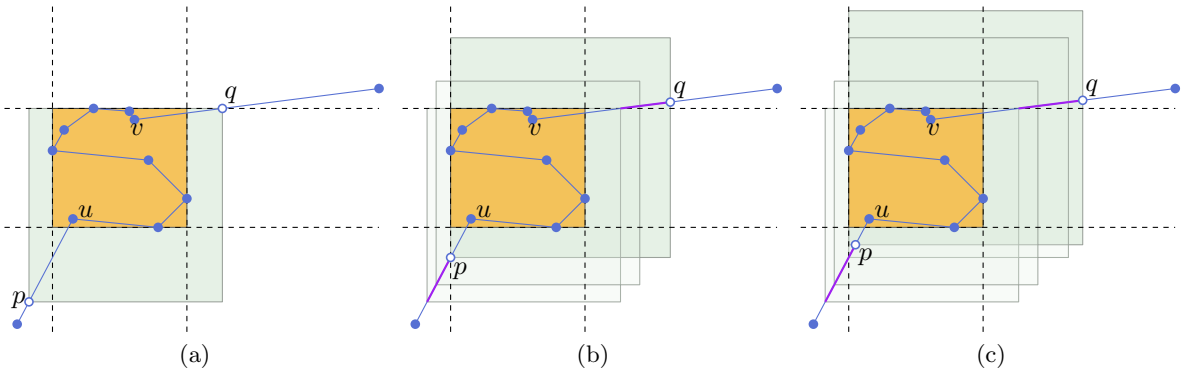

Figure 3: The smallest hotspot for several different starting times. The offset in length Δ is indicated in purple, and $\mathcal{BB}(u, v)$ in orange. (a) and (b) correspond to break points of a piece i of ϕ. (c) is a hotspot on piece $i + 1$.

starts, at time $t = t_p$, and let u and v be the first and the last internal vertex in $\mathcal{T}[p, q]$, respectively. We denote the bounding box of a subtrajectory $\mathcal{T}[a, b]$ by $\mathcal{BB}(a, b)$. We now prove:

LEMMA 10. *The function ϕ is piecewise linear. The $O(n)$ break points of ϕ correspond to hotspots $\mathcal{H}$ such that: (i) p is a vertex of $\mathcal{T}$, (ii) q is a vertex of $\mathcal{T}$, (iii) p_x (p_y) coincides with the minimum or maximum x-coordinate (y-coordinate) of $\mathcal{BB}(u, v)$, (iv) q_x (q_y) coincides with the minimum or maximum x-coordinate (y-coordinate) of $\mathcal{BB}(u, v)$, and (v) $p_x = q_x$ or $p_y = q_y$.*

PROOF. We start by showing that ϕ is linear. Let e be the edge containing p and f the edge containing q. If we move point p along e by some small amount Δ, the length inside the hotspot decreases linearly in Δ. To maintain length L inside $\mathcal{H}$, point q has to move along f (see Fig. 3). The required increase in length on f is identical to the decrease in length on edge e, namely Δ. It follows that p and q both move linearly in Δ. Therefore, ϕ is linear as well.

Clearly, ϕ is piecewise. Next, we show that the break points of ϕ are of types (i) to (v). At any time t, a hotspot $\mathcal{H}$ with radius $\phi(t)$ has two opposite sides, s_1 and s_2, that both contain an internal vertex or an end point of $\mathcal{T}[p, q]$. As time t varies, s_1 and s_2 move. Function ϕ has a break point if and only if the movement of s_1 and s_2 changes. This movement changes when the movement of p and q changes, and when the objects defining s_1 and s_2 change (e.g. s_1 was defined by p, but is now defined by an internal vertex of $\mathcal{T}[p, q]$). The former changes occur when p and q are at trajectory vertices, thus yielding break points of type (i) and (ii). The latter changes occur exactly at events of type (iii) to (v), thus yielding break points of type (iii) to (v).

Finally, we argue that there are $O(n)$ break points. Clearly, there are $O(n)$ break points of type (i) and (ii). It follows that there are also only $O(n)$ pairs of edges (e, f) such that p lies on e and q on f. For each such a pair there are at most $O(1)$ break points of type (v). Point p (q) encounters at most $O(1)$ events of type (iii) (type (iv)) per edge. So the number of break points of these types is $O(n)$ as well. □

Note that ϕ is not continuous. In particular, ϕ is not defined for times t such that $\mathcal{T}[t]$ lies in the interior of $\mathcal{BB}(u, v)$, where u and v are the first and last interior vertices in subtrajectory of length L starting at time t.

