2012 IEEE 13th International Conference on Mobile Data Management

Online Identification of Dwell Regions for Moving Objects

M Reaz Uddin, Chinya V. Ravishankar, Vassilis J. Tsotras

University of California, Riverside, CA, USA
{uddinm,ravi,tsotras}@cs.ucr.edu

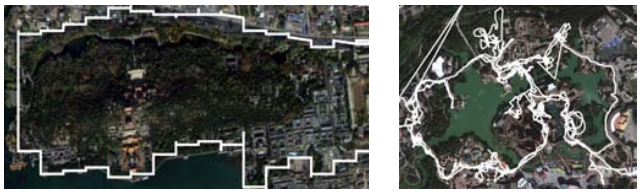
***Abstract*— A region $\mathcal{R}$ is a *dwell region* for a moving object O if, given a threshold distance d and duration t, every point of $\mathcal{R}$ remains within distance d of O for at least time t. Clearly, points within $\mathcal{R}$ are likely to be of interest to O, so identification of O and $\mathcal{R}$ has applications in areas such as monitoring and surveillance, as well as to trajectory simplification. We propose an online algorithm to solve this problem, which can handle dynamic addition and deletion of data in logarithmic time. We assume an incoming stream of object positions, and maintain the upper and lower bounds for the radius of the smallest circle enclosing these positions, as points are added and deleted. These bounds allow us to greatly reduce the number of trajectory points we need to consider in the query, as well as to defer query evaluation. Our method can approximate the radius of the smallest circle enclosing a given subtrajectory within an arbitrarily small user-defined factor. Our experiments show that the proposed method can scale up to hundreds of thousands of trajectories.**

I. Introduction

The widespread deployment of GPS devices has made real-time position data from millions of moving objects readily available. Many applications, especially those involving monitoring and control, require on-line analysis of such data. We need real-time responses to spatiotemporal queries, since positions change rapidly, and queries quickly lose their value. It is essential to keep pace with high incoming data rates.

We consider the problem of online identification of *dwell regions*. A region $\mathcal{R}$ is a dwell region for a moving object O if, given a radius r_q and duration t, if O remains within distance r_q from every point in $\mathcal{R}$ for at least time t. We also consider the case where t is not specified. Incoming position updates for O are grouped into a subtrajectory $\mathcal{S}$, ensuring that O remains within r_q distance from all points in some region. The problem reduces to computing the smallest enclosing circle $\mathrm{SEC}_{\mathcal{S}}$ of the points in $\mathcal{S}$. Computing $\mathcal{R}$ is hard for streaming data. We propose approximate methods to compute dwell regions $\mathcal{R}$.

This problem has many real life applications. In surveillance and security applications, the object O may represent a threat, and the region $\mathcal{R}$ may contain potential targets for O. For example, O might be collecting information about the region $\mathcal{R}$ or maintaining communication with objects in $\mathcal{R}$ which is only possible within a certain distance from $\mathcal{R}$. Fast detection of $\mathcal{R}$ might be of critical importance. Identifying dwell regions is also important in animal behavior tracking, and may reveal animal territories. Wolf packs are known to stalk prey before attacking it. Identifying such behaviors reveals many interesting facts and is very important to ecosystem researchers [1]. The behaviors we consider in this paper include both going around a region, as in Figure 1(a), or random movement in a certain enclosed region, as in Figure 1(b).



(a) (b)

Fig. 1. Behaviors considered in this paper.

Our work also has applications in real time trajectory simplification based on spatio-temporal criteria. One such criterion [2] is the "disk criterion", which collapses into one segment all contiguous trajectory segments that can be enclosed by a fixed size disk. Our data structures and algorithms can be used to maintain a subtrajectory as long as it satisfies the disk criterion. For streaming scenario it is desirable to do this simplification in real time, without storing the data in a physical medium.

We assume that every moving object sends regular position updates to a central system. For every moving object we will have a *streaming window* (or just *window*) of recent position records. We are to identify dwell regions as records are being added to and deleted from this window in real time.

Given a window $\mathcal{S}$, our approach approximates the smallest enclosing circle $\mathrm{SEC}_{\mathcal{S}}$ of the points $\mathcal{S}$ as a polygon with a user-specified number, k, of sides. We maintain data structures which are used to compute upper and lower bounds on the radius of $\mathrm{SEC}_{\mathcal{S}}$. We show that the actual radius is within a factor of $(1+O(\frac{1}{k^2}))$ of the lower bound. The data structures can be updated for addition/deletion in O($k \log n$) time, n is the window size $|\mathcal{S}|$. We can compute the upper and lower bounds in time O(k).

Most of the time, we can decide whether $\mathrm{SEC}_{\mathcal{S}}$ has radius r_q or less from just the upper and lower bounds. When these bounds are insufficient to evaluate the query, we propose a method which allows us to consider only a few points in the window to compute $\mathrm{SEC}_{\mathcal{S}}$. Computing $\mathrm{SEC}_{\mathcal{S}}$ only gives the center of the circle, not the complete region $\mathcal{R}$. We hence propose a method for quickly approximating the region $\mathcal{R}$. Our contributions are:

- We maintain an approximation of $\mathrm{SEC}_{\mathcal{S}}$ in logarithmic update time.

978-0-7695-4713-8/12 $26.00 © 2012 IEEE
DOI 10.1109/MDM.2012.41

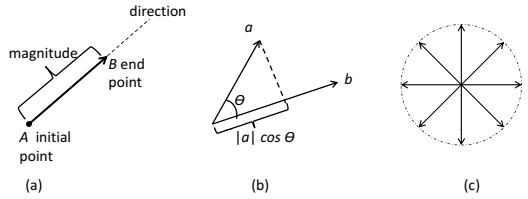


Fig. 2. (a) A Euclidean vector (b) The scalar projection of $\vec{a}$ onto $\vec{b}$ (c) Eight uniformly spaced vectors around a circle.

- We propose upper and lower bounds of the radius of $\mathrm{SEC}_{\mathcal{S}}$, greatly reducing computation time.
- We show that the radius of $\mathrm{SEC}_{\mathcal{S}}$ is within a factor of $(1 + O(\frac{1}{k^2}))$ of the lower bound, for a user defined k.
- We devise a method for selecting a few points using our data structures to compute $\mathrm{SEC}_{\mathcal{S}}$ exactly.
- We discuss how to compute a fairly good approximation of the dwell region $\mathcal{R}$.

The rest of the paper is organized as follows: Section II provides the definitions and background while Section III describes some related works. Our proposed methods and data structures are described in Section IV while Section V describes the query evaluation algorithms. Section VI presents the experimental results and Section VII concludes the paper.

II. Background

Every moving object has a unique object ID and sends its position updates at certain regular intervals. A position update record contains object ID, spatial coordinate, and a timestamp. A ***trajectory***, with a particular id, is a finite sequence of (x_i, y_i, t_i) triples. The $x_i, y_i \in \mathbb{R}^2$ are spatial coordinates, and the $t_i \in \mathbb{R}^+$, are timestamps, with $t_i < t_{i+1}$ for $i = 0, 1, \ldots, n-1$. A ***trajectory segment*** is a straight line between two consecutive tuples $(x_i, y_i, t_i), (x_{i+1}, y_{i+1}, t_{i+1})$ of the same trajectory, where $i \in \mathbb{N}_0$. A ***subtrajectory*** of length m of a trajectory $T = (x_0, y_0, t_0), \ldots, (x_n, y_n, t_n)$, is a subsequence $T' = (x_i, y_i, t_i), \ \ldots, \ (x_{m+i}, y_{m+i}, t_{m+i})$, of m contiguous trajectory segments, where $i \geq 0, m < n$. A single trajectory segment is a subtrajectory of length one.

