Dwell Regions: Generalized Stay Regions For Streaming and Archival Trajectory Data

REAZ UDDIN, University of California, Riverside
MEHNAZ TABASSUM MAHIN, University of California, Riverside
PAYAS RAJAN, University of California, Riverside
CHINYA V. RAVISHANKAR, University of California, Riverside
VASSILIS J. TSOTRAS, University of California, Riverside

A region $\mathcal{R}$ is a dwell region for a moving object O if, given a threshold distance r_q and duration τ_q, every point of $\mathcal{R}$ remains within distance r_q from O for at least time τ_q. Points within $\mathcal{R}$ are likely to be of interest to O, so identification of dwell regions has applications such as monitoring and surveillance. We first present a logarithmic-time online algorithm to find dwell regions in an incoming stream of object positions. Our method maintains the upper and lower bounds for the radius of the smallest circle enclosing the object positions, thereby greatly reducing the number of trajectory points needed to evaluate the query. It approximates the radius of the smallest circle enclosing a given subtrajectory within an arbitrarily small user-defined factor, and is also able to efficiently answer decision queries asking whether or not a dwell region exists. For the offline version of the dwell region problem, we first extend our online approach to develop the ρ-Index, which indexes subtrajectories using query radius ranges. We then refine this approach to obtain the τ-Index, which indexes subtrajectories using both query radius ranges and dwell durations. Our experiments using both real-world and synthetic datasets show that the online approach can scale up to hundreds of thousands of moving objects. For archived trajectories, our indexing approaches speed up queries by many orders of magnitude.

CCS Concepts: • **Information systems → Geographic information systems**; **Data streaming**.

Additional Key Words and Phrases: Spatio-temporal databases, Stay regions, Smallest enclosing circle, Regions of interest, Trajectories.

1 INTRODUCTION

The widespread use of GPS devices has made position data readily available for millions of moving objects. Such data could be real-time streams of moving object locations, or archived trajectory collections. Queries on such data are of great importance in many applications, including monitoring and advertising. In this paper, we revisit the *dwell region* query, introduced in [47] for identifying regions of interest, and show how to address it more efficiently in both online (streaming) and offline (archived) environments. These two cases pose different challenges. In streaming environments, object positions change rapidly, so online dwell region queries must be answered in real-time. In the offline case, the size of archival datasets is a challenge, but preprocessing can be used to improve query times.

Authors' addresses: Reaz Uddin, uddinm@cs.ucr.edu, University of California, Riverside; Mehnaz Tabassum Mahin, mehnaztabassum.mahin@email.ucr.edu, University of California, Riverside; Payas Rajan, payas.rajan@email.ucr.edu, University of California, Riverside; Chinya V. Ravishankar, ravi@cs.ucr.edu, University of California, Riverside; Vassilis J. Tsotras, tsotras@cs.ucr.edu, University of California, Riverside.

2374-0353/2022/6-ART
https://doi.org/10.1145/3543850

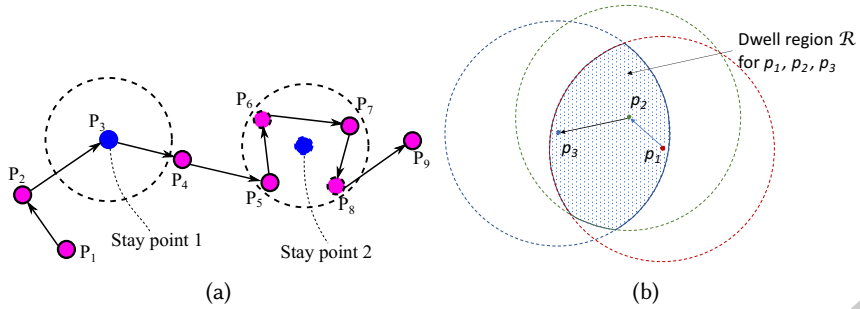


Fig. 1. (a) Two stay points defined by centroids of $\{p_3\}$ and $\{p_5, p_6, p_7, p_8\}$, respectively, and bounded by δ_q and τ_q, (b) A dwell region $\mathcal{R}$ defined by S. All points in $\mathcal{R}$ remain within distance r_q from p_1, p_2, p_3 for at least time τ_q.