Since ϕ is piecewise linear, its minimum occurs at a break point. So, to find a smallest hotspot containing length L, we compute all break points of ϕ and evaluate ϕ at each of them.

We can easily find all break points of ϕ by "sweeping" $\mathcal{T}$ with a subtrajectory $\mathcal{T}[p, q]$ of length L. To quickly find the bounding box and the length of a subtrajectory we represent $\mathcal{T}$ as a balanced binary search tree. Each leaf node represents a trajectory edge e, and stores e, its bounding box, and its length. An internal node $\nu_{a,b}$ represents the subtrajectory $\mathcal{T}[a, b]$ from vertex a to vertex b, and stores $\mathcal{BB}(a, b)$ and the length of $\mathcal{T}[a, b]$. Building this tree takes $O(n \log n)$ time, and allows $O(\log n)$ time queries. Once we have the bounding box and the length of $\mathcal{T}[u, v]$, we can construct and evaluate ϕ in constant time. We have $O(n)$ events in total, each of which we can handle in $O(\log n)$ time. Hence, we can find the global minimum in $O(n \log n)$ time. Thus:

THEOREM 11. *Given a threshold L, we can find a minimum size hotspot $\mathcal{H}^*$ such that $\Phi(\mathcal{H}^*) \geq L$ in $O(n \log n)$ time.*

5. RELATIVE LENGTH

We now focus on finding a hotspot $\mathcal{H}^*$ with center c^* and radius r^* that maximizes the relative trajectory length $\Psi(\mathcal{H}^*) = \Psi(c^*, r^*) = \Upsilon(c^*, r^*)/2r^*$.

Given a hotspot $\mathcal{H}$, a point $p \in \mathcal{H}$, and a radius r. Let $\mathcal{H}_p^r$ be the hotspot $\mathcal{H}$, scaled with p as origin and such that its radius is r. Fix a point p, and consider Ψ as a function of r. More formally, let $\psi_p(r) = \Psi(\mathcal{H}_p^r)$.

LEMMA 12. *ψ_p is a piecewise hyperbolic function. The pieces of ψ_p are of the form $c(1/r) + d$, for $c, d \in \mathbb{R}$, and the break points of ψ_p correspond to hotspots $\mathcal{H}$ such that: (i) a vertex of $\mathcal{T}$ lies on a side of $\mathcal{H}$, or (ii) a corner of $\mathcal{H}$ lies on an edge of $\mathcal{T}$.*

PROOF. Since Υ is a piecewise function, so are Ψ and ψ_p. It is easy to see that the break points of ψ_p are the same as those of Υ. We now show that each piece of ψ_p is of the form $c(1/r) + d$, with $c, d \in \mathbb{R}$. Consider a piece of ψ_p, and let E be the set of contributing edges on that piece. Let $A \subseteq E$ be the set of edges that are completely contained in any hotspot corresponding to this piece, and let $B = E \setminus A$ be the set of remaining edges. For each edge e in B the

length in $\mathcal{H}_p^r$ changes linearly in r. Let $\lambda_e(r)$ denote this length. We then have that

$$\psi_p(r) = \frac{\Upsilon(\mathcal{H}_p^r)}{2r} = \frac{\sum_{e\in A}\|e\|}{2r} + \frac{\sum_{e\in B}\lambda_e(r)}{2r} = a\frac{1}{r} + \frac{\hat{b}r+b}{r}$$
$$= (a+b)\frac{1}{r} + \hat{b},$$

where a, b, and $\hat{b}$ are constants. The lemma follows. □

Let $\mathbb{V}(\mathcal{H})$ denote the set of sides of $\mathcal{H}$ containing a vertex, and let $\mathbb{v}(\mathcal{H}) = |\mathbb{V}(\mathcal{H})|$. Similarly, let $\mathbb{E}(\mathcal{H})$ denote the set of corners of $\mathcal{H}$ that lie on a trajectory edge, and let $\mathbb{e}(\mathcal{H}) = |\mathbb{E}(\mathcal{H})|$.

Lemma 13. *There is a hotspot $\mathcal{H}^*$ that maximizes Ψ such that $\mathbb{v}(\mathcal{H}^*) + \mathbb{e}(\mathcal{H}^*) \geq 3$.*

Proof. Proof by contradiction. Assume that $\mathcal{H}^*$ is a hotspot that maximizes Ψ, with $\mathbb{v}(\mathcal{H}^*) + \mathbb{e}(\mathcal{H}^*) < 3$, and that there is no hotspot $\mathcal{H}$ with $\Psi(\mathcal{H}) \geq \Psi(\mathcal{H}^*)$ and $\mathbb{v}(\mathcal{H}) + \mathbb{e}(\mathcal{H}) > \mathbb{v}(\mathcal{H}^*) + \mathbb{e}(\mathcal{H}^*)$.