For each moving object we maintain a streaming window. A ***streaming window*** of size n is the time-ordered sequence of the latest n positions of the moving object. The length of the window depends on the duration t specified by the query and the frequency of position updates from a moving object. A streaming window is updated by adding the most recent position when a new update record arrives and deleting the least recent point when necessary. For example, in applications like trajectory simplification, records can be added (without any deletion) as long as they satisfy the query condition.

We consider two types of queries. A *region query* (r_q, t) requests the dwell region $\mathcal{R}$, each point of which is within a distance r_q from the trajectory for time at least t. A *decision query* r_q asks whether the positions of a given object in the streaming window fall within distance r_q from any point in $\mathbb{R}^2$. This query returns a Boolean value and the center of the smallest enclosing circle. Decision queries are important for applications like trajectory simplification [2].

Consider circles of radius r_q centered at each point in a window $\mathcal{S}$. The intersection of these circles is precisely the dwell region $\mathcal{R}$, since all points of in streaming window are within r_q from any point in $\mathcal{R}$. We hence consider two approaches. The first maintains the overlap region of a set of circles centered at the object positions in $\mathcal{S}$. The other computes $\mathrm{SEC}_{\mathcal{S}}$. Finding $\mathrm{SEC}_{\mathcal{S}}$ suffices to answer decision queries, but not region queries.

A naive approach to maintaining the overlap between circles is to re-compute their intersections whenever a point is inserted into or deleted from the window. However, to the best of our knowledge, there exists no efficient on-line algorithm to maintain intersection of circles. Additions can be made fast, but a deletion is always $O(n)$ making the update time $O(n)$. Our proposed method is based on approximating the SEC with a polygon of k sides, and calculating the upper and lower bounds of the radius to answer decision queries. We allow the user to make the lower bound arbitrarily close to the actual radius of the SEC by tuning the value of k.

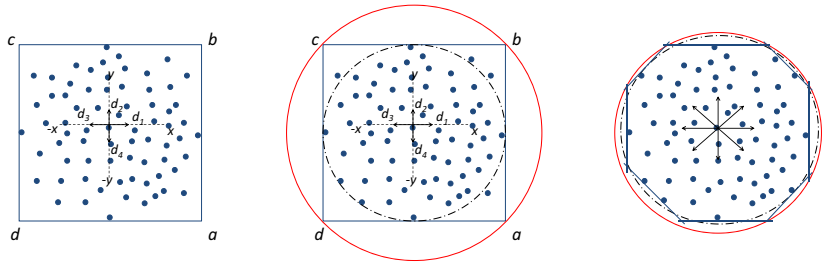
Computing $\mathrm{SEC}_{\mathcal{S}}$ gives only the center of $\mathrm{SEC}_{\mathcal{S}}$, not the entire dwell region $\mathcal{R}$. To answer region queries, we use efficient pruning to avoid unnecessary computation. First, no dwell region $\mathcal{R}$ can exist if the lower bound for the radius of $\mathrm{SEC}_{\mathcal{S}}$ exceeds r_q. In this case, we do not compute the intersecting region. When the upper bound is less than r_q, we compute an overestimate $\mathcal{R}^+$ and an underestimate $\mathcal{R}^-$ for the actual region $\mathcal{R}$. We also identify *critical points* that are more likely to affect the shape of the region. We can efficiently maintain approximations by considering only critical points. Details are described in Section V. We start with some basic definitions.

An n dimensional *vector* is an n-tuple, $(v_1, v_2, \ldots, v_n)$, where v_i is its component along the i^{th} axis. We use an overhead arrow to distinguish a vector, as in $\vec{a}$. The *magnitude* $|v|$ of a vector $\vec{v}$ is denoted as $|v| = \sqrt{v_1^2 + v_2^2 + \ldots + v_n^2}$. The *dot product* of two vectors $\vec{a}$ and $\vec{b}$ is defined as $\vec{a} \cdot \vec{b} = |a||b| \cos\theta = \sum_{i=1}^{n} a_i b_i$. If $\vec{b}$, is a unit vector then the dot product $|a| \cos\theta$ is the length of $\vec{a}$ in the direction of $\vec{b}$, also called the scalar projection of a onto b.

If k vectors are uniformly spaced around a circle, the angle between any two adjacent vectors is $\frac{2\pi}{k}$. Figure 2 shows (a) a Euclidean vector, (b) scalar projection of $\vec{a}$ onto $\vec{b}$ and (c) eight uniformly spaced vectors around a circle. In a Euclidean space each side of a straight line is called a *half space*. Given the straight line $ax_0 + by_0 = c$, $(x_0, y_0) \in \mathbb{R}^2$, one half space is $H = \{(x, y) : ax + by \leq c\}$.

III. Related Work

Trajectories have received much attention recently. The work in [3] considers pattern queries on trajectories, while [4] considers similarity queries for trajecotries. There has also been recent work on finding semantic information from trajectory data [5][6][7]. These works focus on identifying



(a) (b) (c)

Fig. 3. (a) MBR of a set of points (b) inner and outer circles (c) better estimation with higher number of directions.

Algorithm 1 MinDisk($\mathcal{S}$, B)

1: $\mathcal{S}$: a set of points, $\{p_1, p_2, \ldots, p_n\}$.
2: B : may contain at most 3 points, $T \subset \mathcal{S}$.
3: $P = B$ (P: set of points seen so far.)
4: $T = B$ (T: current basis of P.)
5: $D = \text{SEC}_{\mathcal{B}}$
6: **for** each $p_i \in S - B$ **do**
7: $P = P \cup \{p_i\}$
8: **if** p_i is not inside D **then**
9: $T = \text{MinDisk}(P, \textit{basis}(T \cup \{p_i\}))$
10: $D = \text{SEC}_{\mathcal{T}}$
11: **end if**
12: **end for**
13: return T;

regions where objects have remained for a while, but none of these consider identifying dwell regions. These works also do not consider streaming environments. Their approaches are not readily adaptible to our context.

There has also been work on identifying group behaviors, such as flock patterns [8], convoy detection [9], identifying density of moving objects [10], etc. However, in this paper we consider individual trajectories instead of group behavior.

The first deterministic linear time algorithm for the smallest enclosing circle appeared in [11]. Several improvements were presented in [12][13][14]. These methods are based on linear programming techniques and involve expensive computations, such as solving systems of polynomials. None of these methods was designed for streaming environments, and require repeating the computation for every addition/deletion. Welzl proposed a simple-to-implement randomized algorithm [15] with expected linear run-time. It recursively finds three points on the boundary of the circle. The algorithm can handle addition in constant time but deletion has expected linear time. However, the worst case runtime of Welzl's algorithm is quadratic. We will adapt Welzl's algorithm for computing $\text{SEC}_{\mathcal{S}}$ for a small set of points $\mathcal{S}$.

Algorithm 1 describes the pseudo code of Welzl's algorithm. Given a set of points $\mathcal{S} = \{p_1, p_2, \ldots, p_n\}$ there is a set, B, of at most three points that determines $\text{SEC}_{\mathcal{S}}$, e.g., $\text{SEC}_{\mathcal{B}} = \text{SEC}_{\mathcal{S}}$. B is called the basis of $\mathcal{S}$. Computation of basis of a set of at most 4 points can be done in constant time. The algorithm is started with the call MinDisk($\mathcal{S}$, B) where $B = \textit{basis}(p_1, p_2, p_3)$ and the initial circle $D = \text{SEC}_{\mathcal{B}}$. The remaining points in $\mathcal{S}$ are tested one by one whether they are inside D. If the next point p_i is inside D then D is the smallest enclosing circle of the set of points seen so far, P. Otherwise, a recursive call (line 8) is made to compute $\text{SEC}_{\mathcal{P}}$. This time the basis is $\textit{basis}(B \cup \{p_i\})$. When this recursive call returns we have the basis and the smallest enclosing circle of points seen so far.