1.1 Stay Points and Dwell Regions

Earlier works [26, 62, 63] have considered the problem of identifying *stay points* in a trajectory dataset. The terms "stay point" and "stay region" refer to parts of a trajectory in which the object travels no more than distance δ in a specified time τ. More precisely, a set S of consecutive trajectory locations constitutes a *stay region* if the object covers the locations in S in no more time than τ, and no pair of these locations is farther apart than δ. This stay region can be of any shape depending on the trajectory. [11, 36] identify *stay regions* with uncertain boundaries and arbitrary shapes from trajectories. A *stay point* is a geographic representation of the stay region, in some work [26, 33], the centroid of the set of points S is marked as the stay point. Figure 1a shows two types of stay points referred in the literature.

Stay points define locations or regions of natural interest for mobile users or other objects, and the problem of identifying them is important for a variety of reasons. Some examples appear in [41]: stay points are useful for large-scale population and context-aware applications, including city planning, transportation, and communication infrastructure. Stay points often reveal living patterns of populations, and may have public health or medical applications.

1.1.1 Dwell Regions. In this paper, we focus on identifying dwell regions that present a much more difficult technical challenge. Instead of simply considering points along the *trajectory* that remain within a distance δ of each other for at least time τ, we seek the set of points in the *region* which are within a distance r_q from the trajectory points for at least time τ_q.

That is, a region $\mathcal{R} \subset \mathbb{R}^2$ is a dwell region for a moving object O if, given a query radius r_q and time duration τ_q, O remains within distance r_q from every point in $\mathcal{R}$ for at least time τ_q. Let us assume that in the duration τ_q, object O moves through n positions in 2-dimensional space. Let $\mathcal{S}$ be the set of these positions. Computing $\mathcal{R}$ then reduces to computing the smallest enclosing circle (SEC) of the points in $\mathcal{S}$ denoted as $C_{\mathcal{S}}$ [32]. Computing $\mathcal{R}$ in real time is a hard problem. We therefore begin by examining how to efficiently compute $C_{\mathcal{S}}$, and then propose approximate methods to compute the dwell region query.

Figure 1 makes clear the difference between stay points and dwell regions. Figure 2 illustrates the dwell region identification problem, and how we can identify the dwell region from a set of points $\mathcal{S} = \{p_1, p_2, \ldots, p_9\}$. Let's assume that for a given query radius r_q and time duration τ_q, points $\{p_5, p_6, p_7, p_8, p_9\}$ remain within distance r_q from every point of $\mathcal{R}$ for at least τ_q time. We can identify $\mathcal{R}$ from the intersections of the circles centered at points $\{p_5, p_6, p_7, p_8, p_9\}$. Instead of considering all of these 5 points, if we choose p_5, p_6, p_9, that will suffice to identify $\mathcal{R}$. Note that p_5, p_6, p_9 are on the boundary of $C_{\mathcal{S}}$, so the dwell region identification problem reduces to computing $C_{\mathcal{S}}$. Also, the trajectory points defining a dwell region can be inside the dwell region (shaded region in Figure 1b) or outside of the dwell region (Figure 2).

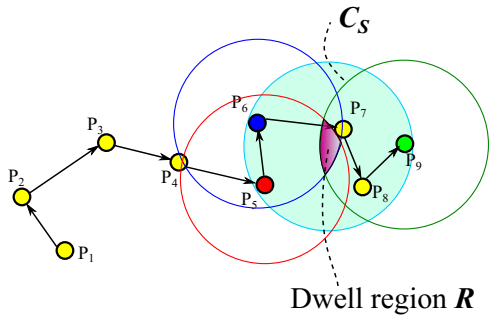


Fig. 2. An example of identifying a dwell region $\mathcal{R}$ from a set of points $\mathcal{S} = \{p_1, p_2, \ldots, p_9\}$ and a given query (r_q, τ_q).