We now show that we can scale or translate $\mathcal{H}^*$ without decreasing Ψ and while keeping $\mathbb{V}(\mathcal{H}^*)$ and $\mathbb{E}(\mathcal{H}^*)$ the same until (i) there is a new vertex on a (new) side of $\mathcal{H}^*$ or, (ii) there is an new edge through a new corner of $\mathcal{H}^*$. This leads to a contradiction, and thus proves the lemma.

Let $\mathcal{H}' = \mathcal{H}^*$, and consider the relative length $\psi(a) = \Psi(\mathcal{H}')$ in $\mathcal{H}'$ as a function of some parameter a. We choose a to be a translation if there are two opposing sides in $\mathbb{V}(\mathcal{H}^*)$, and a scaling otherwise. In both cases we will show that (1) ψ is a piecewise function, (2) $\mathcal{H}^*$ corresponds to an interior value of a piece p of ψ, (3) one of the endpoints a' of p has $\psi(a') \geq \Psi(\mathcal{H}^*)$, and (4) the break points of ψ correspond to hotspots $\mathcal{H}'$ such that (i) there is a vertex on a side of $\mathcal{H}'$ or, (ii) there is an edge through a corner of $\mathcal{H}'$. It follows that the hotspot $\mathcal{H}'$ corresponding to a' has $\Psi(\mathcal{H}') \geq \Psi(\mathcal{H}^*)$ and $\mathbb{v}(\mathcal{H}') + \mathbb{e}(\mathcal{H}') > \mathbb{v}(\mathcal{H}^*) + \mathbb{e}(\mathcal{H}^*)$, as desired.

Consider the case in which $\mathbb{V}(\mathcal{H}^*)$ contains two opposing sides s_1 and s_2. Assume without loss of generality that s_1 and s_2 are horizontal. We now choose a to be the x-coordinate of the center of $\mathcal{H}'$. We leave the radius of $\mathcal{H}'$ fixed, so ψ is a piecewise linear function in a. This proves (1) and (3). The break points of ψ correspond to hotspots such that a vertex of $\mathcal{T}$ lies on a vertical side of $\mathcal{H}'$ or a trajectory edge intersects a corner of $\mathcal{H}'$. This proves (4). Item (2) follows since $\mathbb{v}(\mathcal{H}^*) + \mathbb{e}(\mathcal{H}^*) < 3$.

Consider the case in which $\mathbb{V}(\mathcal{H}^*)$ does not contain two opposing sides. Since $\mathbb{v}(\mathcal{H}^*) + \mathbb{e}(\mathcal{H}^*) < 3$, there is a point $q \in \mathcal{H}^*$ such that if we scale $\mathcal{H}' = \mathcal{H}^*$ by a small amount with q as origin, the vertices on $\partial\mathcal{H}'$ stay on the same side as in $\mathcal{H}^*$, and the edges through corners of $\mathcal{H}'$ go through the same corners in $\mathcal{H}^*$. Let a be the radius of $\mathcal{H}'$ during this scaling. Thus, $\psi(a) = \psi_q(a)$. Items (1), (3) and (4) now directly follow from Lemma 12. Item (2) again holds since $\mathbb{v}(\mathcal{H}^*) + \mathbb{e}(\mathcal{H}^*) < 3$. □

When a corner c of $\mathcal{H}$ lies in the interior of an edge $e = \overline{uv}$, that is, $c \neq u \neq v$, and $e \cap \mathcal{H} = c$, e *touches* $\mathcal{H}$, see Fig. 4. Let $\mathbb{E}_{nt}(\mathcal{H})$ denote the set of corners of $\mathcal{H}$ that lie on trajectory edges that do not touch $\mathcal{H}$. That is, for each corner c in $\mathbb{E}_{nt}(\mathcal{H})$, the edge through c does not touch $\mathcal{H}$. Let $\mathbb{e}_{nt}(\mathcal{H}) = |\mathbb{E}_{nt}(\mathcal{H})|$.