Several heuristics were proposed in [16] for computing circle intersections. However, these heuristics are not useful in environments where points are being added/deleted dynamically. This approach maintains an R-tree [17] for all the *static sites* and computes intersection of circles only when a moving object is out of the *safe zone*. The computation requires traversing the R-tree and making a heap of R-tree entries. With streaming data, the R-tree must be updated and traversed, building the heap for every addition/deletion. Moreover, we will show that our proposed data structures render three of the heuristics in [16] unnecessary.

IV. Data Structures & Algorithms

We present our algorithms and data structures for approximating the smallest enclosing circle $\text{SEC}_{\mathcal{S}}$. We also show how to bound the radius of $\text{SEC}_{\mathcal{S}}$ above and below using circles constructed from the minimum bounding polygon for $\mathcal{S}$. Some symbols used in this paper are listed in table I.

A. Minimum Bounding Polygons

Figure 3(a) shows a set of points and their minimum bounding rectangle (MBR). MBRs indicate the maximum extents of a set of objects or points $\mathcal{S}$. In the 2-D case, we can construct an MBR for $\mathcal{S}$ as follows. We take four vectors $\vec{d_1}, \vec{d_2}, \vec{d_3}, \vec{d_4}$, spaced 90° apart, and four lines $e_i \perp \vec{d_i}, 1 \leq i \leq 4$. Now we sweep each e_i in the direction of $\vec{d_i}$, in from infinity towards $\mathcal{S}$. We stop when each e_i touches a point $p_i \in \mathcal{S}$. The lines e_i form the edges of the MBR. We denote the set of vertices of the MBR as $\mathcal{V}$.

We can generalize this idea to get tighter upper and lower bounds by using k uniformly spaced vectors $\vec{d_1}, \vec{d_2}, \ldots, \vec{d_k}$, $k > 4$. Figure 3(c) shows eight uniformly

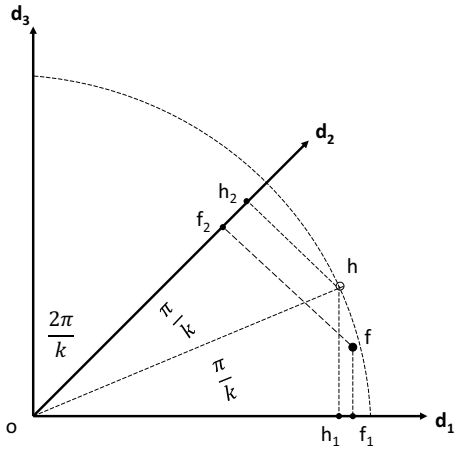


Fig. 4. Computing $\mathcal{S}'$. Although f is a frontier point in the direction of $\vec{d_1}$, it may not lie on the convex hull. Point h is farther from the center of $\mathrm{SEC}_{\mathcal{S}}$ than f. We include all points whose projections exceed $|Of_1|\cos\left(\frac{\pi}{k}\right)$.

spaced vectors and the corresponding bounding convex octagon. As before, the lines e_i will be swept inwards from infinity until they touch points $p_i \in \mathcal{S}$.

Definition 1: The set $\mathcal{F} = \{p_i \in \mathcal{S}\}$ which the lines e_i touch is the set of the *frontier* points of $\mathcal{S}$ in the directions $\vec{d_i}$. We denote the k-polygon bounding the set $\mathcal{S}$ by $MBP(\mathcal{S})$. Clearly, if $\mathcal{V}$ is the set of vertices of $MBP(\mathcal{S})$, then $\mathcal{V} \not\subset \mathcal{S}$. However, $\mathcal{F} \subset \mathcal{S}$ by definition.

B. Queries Using $\mathrm{SEC}_{\mathcal{S}}$ and $MBP(\mathcal{S})$

Let $r_{\mathrm{SEC}_{\mathcal{S}}}$ be the radius of $\mathrm{SEC}_{\mathcal{S}}$. We can now get upper and lower bounds for $r_{\mathrm{SEC}_{\mathcal{S}}}$ as follows. The smallest circle $\mathrm{SEC}_{\mathcal{V}}$ enclosing the set $\mathcal{V}$ of vertices of $MBP(\mathcal{S})$ is guaranteed to contain all the points of $\mathcal{S}$, and yields an overestimate for $\mathrm{SEC}_{\mathcal{S}}$ (see Figure 3(b)). Similarly, $\mathrm{SEC}_{\mathcal{F}}$, the smallest circle enclosing all the frontier points of $\mathcal{S}$ yields an underestimate for $\mathrm{SEC}_{\mathcal{S}}$.

Definition 2: Let the bounding k-polygon $MBP(\mathcal{S})$ for a set of points $\mathcal{S}$, have vertex set $\mathcal{V}$ and frontier $\mathcal{F}$. We define the *under-circle* for $\mathcal{S}$ as $\lfloor \mathrm{SEC}_{\mathcal{S}} \rfloor \triangleq \mathrm{SEC}_{\mathcal{F}}$ and the *over-circle* for $\mathcal{S}$ as $\lceil \mathrm{SEC}_{\mathcal{S}} \rceil \triangleq \mathrm{SEC}_{\mathcal{V}}$. (See Figure 3.)

Let $r_{\lfloor \mathrm{SEC}_{\mathcal{S}} \rfloor}$ and $r_{\lceil \mathrm{SEC}_{\mathcal{S}} \rceil}$ of $\lfloor \mathrm{SEC}_{\mathcal{S}} \rfloor$, and $\lceil \mathrm{SEC}_{\mathcal{S}} \rceil$, respectively. Clearly, $r_{\lfloor \mathrm{SEC}_{\mathcal{S}} \rfloor} \leq r_{\mathrm{SEC}_{\mathcal{S}}} \leq r_{\lceil \mathrm{SEC}_{\mathcal{S}} \rceil}$. We will show that the distance from the center of $\lfloor \mathrm{SEC}_{\mathcal{S}} \rfloor$ to any point in $\mathcal{S}$ will be at most $(1 + O(\frac{1}{k^2}))r_{in}$.

$r_{\lfloor \mathrm{SEC}_{\mathcal{S}} \rfloor}$ and $r_{\lceil \mathrm{SEC}_{\mathcal{S}} \rceil}$ are useful in optimizing decision queries. The query response can be YES or NO whenever $r_q > r_{\lceil \mathrm{SEC}_{\mathcal{S}} \rceil}$ or $r_q < r_{\lfloor \mathrm{SEC}_{\mathcal{S}} \rfloor}$ respectively.

C. Constructing $\mathrm{SEC}_{\mathcal{S}}$

If neither $r_q > r_{\lceil \mathrm{SEC}_{\mathcal{S}} \rceil}$ nor $r_q < r_{\lfloor \mathrm{SEC}_{\mathcal{S}} \rfloor}$ holds, we must construct $\mathrm{SEC}_{\mathcal{S}}$ explicitly. We now show how to construct this with minimal overhead.

Definition 3: The *convex hull* $\mathcal{C}(\mathcal{S})$ of a given set $\mathcal{S}$ is the minimal convex region enclosing $\mathcal{S}$. $\mathcal{H}(\mathcal{S})$ is the set of points defining the boundary of $\mathcal{C}(\mathcal{S})$.

In our case, we know that the convex hull of $\mathcal{S}$ is a convex polygon whose vertices, $\mathcal{H}(\mathcal{S})$, are all in $\mathcal{S}$.

Lemma 1: If $\mathcal{C}(\mathcal{S})$ is the convex hull for a set $\mathcal{S}$, then $\mathrm{SEC}_{H(\mathcal{S})} = \mathrm{SEC}_{\mathcal{S}}$.