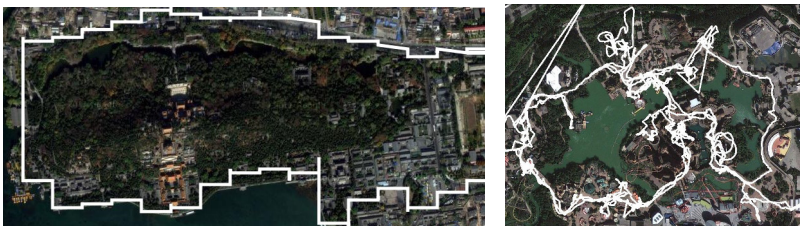
This problem has many real life applications. In security applications, the object O may represent a threat potent at range r_q, so the region $\mathcal{R}$ may contain potential targets for O. Similarly, O may be conducting surveillance on the region $\mathcal{R}$, or communicating with objects within range r_q. 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. Animals can be expected to remain close to their dens or nesting sites for extended periods. Identifying such behaviors can be revealing, and very important to ecosystem researchers [14]. The behaviors we consider in this paper include both going around a region, as in Figure 3a, or random movement in a certain enclosed region, as in Figure 3b.

A possible application of dwell region queries in indoor environments [22] is in suggesting frequently visited places to users. For example, in a museum or a mall, a dwell region of several trajectories indicates a region that is of interest to several patrons. Extracting such dwell regions from archived trajectories and recommending them to other visitors might help improve overall user experience. Identifying dwell regions from archival trajectories can be useful to identify popular regions of interest and in activity or location recommendation systems [58, 62]. However, depending on the trajectory dataset size, these applications can take a long time to find dwell regions which is undesirable. Thus it requires an efficient method to precompute dwell regions and access them while query processing.

Our work also has applications in real time trajectory simplification based on spatio-temporal criteria. One such criterion [4] 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 scenarios, it is desirable to do this simplification in real time, without having to store the data.

1.2 Our Approach and Contributions

Given a set of points $\mathcal{S}$, we denote the smallest enclosing circle for $\mathcal{S}$ by $C_{\mathcal{S}}$. For the online scenario, we assume that we receive regular position updates from each moving object, and will maintain a *streaming window* (or just



(a) (b)

Fig. 3. Behaviors considered in this paper.

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. A window corresponds to a set of location points $\mathcal{S}$. In the following we use window and $\mathcal{S}$ interchangeably.

Given a window $\mathcal{S}$, our approach approximates the smallest enclosing circle $C_{\mathcal{S}}$ of the points in $\mathcal{S}$ 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 $C_{\mathcal{S}}$. 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, where n is the window size $|\mathcal{S}|$. We can compute the upper and lower bounds in time $O(k)$.

Typically, we are able to decide whether $C_{\mathcal{S}}$ 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 $C_{\mathcal{S}}$. Computing $C_{\mathcal{S}}$ only gives the center of the circle, not the complete region $\mathcal{R}$. We then propose a method for quickly approximating the region $\mathcal{R}$. In our experiments with real trajectory datasets, the online approach can handle hundreds of thousands of moving objects. In the archival scenario, trajectories are already available, and can thus be preprocessed. We will adapt the idea of using $C_{\mathcal{S}}$ to build efficient indexes for the archived trajectories, that can speed up dwell region queries by many orders of magnitude.

1.2.1 Our Contributions. In previous work [47], we introduced the problem of identifying dwell regions, but considered only the online version of the problem. In this paper, we also examine the offline version of the problem (on archival data), and show how preprocessing can speed up queries. We address the case of archival data in two stages. First, we develop a natural extension of the ideas we developed for the online case, and define a one-dimensional index called the ρ-Index, which indexes subtrajectories using query radius ranges, i.e., it considers only the distance parameter of dwell region queries. Next, based on an evaluation of the ρ-Index and an analysis of its shortcomings, we develop the two-dimensional τ-Index, which indexes subtrajectories using both query radius ranges and dwell durations, that is, it considers both the distance and time parameters of dwell region queries. We validate the efficiency of our approach on archival data using the case without indexes as the baseline. This being the first work to address this problem, our choice of baseline is reasonable.

We build the ρ-Index by partitioning query radius ranges, and precomputing dwell regions for some fixed radius values r_i. For each such r_i, we use the online algorithm to identify the maximal subtrajectories that can be enclosed within a circle with the radius r_i. To build the τ-Index, we partition both query radius and duration ranges, and use a modified online algorithm that indexes subtrajectories with duration τ and radius r, where, $\tau_i \leq \tau \leq \tau_{i+1}$ and $r_{j-1} < r \leq r_j$. These indexes can be used to answer dwell region queries.