Lemma 14. *There is a hotspot $\mathcal{H}^*$ that maximizes Ψ such that $\mathbb{v}(\mathcal{H}^*) + \mathbb{e}_{nt}(\mathcal{H}^*) \geq 3$.*

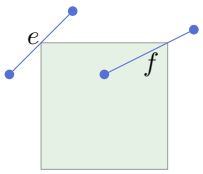

Figure 4: Edge e touches the hotspot, edge f does not.

Proof. It follows from Lemma 13 that there is a hotspot $\mathcal{H}^*$ maximizing Ψ, with $\mathbb{v}(\mathcal{H}^*) + \mathbb{e}(\mathcal{H}^*) \geq 3$. We now show by contradiction that $\mathbb{v}(\mathcal{H}^*) + \mathbb{e}_{nt}(\mathcal{H}^*) \geq 3$.

Let r^* be the radius of $\mathcal{H}^*$, and let e be an edge that touches $\mathcal{H}^*$ in corner c. Hence, $c \in \mathbb{E}_{nt}(\mathcal{H}^*)$. If no such edge, and thus no such corner, exists then we immediately arrive at a contradiction. If e does exist, we will show that this contradicts the fact that $\mathcal{H}^*$ maximizes Ψ.

There is a point p such that we can scale $\mathcal{H}^*$ while maintaining a total of two objects $o_1 \neq c$ and $o_2 \neq c$ in $\mathbb{V}(\mathcal{H}^*)$ and $\mathbb{E}(\mathcal{H}^*)$. Now consider the piecewise hyperbolic function ψ_p (Lemma 12). Since $\mathcal{H}^*$ is optimal, the value r^* is a break point of ψ_p. Let $[\check{r}, r^*]$ and $[r^*, \hat{r}]$ be the pieces incident to r^*, and let $\check{\psi}$ and $\hat{\psi}$ denote the function ψ_p restricted to their corresponding pieces.

Since $\mathcal{H}^*$ maximizes Ψ we have $\check{\psi}(r) = -a\frac{1}{r} + b$, and $\hat{\psi}(r) = c\frac{1}{r} + d$, for some $a, c \geq 0$ and $b, d \in \mathbb{R}$. We now consider the length of edge e in the hotspot as a function of r on the interval $[r^*, \hat{r}]$. Since this length $\lambda_e(r)$ is positive, we have $\lambda_e(r) = gr - h$, for some $g, h \geq 0$. This gives us

$$\hat{\psi}(r) = \check{\psi}(r) + \frac{\lambda_e(r)}{2r} = \check{\psi}(r) - \frac{h}{2r} + \frac{g}{2} = (-a - \frac{h}{2})\frac{1}{r} + b + \frac{g}{2}.$$

Hence $c = (-a - h/2) < 0$. Contradiction. □

An Algorithm to Maximize Ψ. By Lemma 14 there are three objects, vertices or edges, bounding an optimal hotspot. Hence, there is a hotspot maximizing Ψ such that

- $\partial\mathcal{H}^*$ contains three vertices on three different sides,
- $\partial\mathcal{H}^*$ contains two vertices on different sides and one edge intersects a corner of $\mathcal{H}^*$,
- a vertex lies in a corner of $\mathcal{H}^*$, and there is either a second vertex on $\partial\mathcal{H}^*$ or an edge going through a different corner of $\mathcal{H}^*$,
- $\partial\mathcal{H}^*$ contains one vertex and two edges intersects corners of $\mathcal{H}^*$, or
- three edges intersect corners of $\mathcal{H}^*$.

In all cases the edges do not touch $\mathcal{H}^*$.

We now compute a hotspot maximizing Ψ in each of these cases, and then simply pick a best one. In each case, our global approach is to fix two of the three objects. This fixes two of the three degrees of freedom. We then express Υ as a function of the remaining degree of freedom a. This function Υ is piecewise linear in a, and will have events/break points when a third object bounds the hotspot $\mathcal{H}$. Thus, we can find an optimal solution by evaluating Υ and Ψ at each of the break points.

How we find all break points differs per case, but we will show that in each case we can find the $O(n)$ break points in linear time. Computing and maintaining Υ also takes linear time in total, since each update can be handled by adding and/or subtracting a simple linear function that describes

the change at that break point. There are $O(n^2)$ pairs of objects that we can fix, so we can handle each case in $O(n^3)$ time in total.