Proof: By definition, $\mathrm{SEC}_{H(\mathcal{S})}$ is the minimal circle enclosing $\mathcal{C}(\mathcal{S})$, which is the minimal convex region enclosing $\mathcal{S}$. □

We proceed as follows. Clearly, $\mathcal{C}(\mathcal{S}) \subseteq MBP(\mathcal{S})$, since $MBP(\mathcal{S})$ may include dead space beyond $\mathcal{C}(\mathcal{S})$. We identify a subset $\mathcal{S}' \subseteq \mathcal{S}$ such that $\mathcal{H}(\mathcal{S}) \subseteq \mathcal{S}'$. We will find $\mathrm{SEC}_{\mathcal{S}'}$, which will give us $\mathrm{SEC}_{H(\mathcal{S})}$, and consequently $\mathrm{SEC}_{\mathcal{S}}$. This approach is efficient, since we expect $\mathcal{S}'$ to be much smaller than $\mathcal{S}$.

D. The Algorithm

The quality of the algorithm depends on k, the number of uniformly spaced vectors used. The algorithm maintains the frontier point corresponding to each vector. We now show how to identify frontier points, and how to update them dynamically as points are added to and deleted from the window $\mathcal{S}$.

Each point $p \in \mathcal{S}$ defines a vector. To identify a frontier point lying on an edge of $MBP(\mathcal{S})$, we identify a point whose projection onto the corresponding unit vector has maximum length. We calculate the dot product of each of the k vectors with each point of $\mathcal{S}$, and use the point with maximum projection length for each of the edges. The algorithm works as follows.

Select k unit vectors $\vec{d_1}, \vec{d_2}, \ldots, \vec{d_k}$ uniformly spaced around a circle. For each $\vec{d_i}$, maintain the point p in the current set that maximizes $\vec{d_i} \cdot \vec{p}$, which will be the point p furthest in direction of $\vec{d_i}$. This requires computing n dot products and building a max heap with the values of these dot products. There is one max heap for each vector. The point at the root of a heap is the one that has maximum scalar projection on the corresponding vector. Calculating the dot products is $O(n)$ for each vector and is performed only once, the first time a moving object reaches the required window size. Heap building is also done at the same time and its complexity is $O(n \log n)$. Addition (deletion) of a point from the streaming window requires one addition (deletion) from the heap. This can be done in $\mathrm{O}(\log n)$ time per unit vector i.e. $O(k \log n)$ time for k vectors. The set $\mathcal{F}$ consists of the points at the heap roots. The set $\mathcal{V}$ is computed from the intersection of adjacent edges of $MBP(\mathcal{S})$.

Algorithm 2 describes the initial processing. The dot product between each vector and each point is calculated in line 7. One heap is built with the values of dot products for each vector. The heapify operation at line 9 builds the heap, which takes $O(n \log n)$. The dot product calculation for each vector in lines 6—8 is $O(n)$. As a result, the complexity of this preprocessing for k vectors is $O(kn + kn \log n) = O(kn \log n)$. Note that a particular point will be at different locations in different heaps because of different dot product values with different vectors. If we want to access a particular point p_j and/or its dot product $\vec{d_i} \cdot \vec{p_j}$ of with the vector $\vec{d_i}$ we need to

$\mathcal{S}$	A set of 2D points (possibly from the streaming window).
$\text{SEC}_{\mathcal{S}}$	Smallest Enclosing Circle of a set $\mathcal{S}$ of points
$MBP(\mathcal{S})$	Minimum k-polygon bounding $\text{SEC}_{\mathcal{S}}$
$\mathcal{F}$	Frontier points $\mathcal{S}$ on the edges of $MBP(\mathcal{S})$.
r	Radius of $\text{SEC}_{\mathcal{S}}$.
r_{in}	Radius of $\lfloor \text{SEC}_{\mathcal{S}} \rfloor$, the inner circle.
r_{out}	Radius of $\lceil \text{SEC}_{\mathcal{S}} \rceil$, the outer circle.
H	Half space.
$r(S)$	Radius of the $\text{SEC}_{\mathcal{S}}$.
h_i	Heap corresponding to unit vector d_i

TABLE I

SYMBOLS USED IN THIS PAPER.

Algorithm 2 buildHeaps(S, H, d)

```
S : streaming window, {p_1, p_2, ..., p_n}.
H : k heaps, {h_1, h_2, ..., h_k}. One for each direction being tracked.
d : k directions, {d_1, d_2, ..., d_k}.

for each d_i in d do
    for each point p_j in S do
        h_i[j] = d_i.p_j
    end for
    heapify(h_i)
end for
```

know where is this value in the heap h_i. For example we need to access a point when if it is to be deleted. To achieve this we also build a *Lookup Table* (LT) while building these heaps. The lookup table contains the location of a point in every heap. For example $LT[\vec{d_i}][\vec{p_j}]$ contains a pointer to $\vec{d_i} \cdot \vec{p_j}$ in h_i.

Algorithm 3 update(S, H, d, p)

```
S : streaming window, {p_1, p_2, ..., p_n}.
H : k heaps, {h_1, h_2, ..., h_k}. One for each direction being tracked.
d : k directions, {d_1, d_2, ..., d_k}.
p : next point to be added in S.

for each d_i in d do
    delete LT[d_i][p_1] from h_i
    val = d_i.p
    insert val into h_i
end for
delete p_1 from S
add p to the end of S
```

After building the heaps we must update them when inserting and deleting points from the $\mathcal{S}$, Algorithm 3 describes the update step. At each update step, a point p is added to the window $\mathcal{S}$ as the most recent point, and the least recent point, p_1, is deleted from $\mathcal{S}$. This requires deleting the record for to p_1 from each heap (LT is used). Next, the dot product between p and each vector $\vec{d_i}$ is calculated and the result inserted in the corresponding heap. Insertion and deletion in a heap being $O(\log n)$, update is a $O(k \log n)$ operation.

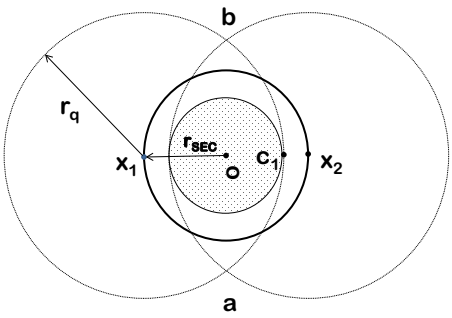


Fig. 5. The shaded region is $\mathcal{R}^-$.

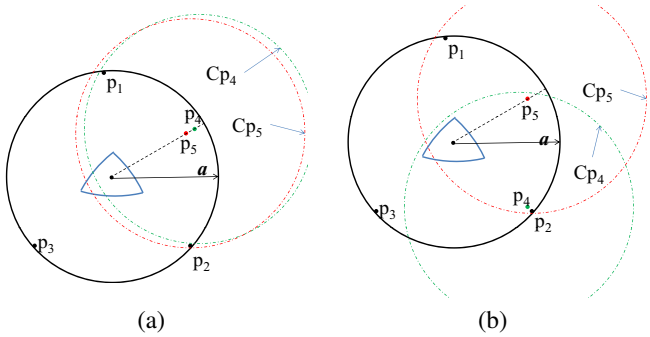


Fig. 6. Showing the importance of points near the boundary to compute the dwell region $\mathcal{R}$ when the are at (a) the same angular position (b) a different angular position.

V. QUERY EVALUATION

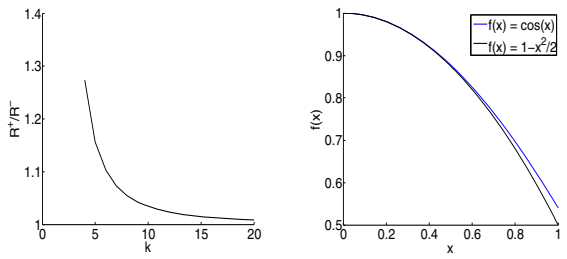
We have described the data structures used to maintain the upper and lower bounds and to approximate the radius of $\text{SEC}_{\mathcal{S}}$. We now show how to evaluate queries. We first consider decision queries, which ask whether or not the current window satisfies the query condition.