Since the ρ-Index indexes subtrajectories corresponding to a predetermined set of query radii, it does not directly yield dwell regions for any arbitrary query radius. Whereas the τ-Index indexes subtrajectories corresponding to both radius and duration ranges, the result set of dwell regions can be built directly using the τ-Index. Our experimental results show that both ρ-Index and τ-Index can offer several orders of magnitude speedup in query evaluation time over the non-indexed dwell region queries. However, since the τ-Index consists of partitions along two dimensions, it has greater pruning power, and therefore can answer queries several times faster than the simpler ρ-Index.

Our contributions are as follows:

- We show how to maintain an approximation of $C_{\mathcal{S}}$ in logarithmic update time.
- We devise a method to compute $C_{\mathcal{S}}$ exactly using a few points, based on our data structures. We also show how to quickly obtain a good approximation of the dwell region $\mathcal{R}$.
- We identify upper and lower bounds for the $C_{\mathcal{S}}$ radius, and show that the radius of $C_{\mathcal{S}}$ is within a factor of $(1 + O(\frac{1}{k^2}))$ of the lower bound, for user defined k.

- We extend the online method to dwell region queries on archival data. We present two novel indexes: ρ-Index and τ-Index, which speed up queries by using precomputed results.
- We present extensive experimental results for both online and offline versions of the problem, the code for which is publicly available [1].

The rest of the paper is organized as follows: Section 2 provides the definitions and background while Section 3 describes related work. Our proposed methods and data structures are described in Section 4. Sections 5 and 6 present the preprocessing and query evaluation methods for both online and offline versions respectively. Section 7 presents the experimental results and Section 8 concludes the paper.

2 BACKGROUND

We assume the following environment of objects moving in a 2-dimensional space. Every moving object has a unique object *oid* and sends its position updates at certain regular intervals. A position update contains the object *oid*, its spatial coordinates $x_i, y_i \in \mathbb{R}^2$, and a timestamp $t_i \in \mathbb{R}^+$. A ***trajectory*** is created by subsequent position updates from a given object and is a finite sequence of triples (x_i, y_i, t_i) with $t_i < t_{i+1}$ for $i = 0, 1, \ldots$ A ***trajectory segment*** is the 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. A ***subtrajectory*** of length m of a trajectory $T = (x_0, y_0, t_0), \ldots, (x_p, y_p, t_p)$, 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 $i \geq 0, m < p$. 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 *dwell region query* (r_q, τ_q) finds region $\mathcal{R}$, each point of which is within a distance r_q from any point on the trajectory of moving object for time at least τ_q. A *decision query* (r_q, τ_q) asks whether the positions of a given object during the window of size τ_q, fall within distance r_q from any point in $\mathbb{R}^2$. This decision query returns a Boolean value and the center of the smallest enclosing circle. Decision queries are important for applications like trajectory simplification [4].

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 $\mathcal{S}$ are within distance 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 $C_{\mathcal{S}}$, the smallest enclosing circle for $\mathcal{S}$. Finding $C_{\mathcal{S}}$ suffices to answer decision queries, but not dwell region queries.

A naive approach to maintaining the overlap between circles is to recompute their intersections whenever a point is inserted into or deleted from the window. However, to the best of our knowledge, there exists no efficient online algorithm to maintain intersection of circles. Additions of a point 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 smallest enclosing circle 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 smallest enclosing circle by tuning k.

Computing $C_{\mathcal{S}}$ gives only the center of $C_{\mathcal{S}}$, 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 $C_{\mathcal{S}}$ exceeds r_q. In this case, we do not compute the intersecting region. When the upper bound is

[1] https://github.com/PayasR/DwellRegions-open

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, $B \subset \mathcal{S}$.
3: $\mathcal{S}' = B, B' = B, D = C_B$ ▷ $\mathcal{S}'$: set of points seen so far, B': current basis of $\mathcal{S}'$.
4: **for** each $p_i \in (\mathcal{S} - \mathcal{S}')$ **do**
5: $\quad \mathcal{S}' = \mathcal{S}' \cup \{p_i\}$
6: $\quad$ **if** p_i is not inside D **then**
7: $\quad\quad B' = \text{MinDisk}(\mathcal{S}', \text{Basis}(B' \cup \{p_i\}))$
8: $\quad\quad D = C_{\mathcal{S}'}$
9: $\quad$ **end if**
10: **end for**
11: **return** B'