We use the following simple data structures throughout the different cases. We construct two lists $\mathcal{L}_x$ and $\mathcal{L}_y$ of all vertices, $\mathcal{L}_x$ sorted on increasing x-coordinate, and $\mathcal{L}_y$ sorted on increasing y-coordinate. Furthermore, we explicitly build the arrangement $\mathcal{A}$ on the supporting lines of the edges of $\mathcal{T}$. With this arrangement we can now answer the following queries in $O(n)$ time. Given a query (half-)line, or *ray*, ℓ find all trajectory edges intersected by ℓ in the order in which they are intersected. We do this by simply walking along ℓ in the arrangement $\mathcal{A}$. Since the zone of ℓ in $\mathcal{A}$ has linear complexity, such a query takes $O(n)$ time.

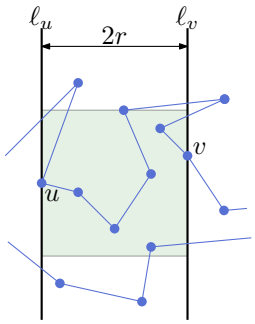

Figure 5: The three vertices case.

Three Vertices. There is an optimal hotspot such that three sides contain a vertex. Two of these sides must be parallel. Assume that these two sides are contained by the vertical lines ℓ_u and ℓ_v (the case that ℓ_u and ℓ_v are horizontal is handled analogously), and that these lines are at distance $2r$ from each other (see Fig. 5). These two vertical lines bound a vertical slab, we now place a square hotspot $\mathcal{H}$ with radius r at the bottom of this slab, and shift it upwards. Let a be the the y-coordinate of the top of $\mathcal{H}$, and consider Υ as a function of a. Each time a side of $\mathcal{H}$ hits a vertex, or a corner of $\mathcal{H}$ hits a trajectory edge, we get a break point.

Two Vertices, One Edge. Let u and v be the vertices on $\partial\mathcal{H}^*$. When u and v lie on opposing sides of $\mathcal{H}^*$ we can use the same approach as in the previous case. When u and v lie on neighboring sides we use the following approach. Assume without loss of generality that u lies on the bottom side of $\mathcal{H}^*$ and v on the left side of $\mathcal{H}^*$. This means that the bottom left corner o of $\mathcal{H}^*$ is uniquely defined by u and v.

We start with an arbitrarily small empty hotspot $\mathcal{H}$ with its bottom left corner at o. We now scale $\mathcal{H}$ with o as origin. Let a denote the scaling parameter, and consider Υ as a function of a. The function Υ combinatorially changes when any side of $\mathcal{H}$ hits a vertex, or any corner of $\mathcal{H}$ hits a trajectory edge. We find these times by querying $\mathcal{A}$ with a horizontal ray, a vertical ray, and a diagonal ray starting at o, and merging their results with $\mathcal{L}_x$ and $\mathcal{L}_y$ (see Fig. 6).

One Corner Vertex. Let v be the vertex of $\mathcal{T}$ on a corner of $\mathcal{H}^*$. We define $o = v$, and then handle this case analogous to the previous case.

One Vertex, Two Edges. Let v be the vertex on $\partial\mathcal{H}^*$, and let e be the edge through a corner of $\mathcal{H}^*$. Assume without loss of generality that v lies on the bottom side of $\mathcal{H}^*$. We

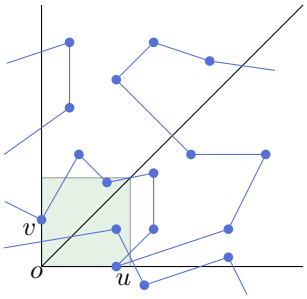

Figure 6: The two vertices, one edge case.

distinguish two subcases: e intersects a bottom corner of $\mathcal{H}^*$, or e intersects a top corner of $\mathcal{H}^*$.

In the first case, the horizontal line through v intersects e in a point o. We again consider scaling $\mathcal{H}$ with origin o, so this case is handled analogously to the case where there were two vertices on $\partial\mathcal{H}^*$.

In the second case, e intersects $\mathcal{H}$ in a top corner c_1. Let c_2 be the other top corner. All hotspots that have corner c_1 on e, and u on the bottom side, have their other upper corner, c_2, on a line m (see Fig. 7). We consider these hotspots and the function Υ by increasing size a. We find the break points as follows. To find all edges that could intersect $\mathcal{H}$ in corner c_2, we query $\mathcal{A}$ with m, oriented such that the hotspots $\mathcal{H}$ get larger along m. We find the intersections with other corners, ordered by increasing size a, by querying $\mathcal{A}$ with two horizontal half-lines starting at v. We then merge these three lists with the sorted lists of vertices to get a list of all break points.