As we have seen, we maintain k heaps h_i corresponding to the k vectors $\vec{d_i}$. The root of heap h_i is a frontier point, since it maximizes the extent of $MBP(\mathcal{S})$ in $\vec{d_i}$'s direction. Let the bounding k-polygon $MBP(\mathcal{S})$ for $\mathcal{S}$ have vertices $\mathcal{V}$ and frontier points $\mathcal{F}$.

To get $r_{\lfloor \text{SEC}_{\mathcal{S}} \rfloor}$ and $r_{\lceil \text{SEC}_{\mathcal{S}} \rceil}$, we use Welzl's algorithm to compute $\text{SEC}_{\mathcal{V}}$ and $\text{SEC}_{\mathcal{F}}$ (see Definition 2). As already noted, $r_{\lfloor \text{SEC}_{\mathcal{S}} \rfloor}$ and $r_{\lceil \text{SEC}_{\mathcal{S}} \rceil}$ can be used to answer decision queries without actually computing $\text{SEC}_{\mathcal{S}}$. However, when $r_{\lfloor \text{SEC}_{\mathcal{S}} \rfloor}$ and $r_{\lceil \text{SEC}_{\mathcal{S}} \rceil}$ are insufficient for this purpose (when $r_q < r_{\lceil \text{SEC}_{\mathcal{S}} \rceil}$ and $r_q > r_{\lfloor \text{SEC}_{\mathcal{S}} \rfloor}$) we must compute $\text{SEC}_{\mathcal{S}}$.

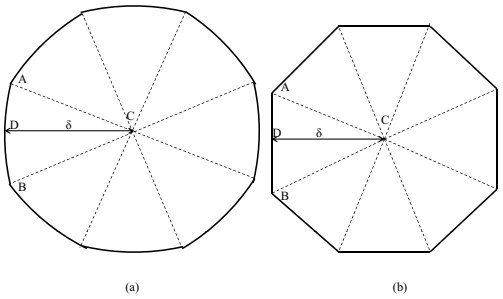
A. *Efficient Computation of* $\text{SEC}_{\mathcal{S}}$

We now show how to compute $\text{SEC}_{\mathcal{S}}$ using only a small subset $S' \subset S$ of points. Our central idea is to identify a set of points $\mathcal{S}'$ that includes all points on the convex hull $\mathcal{C}(\mathcal{S})$. In Figure 4, let O be the center of $\text{SEC}_{\mathcal{S}}$, and consider the angular sector between vectors $\vec{d_1}$ and $\vec{d_2}$. Let f be the point in this sector with the maximal projection on to $\vec{d_1}$, so f is the frontier point in the direction of $\vec{d_1}$.



(a) (b)

Fig. 7. Trigonometric functions. (a) The ratio $\frac{\mathcal{R}^+}{\mathcal{R}^-} = \frac{k \tan\theta}{\pi}$ as k changes (b)Approximation of $\cos x$ with $1 - \frac{x^2}{2!}$.



(a) (b)

Fig. 8. Replacing the bounding arcs of the dwell region with straight lines. (a) actual region (b) bounding arcs replaced with straight lines.

However, f need not be the furthest point in this sector from O. As Figure 4 shows, there could be a non-frontier point h in this sector such that $|Oh| > |Of|$, but if we consider their projections h_1, f_1 on $\vec{d_1}$, we have $|Oh_1| < |Of_1|$. It is quite likely that h lies on the convex hull $\mathcal{C}(\mathcal{S})$, but we would miss it if we only looked at projections on the vectors $\vec{d_i}$.

Our challenge is to include all such points h in $\mathcal{S}'$. We first note that this situation arises because the angular distance of f from $\vec{d_1}$ is less than that of h. (If we consider their projections h_2, f_2 on to $\vec{d_2}$, we find $|Oh_2| > |Of_2|$.) We first observe that the projection of f on $\vec{d_1}$ will be largest when f lies on $\vec{d_1}$.

Let the line Oh bisect the sector between $\vec{d_1}$ and $\vec{d_2}$. (We make this choice since it also minimizes the projection of h on both $\vec{d_1}$ and $\vec{d_2}$, and maximizes the likelihood that point h will not be a frontier point.) Let f lie on $\vec{d_1}$ and rotate the line Of so that it coincides with the bisector Oh. The projection of f on $\vec{d_1}$ will now be $\lambda_f = |Of| \cos\left(\frac{\pi}{k}\right)$. By selecting all points whose projections on $\vec{d_1}$ are of length at least λ_f, we are sure to get all points in the half-sector that are at least as far from the center as f is, and remain candidates for $\mathcal{C}(\mathcal{S})$. To get $\mathcal{S}'$, we proceed as follows:

1) Place all frontier points $f_i \in \mathcal{F}$ into $\mathcal{S}'$.
2) Let f_i be the frontier point in the direction of vector $\vec{d_i}$, and let its projection on $\vec{d_i}$ be λ_{f_i} Place into $\mathcal{S}'$ all points in the half-sector between $\vec{d_i}$ and $\vec{d_{i+1}}$ whose projections on $\vec{d_i}$ are larger than $\lambda_{f_i} \cos\left(\frac{\pi}{k}\right)$.

We can now state the following result.

Theorem 1: $\mathrm{SEC}_{\mathcal{S}} = \mathrm{SEC}_{\mathcal{S}'}$.

Proof: Since the convex hull $\mathcal{C}(\mathcal{S})$ is the maximal convex region enclosing $\mathcal{S}$, all frontier points $f_i \in \mathcal{F}$ are in $\mathcal{C}(\mathcal{S})$. Step 1) above places $\mathcal{F}$ into $\mathcal{S}'$. However, not all points in $\mathcal{C}(\mathcal{S})$ are in $\mathcal{F}$. Convexity of $\mathcal{C}(\mathcal{S})$ ensures that such points must be farther from the center of $\mathrm{SEC}_{\mathcal{S}}$ than the frontier points. Step 2) above places all such points into $\mathcal{S}'$. □

By Theorem 1, we need consider only points in $\mathcal{S}'$, which is much smaller than $\mathcal{S}$. Queries now run much faster.

B. Dwell Region Queries

As we have seen, the dwell region $\mathcal{R}$ is the intersection of all the circles of radius r_q centered at each point of $\mathcal{S}$. When $r_{\mathrm{SEC}_{\mathcal{S}}} = r_q$, the circles centered at the points on the perimeter of $\mathrm{SEC}_{\mathcal{S}}$ will share only its center. In that case, $\mathcal{R}$ will consist of only one point, namely, the center of $\mathrm{SEC}_{\mathcal{S}}$.

Next, consider the case when $r_q \geq r_{\mathrm{SEC}_{\mathcal{S}}}$. Let C_x be a circle of radius r_q centered at a point x on the circumference of $\mathrm{SEC}_{\mathcal{S}}$. Consider the region $\mathcal{R}^- = \bigcap_x C_x$. (See shaded region in Figure 5.) If we move the center of C_x along the circumference of $\mathrm{SEC}_{\mathcal{S}}$, the intersection of the resulting circles will be a disk $\mathcal{R}^-$ of radius $\delta = r_q - r_{\mathrm{SEC}_{\mathcal{S}}}$ centered at the center of $\mathrm{SEC}_{\mathcal{S}}$.

We can also calculate a region, $\mathcal{R}^+$, that contains $\mathcal{R}$. The intuition behind our method is as follows. Suppose we have a partially computed region $\mathcal{R}_m$, which is the intersection of some m circles. To get $\mathcal{R}$, we must compute the intersection of remaining $|S| - m$ circles with $\mathcal{R}_m$. Now, if any of these circles fully contains $\mathcal{R}_m$, then it will not affect $\mathcal{R}_m$ at all. Our goal is to use those points first that are more likely to intersect $\mathcal{R}_m$, since the remaining circles are less likely to have an effect on $\mathcal{R}_m$.