Algorithm 2 Basis (A)

1: A : set of 4 points $\{p_1, p_2, p_3, p_4\}$
2: A' = all possible subsets of size 3 points of A
3: **for** each set of 3 points A'_i in A' **do**
4: $\quad c$ = circle defined by points of A'_i
5: $\quad$ **if** c contains all points in A **then**
6: $\quad\quad$ **return** A'_i as the basis
7: $\quad$ **end if**
8: **end for**

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 4.5.

2.1 Computing the Smallest Enclosing Circle C_S

The first deterministic linear time algorithm for the smallest enclosing circle appeared in [32]. Several improvements were presented in [10, 12, 43]. 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 [52] 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 C_S for a small set of points $\mathcal{S}$.

Algorithm 1 gives the pseudo code for Welzl's algorithm. Given a set of two dimensional points $\mathcal{S} = \{p_1, p_2, \ldots, p_n\}$, there is a "basis" set B of at most three points that determines C_S, i.e., $C_B = C_S$. Computation of the basis of a set of at most 4 points can be done in constant time (Algorithm 2). Three points are randomly selected from $\mathcal{S}$ as the initial basis, i.e. $B = \{p_i, p_j, p_k\}$, $i, j, k \leq n$, and the initial circle is $D = C_B$. The algorithm is started with the call MinDisk ($\mathcal{S}, B$). The remaining points in $\mathcal{S} - B$ are tested one by one whether they are inside D. Let $\mathcal{S}'$ be the set of points seen so far. If a point $p_i \in (\mathcal{S} - \mathcal{S}')$ is inside D, then D is the smallest enclosing circle of $\mathcal{S}'$. Otherwise, a recursive call (Line 7) is made to compute $C_{\mathcal{S}'}$. This time the basis is Basis($B' \cup \{p_i\}$). When this recursive call returns we have the basis and the smallest enclosing circle of points seen so far.

3 RELATED WORK

Retrieving semantic and/or activity information from trajectories has attracted much research attention, as it has applications in many domains. A number of works consider finding regions of interest (ROIs) such as restaurants, museums, or parks, and variations thereof. In [1, 53], information on ROIs is given in a relational database, and a join operation between trajectory and ROI relations is performed to evaluate activity sequence queries. It is assumed that the querying application will specify a finite set of pairs (Δ, τ) of interesting geographic regions Δ and durations τ. If a trajectory spends at least τ duration in a specified region, then the portion of the trajectory inside region Δ is considered a *stop area* in that trajectory.

Other work on discovering ROIs include [5, 26, 37, 44, 46]. For example, [44] identifies an ROI as the cluster of object locations that is dense and remains closer to a given route. In [46], a region is considered as interesting if the objects have remained within the region for some time interval threshold. In [26, 62, 63], the notion of "stay point" is presented using a maximum distance threshold D_t and a minimum duration threshold T_t. Stay points are obtained by averaging the points of the trajectory, and may not themselves be on a trajectory (like stay point 2 in Figure 1a). However, reducing a subtrajectory to a particular point, possibly not on the trajectory, can lead to loss of information. For example, if density-based methods are applied on stay points to identify stay regions, they might generate false negatives.

In [37], ROIs are discovered for each individual trajectory instead of considering all trajectories and identifying commonly interesting places. The DBSCAN method [13] is modified so that parts of a particular trajectory within a small region and with sufficient stay duration in that region will be considered as clusters (ROIs). Related are also works on density-based queries over moving object databases. A spatial area is dense if the number of moving objects it contains is above some threshold [18, 34].