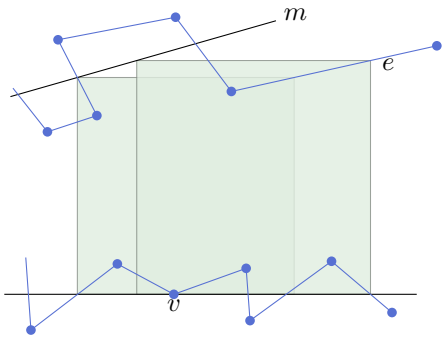

Figure 7: The one vertex, two edges case.

Three Edges. Let e, f, and g be the edges through corners of $\mathcal{H}^*$, of which e and f are through opposing corners c_e and c_f. Let ℓ_e and ℓ_f be the lines containing e and f, respectively (see Fig. 8). Consider all hotspots that have e through corner c_e and f through c_f. The remaining two corners of these hotspots lie on half-lines m_1 and m_2, starting at the intersection point p of ℓ_e and ℓ_f.

We now consider Υ as a function of the size a of the hotspots that have e through corner c_e and f through corner c_f. To compute the break points of Υ, we query $\mathcal{A}$ with m_1, m_2, ℓ_e and ℓ_f. We merge these lists with the sorted lists of vertices to get all break points, ordered on increasing size a.

We now conclude:

THEOREM 15. *We can find a minimum size hotspot $\mathcal{H}^*$ that maximizes Ψ in $O(n^3)$ time.*

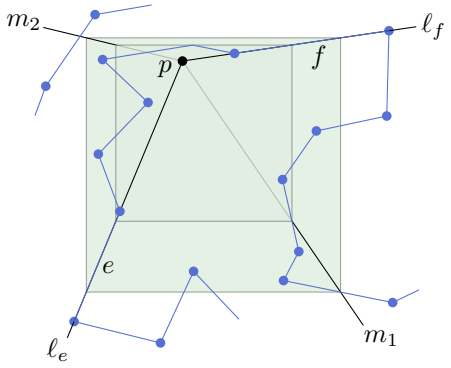

Figure 8: The three edges case.

6. EXTENSIONS

In this section we briefly discuss various extensions to our algorithms, in particular, multiple entities and differently shaped hotspots.

Multiple Entities. Suppose that instead of one moving entity with one trajectory $\mathcal{T}$, we have many entities, and thus many trajectories $\mathcal{T}_1, .., .\mathcal{T}_m$. Can we still find a hotspot $\mathcal{H}$ that maximizes the length inside it, or minimizes the size required to get at least a certain length? Our algorithms simply treat trajectory $\mathcal{T}$ as a set of line segments. So they are immediately applicable to multiple trajectories as well.

Convex Polygonal Hotspots of a Given Shape. The algorithms that use the total length (the ones from Section 3), are still applicable when the hotspot is a convex polygon of a given shape. If the polygon has k vertices, each trajectory edge produces $O(k)$ line segments in subdivision $\mathcal{S}$. Thus, $\mathcal{S}$ may have a total complexity of $O(k^2n^2)$. We then solve the fixed radius version (for an appropriate definition of radius) in $O(k^2n^2)$ time. Similarly, the fixed length version takes $O(k^2n^2 \log^2(kn))$ time.

If we consider the relative length inside $\mathcal{H}$, Lemmas 13 and 14 still hold, even if $\mathcal{H}$ is a convex polygon of complexity k of a given shape. This means that there are still three objects "bounding" $\mathcal{H}^*$; one for each parameter specifying the position of $\mathcal{H}^*$. So, the global approach of our algorithm is still applicable. Which, and how many, ray shooting queries we have to perform to obtain the break points depends on the shape of the hotspot.

For the algorithms that use the contiguous length/time it is not immediately clear how to extend them to work for arbitrarily, but fixed, shaped polygons. We can extend the solution for fixed r, in case $\mathcal{H}$ is a regular k-gon.

Other Types of Hotspots. When the hotspot has curved boundaries, we can no longer maximize Υ, Φ, and Ψ, as described in Section 2. Hence, our algorithms do not easily extend to these cases. When the hotspot $\mathcal{H}$ is not convex, the intersection of a single edge with $\mathcal{H}$ may consist of several line segments. This will increase the time required to evaluate Υ, Φ, and Ψ. Furthermore, this may lead to a large increase of complexity in the parameter space. When the shape of the hotspot is not predefined it is not clear how define the problem as a maximization problem. Hence, in this case we cannot directly apply our algorithms either.