Intuitively, points closer to the boundary of $\mathrm{SEC}_{\mathcal{S}}$ are more likely to affect the region $\mathcal{R}_m$, as we will illustrate through an example. Our reasoning is similar to that used to prove Lemma 3 of [16].

Figure 6(a) shows $\mathrm{SEC}_{\mathcal{S}}$ for som set of points $\mathcal{S}$. Assume that points p_1, p_2, p_3 are on the boundary and c is the center of the $\mathrm{SEC}_{\mathcal{S}}$. The partial region $\mathcal{R}_m$ appears near the center as the intersection of three circles of radius r_q centered at p_1, p_2, p_3, repectively. Consider two other points p_4, p_5 inside $\mathrm{SEC}_{\mathcal{S}}$ lying on the same radial vector $\vec{a}$. Let p_5 be closer to the center of $\mathrm{SEC}_{\mathcal{S}}$ than p_4, and let C_{p_4} and C_{p_5} be circles of radius r_q centered at p_4, p_5 respectively. The minimum distances from the center c to any point on C_{p_4} and C_{p_5} are $\delta_1 = r_q - |cp_4|$ and $\delta_2 = r_q - |cp_5|$ respectively. Since $\delta_1 < \delta_2$, C_{p_5} is more likely to fully include $\mathcal{R}_m$ than C_{p_4}.

This illustration shows the importance of points closer to the boundary that are at a same angular position in the circle. However, if p_4 and p_5 had different angular position then p_5 might have trimmed $\mathcal{R}'$ more than p_4, Figure 6(b). Nevertheless, if the points are uniformly distributed, considering them in order from the boundary towards the center will still give us a better chance to get points which are more likely to affect the shape of $\mathcal{R}$. Our experimental evaluation shows that when

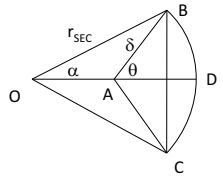


Fig. 9. Approximating $\mathcal{R}^+$.

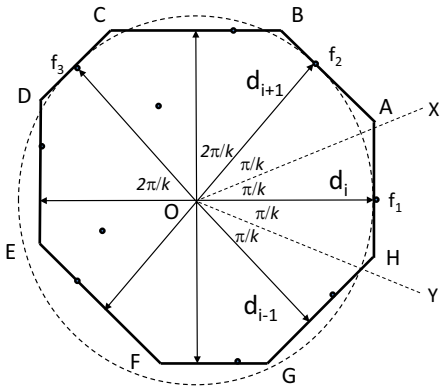


Fig. 10. Proving Theorem 2. $MBP(\mathcal{S}) = \text{ABCDEFGH}$.

the answer to the decision query is "yes", points are fairly uniformly distributed around the circle and thus the above heuristic applies in practice.

To calculate a region $\mathcal{R}^+ \supseteq \mathcal{R}$, it suffices to consider points in a subset of $\mathcal{S}$. Our goal is to make this subset as small as possible, and make $\mathcal{R}^+$ as close to $\mathcal{R}$ as possible. Towards this goal, we select points closer to the boundary of $\text{SEC}_\mathcal{S}$ first. The heap data structures we maintain can be used to select points that are closer to the boundary of the circle. If we consider only the k points that make up the frontier points (the heap roots), we will get an intersecting region that contains $\mathcal{R}$. If a tighter approximation is required (at the cost of more CPU time) points in the set $\mathcal{S}'$ (figure 4) can be considered.

[16] describes five heuristics to discard circles which are not going to affect the intersecting region. Heuristics 2, 3, 5, are used to discard circles that do not have a common intersecting region. For our case since we want to find the intersection only when every circle shares the intersecting region we do not need to consider heuristics 2, 3, 5. Heuristics 1, 4, are used to discard circles that fully contain the intersecting region computed so far and so are not going to affect the region. By considering points in the above mentioned order and applying heuristics 1 and 4 from [16] we can further avoid computing unnecessary circle intersections.

The correctness of the algorithm for the decision query follows from 1) the fact that we actually compute the SEC when upper/lower bounds cannot decide the query answer and 2) theorem 1.

C. Goodness of the Approximation

To measure the goodness of the region approximations, we will attempt to derive the ratio $\mathcal{R}^+/\mathcal{R}^-$. The area of the circular region $\mathcal{R}^-$ is $\pi\delta^2$, where $\delta = (r_q - r_{\text{SEC}_\mathcal{S}})$. We calculate the area of $\mathcal{R}^+$ in the following ideal scenario. We assume $\mathcal{R}^+$ is calculated using k frontier points at the heap roots, so that there are k circular arcs defining the boundary of $\mathcal{R}^+$. We further assume that the arc lengths are equal. Figure 8(a) shows $\mathcal{R}^+$ with eight bounding arcs. Each of the sectors in this figure is equivalent to the sector ABDC in Figure 9. We obtain the area of this sector as follows.

We begin by noting that $|BC| = 2\delta\sin(\theta)$, and further that

$$\alpha = \arcsin\left(\frac{|BC|}{2r_{\text{SEC}_\mathcal{S}}}\right) = \arcsin\left(\frac{\delta\sin(\theta)}{r_{\text{SEC}_\mathcal{S}}}\right). \qquad (1)$$

Standard formulas yield the area of the circular segment

$$\text{BCDB} = \frac{r^2_{\text{SEC}_\mathcal{S}}}{2}\left(2\alpha - \sin(2\alpha)\right).$$

Now, applying elementary trigonometry and simplifying,

$$\begin{aligned}\text{ABDC} &= \text{ABCA} + \text{BCDB}\\ &= \frac{1}{4}\delta^2\sin(2\theta) + \frac{r^2_{\text{SEC}_\mathcal{S}}}{2}\left(2\alpha - \sin(2\alpha)\right).\end{aligned}$$

The combined area of all sectors in Figure 8 is k times the above area. Hence,

$$\begin{aligned}\frac{\mathcal{R}^+}{\mathcal{R}^-} &= \frac{k\left(\frac{1}{4}\delta^2\sin(2\theta) + \frac{r^2_{\text{SEC}_\mathcal{S}}}{2}\left(2\alpha - \sin(2\alpha)\right)\right)}{\pi\delta^2}\\ &= \frac{k}{\pi}\left(\frac{1}{4}\sin(2\theta) + \left(\frac{r_{\text{SEC}_\mathcal{S}}}{\delta}\right)^2\left(\alpha - \frac{1}{2}\sin(2\alpha)\right)\right)\end{aligned}$$

We can now use the value of α in Equation 1.