There are also works that consider identifying interesting patterns from trajectory data. For example, [6, 40, 54] consider various similarity queries on trajectories. An approach to mine common sequence of locations, visited with similar travel times between them appears in [15]. An example is *Railway Station* $\xrightarrow{15\ min}$ *Book Store* $\xrightarrow{30\ min}$ *University*, meaning that *Railway Station* to *Book Store* to *University* is a common travel sequence with travel time between them 15 min and 30 min respectively. Similarly, [63] finds travel sequences for a user based on their location history and the number of visits. There has also been work on annotating trajectories with semantic information. A survey appears in [38]. A semantic trajectory can be a sequence of Points-of-interest (POIs) with both location and text information which can then be used to find similar trajectories [9]. [57] identifies trajectories from a semantic trajectory database that contain most relevant keywords to the query and require minimum distance to travel. [22] introduces a semantic indoor trajectory model. Work in [50] considers flexible pattern queries on trajectories. Trajectory data is used to find better routes in [30, 51, 56]. In [7, 55], trajectories are used to understand city dynamics, causes of traffic jams, etc.

Research has considered using user activities to identify interesting behavior. Work on single user activity includes inferring transportation mode [61], periodic activity recognition [28], route prediction [2, 23], etc. In [29, 39, 61], the authors consider users' locations to recognize their activities and identify their transportation routines. Activity recommendation from trajectory data [58, 62] is another popular research trend.

There has also been research on identifying collaborative behaviors, such as flock patterns [49], convoy detection [19], swarm identification [27], assemblies [48], or rendezvous regions [45]. The work in [34] uses the current location of moving objects to identify the density of moving objects, while [24, 25] consider trajectory clustering. In [35], clustering on big trajectory data is studied to find the hotspots in a parallel setting while [20, 21] considers detecting hotspots from indoor trajectory data. Here, we consider finding dwell regions for individual trajectories.

Related is also the work in [8] that proposes several heuristics for computing circle intersections. Unfortunately, these heuristics are not useful in environments where points are added/deleted dynamically. In particular, an

R-tree [17] maintains all the *static sites* and circle intersections are computed 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 [8] unnecessary.

In previous work [47], we addressed the dwell region problem in streaming environments. In this paper, along with the streaming environment, we consider the archival trajectories and propose index structures to identify dwell regions in the archival setting.

4 DATA STRUCTURES & ALGORITHMS FOR COMPUTING C_S

We proceed with the presentation of our algorithms and data structures for approximating the smallest enclosing circle C_S. We also show how to bound the radius of C_S above and below by using circles constructed from the minimum bounding polygon for S. Some of the notable symbols used in this section to discuss the algorithm for computing C_S are listed in Table 1.

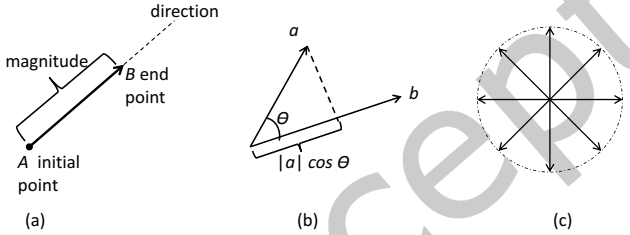


Fig. 4. (a) A Euclidean vector, (b) Scalar projection of $\vec{a}$ onto $\vec{b}$, (c) Eight uniformly spaced vectors in a circle.

An n-dimensional *vector* $\vec{v}$ is an n-tuple $(v_1, v_2, \ldots, v_n)$, where v_i is its component along the i^{th} axis. The *magnitude* of vector $\vec{v}$ is $|\vec{v}| = \sqrt{\sum_{i=1}^{n} v_i^2}$. The *dot product* of two vectors $\vec{a}$ and $\vec{b}$ is $\vec{a} \cdot \vec{b} = \sum_{i=1}^{N} a_i b_i = |\vec{a}||\vec{b}| \cos\theta$, where θ is the angle between $\vec{a}$ and $\vec{b}$. If $\vec{b}$ is a unit vector then the dot product $\vec{a} \cdot \vec{b}$ is the component of $\vec{a}$ in the direction of $\vec{b}$, also called the projection of $\vec{a}$ onto $\vec{b}$.