7. CONCLUDING REMARKS

Hotspots are small regions where a moving entity spends a significant amount of time. We presented algorithms to locate optimal hotspots from trajectory data based on path length rather than time, but versions based on time require essentially the same algorithms and the same efficiency is obtained. Five variations of the problem were considered. When all visits to the hotspot count our algorithms take roughly quadratic time: $O(n^2)$ time if we want to maximize the time in a fixed size square hotspot, and $O(n^2 \log^2 n)$ time if we want to minimize the size of the hotspot for a fixed time the entity spends inside. If we are interested only in the longest contiguous visit, we can solve both variations in $O(n \log n)$ time. Maximizing the relative time inside, compared to the size of the hotspot, takes $O(n^3)$ time. Our algorithms can find multiple hotspots by removing the edges and edge parts that lie inside the hotspot found, and repeating the algorithm on the remaining edges, although for the two contiguous versions this may not give the desired results.

There are various heuristic extensions possible to let the algorithm with cubic runtime be much faster in practice. For all algorithms we can apply trajectory simplification to improve the efficiency if needed, because our methods can handle irregularly sampled data without problems (unlike point-based methods).

Our algorithms directly extend to multiple entities. However, when multiple entities are considered several other variations of the problem exist. For example, find a smallest hotspot $\mathcal{H}$ such that all entities spend at least L time in $\mathcal{H}$. We can again consider the total time, the longest contiguous time, and we can even vary whether or not the entities need to be present at the same time(s). We can also consider only the longest visit to the hotspot for each entity, and try to maximize the sum of those durations over all entities. All these variations are interesting options for future work.

8. ACKNOWLEDGMENTS

F.S. was supported by the Netherlands Organisation for Scientific Research (NWO) under project no. 612.001.022, and by EU Cost Action IC0903 (MOVE). J.G. was funded by the Australian Research Council FT100100755.

References

[1] P. Agarwal, M. Sharir, and S. Toledo. Applications of parametric searching in geometric optimization. *J. of Algorithms*, 17(3):292–318, 1994.

[2] H. Alt, B. Behrends, and J. Blömer. Approximate matching of polygonal shapes. *Ann. Math. Artif. Intell.*, 13(3-4):251–265, 1995.

[3] H. Alt and M. Godau. Computing the Fréchet distance between two polygonal curves. *Int. J. Comput. Geom. Appl.*, 5:75–91, 1995.

[4] L. O. Alvares, V. Bogorny, B. Kuijpers, J. de Macêdo, B. Moelans, and A. Vaisman. A model for enriching trajectories with semantic geographical information. In *Proc. 15th ACM Int. Sympos. on Geographic Information Systems*, page 22. ACM, 2007.

[5] R. Anderson, P. Beanie, and E. Brisson. Parallel algorithms for arrangements. *Algorithmica*, 15(2):104–125, 1996.

[6] B. Aronov, A. Driemel, M. van Kreveld, M. Löffler, and F. Staals. Segmentation of trajectories for non-monotone criteria. In *Proc. 24th Annual ACM-SIAM Sympos. on Discrete Algorithms*, pages 1897–1911, 2013.

[7] M. Benkert, B. Djordjevic, J. Gudmundsson, and T. Wolle. Finding popular places. *Int. J. Comput. Geometry Appl.*, 20(1):19–42, 2010.

[8] M. Benkert, J. Gudmundsson, F. Hübner, and T. Wolle. Reporting flock patterns. *Comput. Geom.*, 41:111–125, 2008.

[9] K. Buchin, M. Buchin, and J. Gudmundsson. Detecting single file movement. In *Proc. 16th ACM Int. Conf. on Advances in Geographic Information Systems*, pages 288–297, 2008.

[10] K. Buchin, M. Buchin, J. Gudmundsson, M. Löffler, and J. Luo. Detecting commuting patterns by clustering subtrajectories. *Int. J. Comput. Geom. Appl.*, 21(3):253–282, 2011.

[11] K. Buchin, M. Buchin, M. van Kreveld, and J. Luo. Finding long and similar parts of trajectories. *Comput. Geom.*, 44(9):465–476, 2011.

[12] M. Buchin, A. Driemel, M. van Kreveld, and V. Sacristan. An algorithmic framework for segmenting trajectories based on spatio-temporal criteria. In *Proc. 18th ACM Int. Sympos. on Advances in Geographic Information Systems*, pages 202–211, 2010.

[13] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. MIT Press, 2001.