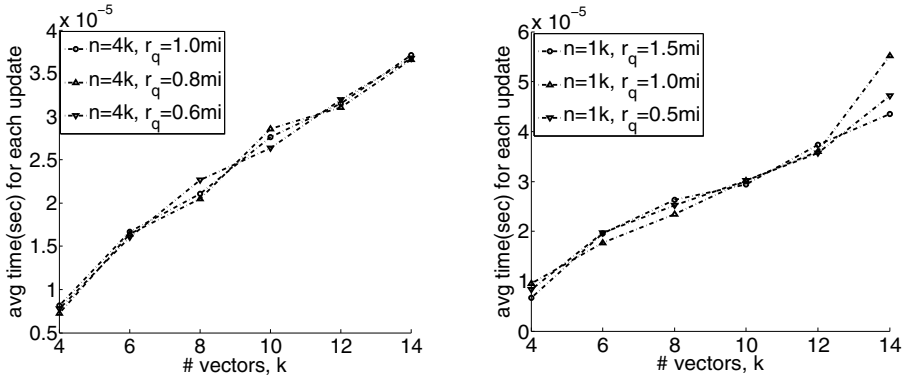
While the above analysis gives the exact equation of the ratio, we do the following estimation to better understand how it changes with k. Consider the area $CADB$ where C is the center of the $\text{SEC}_\mathcal{S}$. Recall that the centers of the bounding arcs are on the boundary of $\text{SEC}_\mathcal{S}$. The closest point from C on arc AB is D, with $|CD| = \delta$. The bounding arcs are expected to be very small and hence these arcs can be replaced with straight lines at distance δ from C. The result is a polygon with k boundary lines. Figure 8(b) shows the polygon with eight lines which replaces the area of figure 8(a). The computation of $\mathcal{R}^+/\mathcal{R}^-$ is given below-

$$\begin{aligned}&\angle ACD = \frac{1}{2}\angle ACB = \pi/k = \theta\\ &AD = \delta\tan\theta\\ &\triangle\, CAD = \frac{1}{2}AD \times CD = \frac{1}{2}\delta^2\tan\theta\\ &\Rightarrow \triangle\, CAB = \delta^2\tan\theta\\ &\therefore \frac{\mathcal{R}^+}{\mathcal{R}^-} = \frac{k\delta^2\tan\theta}{\pi\delta^2}\\ &\qquad = \frac{k\tan\theta}{\pi}\end{aligned}$$

Location	Time of collection	Number of trajectories	Number of spatial points	Sampling frequency	Description
Beijing, China	Apr 2007 to Aug 2009	165	24778552	2-5 sec	**GeoLife Data:**
Starkey, Oregon, USA	April to August of 1993-1996	253	>287,000	1hr	**DeerElk Data:**

TABLE II

DESCRIPTION OF REAL DATA SET.



(a)GeoLife Data (b)ElkDeer Data

Fig. 11. Average time required for each update of the data structures and evaluating the query for a moving object.

We want this ratio to be as close to 1.0 as possible. Figure 7(a) shows how this ratio decreases towards 1.0 as k increases. As we increase k linearly, $\tan\theta$ decreases at a faster rate than the rate of increase of k. This results in decreasing the ratio with the increase of k.

Finally, we prove that $r_{\mathrm{SEC}_\mathcal{S}} \approx (1 + O(1/k^2))r_{\mathrm{SEC}_\mathcal{F}}$. This means that we can get an arbitrarily good approximation of the radius by maintaining just a constant number k of direction vectors, in $\mathrm{O}(\log n)$ time per update.

Theorem 2: Let $\mathcal{S}$ be the current set of points in the streaming window, and $\mathcal{F}$ be the set of frontier points. Then, $r_{\mathrm{SEC}_\mathcal{S}} \leq r_{\mathrm{SEC}_\mathcal{F}}(1 + O(1/k^2))$.

Proof: Figure 10 shows a set of points $\mathcal{S}$, their minimum bounding k-polygon $MBP(\mathcal{S}) = \mathrm{ABCDEFGH}$, the set $\mathcal{F}$ of frontier points in the directions of vectors $\vec{d_1}, \vec{d_2}, \ldots$, as well as the frontier circle $\mathrm{SEC}_\mathcal{F}$ defined by the frontier points f_1, f_2, and f_3. Let O be the center of $\mathrm{SEC}_\mathcal{F}$. Clearly, $|Of_1| = r_{\mathrm{SEC}_\mathcal{F}}$. Let OX bisect the angle between $\vec{d_i}$ and $\vec{d_{i+1}}$, and OY bisect the angle between $\vec{d_i}$ and $\vec{d_{i-1}}$.

The sector between the bisectors OX and OY is fully contained in the isosceles triangle defined by the lines OX, OY, and AH (extended as needed). From elementary trigonometry, all points in this isosceles triangle, and hence all points in the sector of $MBP(\mathcal{S})$ defined by OX and OY lie within distance $|Of_1|/\cos\left(\frac{\pi}{k}\right) = r_{\mathrm{SEC}_\mathcal{F}}/\cos\left(\frac{\pi}{k}\right)$ of the point O.

Since all points in $\mathcal{S}$ lie within the polygon $MBP(\mathcal{S})$, all points $p_i \in \mathcal{S}$ in this sector are within distance $r_{\mathrm{SEC}_\mathcal{F}}/\cos\left(\frac{\pi}{k}\right)$ of the point O. Clearly, this means that we can include all of $\mathcal{S}$ in a circle of this radius.

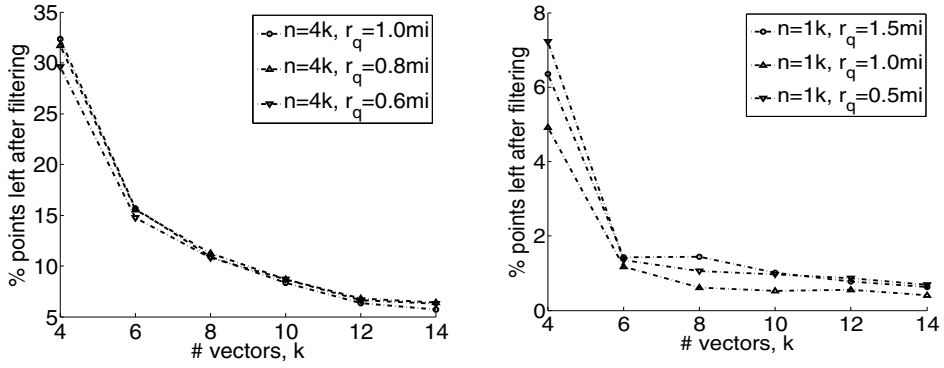
Hence, $r_{\mathrm{SEC}_\mathcal{S}} \leq r_{\mathrm{SEC}_\mathcal{F}}/\cos\left(\frac{\pi}{k}\right)$. We can now use a Taylor series expansion to obtain $\cos\left(\frac{\pi}{k}\right) = 1 - \frac{\pi^2}{2k^2} + O\left(\frac{\pi^4}{k^4}\right)$, so that $\cos\left(\frac{\pi}{k}\right) = 1 - O\left(\frac{1}{k^2}\right)$ as $\left(\frac{\pi}{k}\right) \to 0$, that is, as k increases. We now have $r_{\mathrm{SEC}_\mathcal{S}} \leq r_{\mathrm{SEC}_\mathcal{F}}\left(1 + O\left(\frac{1}{k^2}\right)\right)$. □

VI. EXPERIMENTS

All the experiments were run in an Intel Xeon 3.0GHz processor running Linux 2.6.18 with 8GB of main memory. In our experiments we use two real datasets. The GeoLife dataset [18] contains public activity data (i.e. shopping, dining, sightseeing, hiking, cycling etc.) in Beijing, China. The DeerElk data contains the trajectories of deer, elk and cattle in the Starkey Experimental Forest and Range in Oregon, USA [19]. Table II provides the description of the real datasets. Different set of parameter (window size, query radius, and number of vectors) values were considered for different datasets as indicated in the plots.

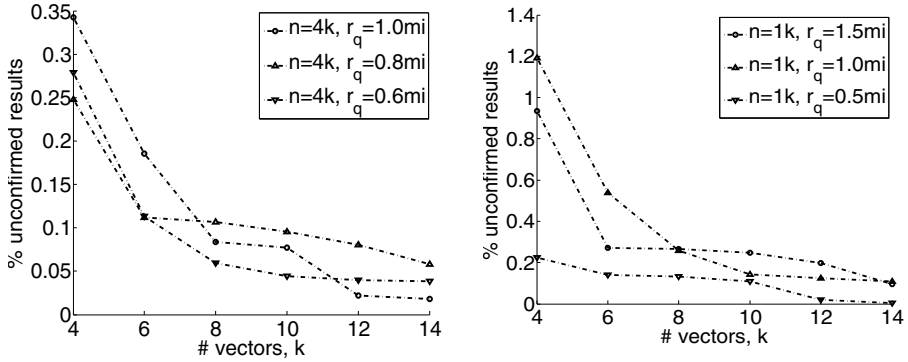
First, we examine the average time required to update the data structure as the number of vectors changes. The experimental results appear in figure 11. As expected, with the increase of the number vectors the time required for each update increases. However the rate of increase is very slow, e.g., using up to 10 vectors the update process is efficient enough to handle orders of hundreds of thousands of moving objects. Since the approximated radius is a factor of $1+O(\frac{1}{k^2})$, using $k = 10$ can approximate the radius within order of 1% of the actual radius.

To depict the pruning power of our data structures, next, we compute the fraction of points used by our method to evaluate the query when the upper and lower bounds are not enough to answer the query. As seen in figure 12, while the number of vectors increases, the fraction of points considered falls sharply. For example with 10 vectors we need to consider only 10% of the window size.



(a)GeoLife Data (b)ElkDeer Data

Fig. 12. Fraction of points selected to calculate actual SEC.



(a)GeoLife Data (b)ElkDeer Data

Fig. 13. Fraction of results where actual SEC has to be computed (e.g., heuristics do not answer the query).

4k, 1.0mi	0.525011	0.514656	0.495247	0.475605
4k, 0.8mi	0.523428	0.516502	0.486613	0.470082
4k, 0.6mi	0.522052	0.498885	0.479042	0.461957

TABLE III
RESULTS OF CHI-SQUARE TEST.

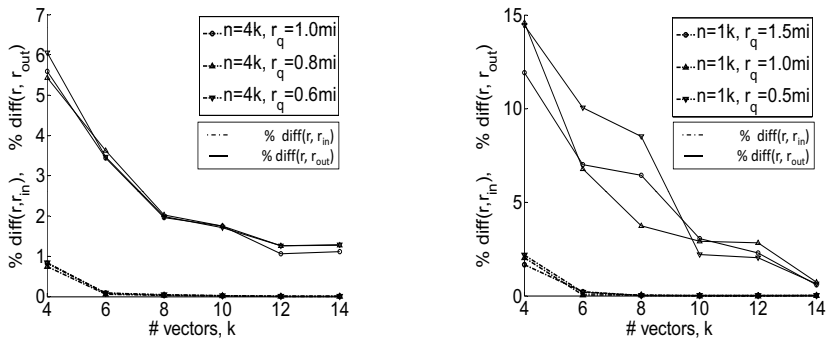
Figure 13 shows the percent of unconfirmed results as the number of vectors changes. The fraction of unconfirmed results decreases as the number of vectors increases. As a matter of fact, the query is actually evaluated less than 2% of the times, while in the rest it is answered through the heuristics.

Figure 14 depicts the percentage of difference between the actual radius of the SEC and the upper and lower bounds. While the lower bound is always very close to the actual radius the upper bound might be higher.

Finally, we examined the distribution of points around the circle when there is a circle of desired radius. We do Chi-square test of the angular position of the points near the perimeter of the circle. These are the points in the set $\mathcal{S}'$. The null hypothesis for the Chi-square test is "Points are uniformly distributed around the circle". Table III shows the p-values of Chi-square test. A p-value of 0.05 or less would mean that there is a significant difference between the observed distribution and the theoretical (uniform) distribution. Its equivalent to say that with 95% confidence interval a significant difference is observed. In that case we could reject the null hypothesis. As the experimental results show, p-values are around $0.5 - 0.6$, i.e., we cannot reject the null hypothesis with 95% confidence interval.

VII. CONCLUSIONS

This paper considers a novel problem for moving objects: real time identification of moving objects that are staying within a certain distance from a (unspecified) dwell region, for at least a certain duration. The main contribution of our work is to propose online method to evaluate the decision query in logarithmic time. Our proposed methods lend to approximating the radius of the smallest enclosing circle of a given subtrajectory within user defined arbitrary factor and efficient approximation of the dwell region. Our experimental evaluation shows that our algorithm is capable of evaluating the query condition on hundreds of thousands of moving objects per second on average.



(a)GeoLife Data (b)ElkDeer Data

Fig. 14. Comparison between $r_q(= r), r_{\lfloor \mathrm{SEC}_{\mathcal{S}} \rfloor}(= r_{in})$ and $r_{\lceil \mathrm{SEC}_{\mathcal{S}} \rceil}(= r_{out})$.

VIII. Acknowledgments

We would like to thank Neal E. Young and Claire Mathieu for their useful suggestion and comments.

References

[1] D. Fortin, H. L. Beyer, M. S. Boyce, D. W. Smith, T. Duchesne, and J. S. Mao, "Wolves influence elk movements: behavior shapes a trophic cascade in yellowstone national park," in *Ecological Society of America*, vol. 86, 2005, pp. 1320–1330.

[2] M. Buchin, A. Driemel, M. van Kreveldz, and V. Sacristn, "An algorithmic framework for segmenting trajectories based on spatio-temporal criteria," in *SIGSPATIAL*, November 2010.

[3] M. R. Vieira, P. Bakalov, and V. Tsotras, "Querying trajectories using flexible patterns," in *EDBT*, 2010, pp. 406–417.

[4] J.-G. Lee, J. Han, and K.-Y. Whang, "Trajectory clustering: A partition-and-group framework," in *ACM SIGMOD*, 2007, pp. 593–604.

[5] X. Cao, G. Cong, and C. S. Jensen, "Mining significant semantic locations from gps trajectory," in *VLDB*, 2010, pp. 1009–1020.

[6] Y. Zheng, L. Zhang, X. Xie, and W.-Y. Ma, "Mining interesting locations and travel sequences from gps trajectories," in *WWW*, 2009.

[7] M. R. Uddin, C. V. Ravishankar, and V. J. Tsotras, "Finding regions of interest from trajectory data," in *MDM*, 2011.

[8] M. R. Vieira, P. Bakalov, and V. Tsotras, "On-line discovery of flock patterns in spatio-temporal data," in *ACM GIS*, 2009, pp. 286–295.

[9] H. Jeung, M. L. Yiu, X. Zhou, C. S. Jensen, and H. T. Shen, "Discovery of convoys in trajectory databases," in *PVLDB*, vol. 1, no. 1, 2008, pp. 1068–1080.

[10] J. Ni and C. V. Ravishankar, "Pointwise-dense region queries in spatio-temporal databases," in *IEEE ICDE*, 2007, pp. 1066–1075.

[11] N. Megiddo, "Linear-time algorithms for linear programming in $\mathbb{R}^3$ and related problems," in *FOCS*, Nov 1982, pp. 329 – 338.

[12] K. L. Clarkson, "Las vegas algorithms for linear and integer programming when the dimension is small." in *JACM*, vol. 42, no. 2, March 1995.

[13] M. E. Dyer and A. M. Frieze, "A randomized algorithm for fixed-dimensional linear programming." in *Mathematical Programming*, vol. 44, no. 1-3, May 1989.

[14] R. Seidel, "Linear programming and convex hulls made easy." in *sixth annual Symposium on Computational Geometry*, 1990.

[15] E. Welzl, "Smallest enclosing disks (balls and ellipsoids)," in *New Results and New Trends in Computer Science, Lecture Notes in Computer Science*, vol. 555, 1991, pp. 359–370.

[16] M. A. Cheema, L. Brankovic, X. Lin, W. Zhang, and W. Wang, "Multi-guarded safe zone: An effective technique to monitor moving circular range queries," in *ICDE*, March 2010, pp. 189 – 200.

[17] A. Guttman, "R-trees: A dynamic index structure for spatial searching," in *ACM SIGMOD*, 1984, pp. 47–57.

[18] http://research.microsoft.com/en-us/projects/geolife/.

[19] http://www.fs.fed.us/pnw/starkey/.