[14] S. Dodge, R. Weibel, and A.-K. Lautenschütz. Towards a taxonomy of movement patterns. *Information Visualization*, 7(3-4):240–252, 2008.

[15] P. Fauchald and T. Tveraa. Using first-passage time in the analysis of area-restricted search and habitat selection. *Ecology*, 84(2):282–288, 2003.

[16] S. Gaffney and P. Smyth. Trajectory clustering with mixtures of regression models. In *Proc. 5th Int. Conf. on Data Discovery and Data Mining*, pages 63–72, 1999.

[17] J. Gudmundsson, P. Laube, and T. Wolle. Movement patterns in spatio-temporal data. In S. Shekhar and H. Xiong, editors, *Encyclopedia of GIS*, pages 726–732. Springer, 2008.

[18] J. Gudmundsson, M. van Kreveld, and B. Speckmann. Efficient detection of patterns in 2D trajectories of moving points. *GeoInformatica*, 11:195–215, 2007.

[19] A. Jain, M. N. Murty, and P. Flynn. Data clustering: A review. *ACM Comput. Surv.*, 31(3):264–323, 1999.

[20] H. Jeung, M. Yiu, and C. Jensen. Trajectory pattern mining. In Y. Zheng and X. Zhou, editors, *Computing with Spatial Trajectories*, pages 143–177. Springer, 2011.

[21] A. Johnson, J. Wiens, B. Milne, and T. Crist. Animal movements and population dynamics in heterogeneous landscapes. *Landscape Ecology*, 7:63–75, 1992.

[22] E. J. Keogh. Exact indexing of dynamic time warping. In *Proc. 28th Int. Conf. on Very Large Data Bases*, pages 406–417, 2002.

[23] P. Laube, M. van Kreveld, and S. Imfeld. Finding REMO – detecting relative motion patterns in geospatial lifelines. In P. Fisher, editor, *Developments in Spatial Data Handling: 11th Int. Sympos. on Spatial Data Handling*, pages 201–215, 2004.

[24] J. Lee, J. Han, and K.-Y. Whang. Trajectory clustering: a partition-and-group framework. In *Proc. ACM Int. Conf. on Management of Data*, pages 593–604, 2007.

[25] N. Megiddo. Applying parallel computation algorithms in the design of serial algorithms. *J. ACM*, 30(4):852–865, 1983.

[26] B. Moreno, V. Times, C. Renso, and V. Bogorny. Looking inside the stops of trajectories of moving objects. In *Proc. XI Brazilian Sympos. on Geoinformatics*, pages 9–20. MCT/INPE, 2010.

[27] D. Mount, R. Silverman, and A. Wu. On the area of overlap of translated polygons. *Computer Vision and Image Understanding*, 64(1):53–61, 1996.

[28] M. Nanni and D. Pedreschi. Time-focused clustering of trajectories of moving objects. *J. of Intelligent Information Systems*, 27:285–289, 2006.

[29] A. Palma, V. Bogorny, B. Kuijpers, and L. O. Alvares. A clustering-based approach for discovering interesting places in trajectories. In *Proc. 2008 ACM Sympos. on Applied Computing*, pages 863–868, 2008.

[30] Y. Shiloach and U. Vishkin. An $O(\log n)$ parallel connectivity algorithm. *J. of Algorithms*, 3(1):57–67, 1982.

[31] S. Spaccapietra, C. Parent, M. Damiani, J. de Macêdo, F. Porto, and C. Vangenot. A conceptual view on trajectories. *Data Knowl. Eng.*, 65(1):126–146, 2008.

[32] S. Tiwari and S. Kaushik. Mining popular places in a geo-spatial region based on GPS data using semantic information. In *Proc. 8th Int. Workshop on Databases in Networked Information Systems*, volume 7813 of *LNCS*, pages 262–276. Springer, 2013.

[33] F. Verhein and S. Chawla. Mining spatio-temporal association rules, sources, sinks, stationary regions and thoroughfares in object mobility databases. In *Proc. 11th Int. Conf. on Database Systems for Advanced Applications*, volume 3882 of *LNCS*, pages 187–201. Springer, 2006.

[34] M. Vlachos, D. Gunopulos, and G. Kollios. Discovering similar multidimensional trajectories. In *Proc. 18th Int. Conf. on Data Engineering*, pages 673–684, 2002.

[35] H. Yoon and C. Shahabi. Robust time-referenced segmentation of moving object trajectories. In *Proc. 8th IEEE Int. Conf. on Data Mining*, pages 1121–1126, 2008.