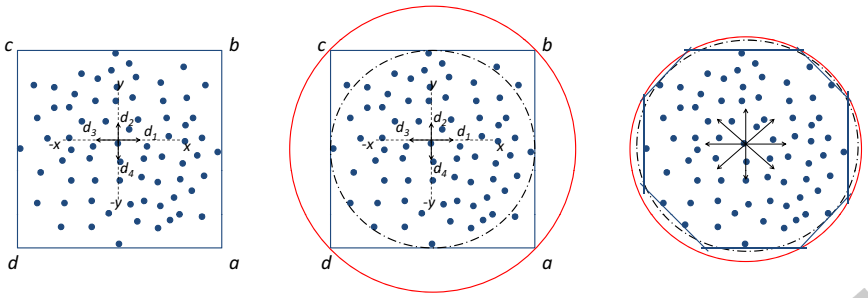
If k vectors are uniformly spaced around a circle, the angle between any two adjacent vectors is $\frac{2\pi}{k}$. Figure 4 shows (a) a Euclidean vector, (b) scalar projection of $\vec{a}$ onto $\vec{b}$ and (c) eight uniformly spaced vectors around a circle. In a Euclidean space each side of a straight 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 $H = \{(x, y) : ax + by \leq c\}$.

4.1 Minimum Bounding Polygons and Frontiers

Figure 5a shows a set of points and its minimum bounding rectangle (MBR), suggestive of the maximum extents of a set of points in S. In the 2-D case, we can construct an MBR for 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 a direction orthogonal to $\vec{d_i}$, in from infinity towards S. We stop when each e_i touches a point $p_i \in S$. The lines e_i form the edges of the MBR.

We can generalize, and 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 5c shows eight uniformly spaced vectors and the bounding convex octagon they define. The lines e_i are swept inwards from infinity until they touch points $p_i \in S$.

We denote the k-polygon bounding S by $\mathcal{P}_S^k$. If $\mathcal{V}_S^k$ is the set of vertices of $\mathcal{P}_S^k$, then $\mathcal{V}_S^k \not\subset S$.



(a) (b) (c)

Fig. 5. (a) A set $\mathcal{S}$ of points and its bounding k-polygon, $k = 4$. The polygon has vertices $\mathcal{V}_{\mathcal{S}}^k = \{a, b, c, d\}$. The frontier $\mathcal{F}_{\mathcal{S}}^k$ is the set of points touching the polygon's edges. (b) The in-circle $C_{\mathcal{S}}^{in}$ is the smallest circle enclosing $\mathcal{F}_{\mathcal{S}}^k$, and the out-circle $C_{\mathcal{S}}^{out}$ is the smallest circle enclosing $\mathcal{V}_{\mathcal{S}}^k$. (c) The effect of using $k = 8$.

Definition 1. *(Frontier) Given a minimal bounding k-polygon $\mathcal{P}_{\mathcal{S}}^k$, the frontier of $\mathcal{S}$ is the set $\mathcal{F}_{\mathcal{S}}^k$ of points $p_i \in \mathcal{S}$ lying on the edges of $\mathcal{P}_{\mathcal{S}}^k$.*

We note that $\mathcal{V}_{\mathcal{S}}^k \not\subset \mathcal{S}$, but $\mathcal{F}_{\mathcal{S}}^k \subset \mathcal{S}$ by definition.

4.2 Using $C_{\mathcal{S}}$ and $\mathcal{P}_{\mathcal{S}}^k$ in Queries

Let $r_{\mathcal{S}}$ be the radius of $C_{\mathcal{S}}$. We can now get upper and lower bounds for $r_{\mathcal{S}}$ as follows. The smallest circle $C_{\mathcal{V}_{\mathcal{S}}^k}$ enclosing the set $\mathcal{V}_{\mathcal{S}}^k$ of vertices of $\mathcal{P}_{\mathcal{S}}^k$ is guaranteed to contain all the points of $\mathcal{S}$, and yields an overestimate for $C_{\mathcal{S}}$ (see Figure 5b). Similarly, $C_{\mathcal{F}_{\mathcal{S}}^k}$, the smallest circle enclosing all the frontier points of $\mathcal{S}$ yields an underestimate for $C_{\mathcal{S}}$.

Definition 2. *(In- and out-circles) Let the minimum bounding k-polygon $\mathcal{P}_{\mathcal{S}}^k$ for $\mathcal{S}$ have vertex set $\mathcal{V}_{\mathcal{S}}^k$ and frontier $\mathcal{F}_{\mathcal{S}}^k$. The in-circle for $\mathcal{S}$ is $C_{\mathcal{S}}^{in} \stackrel{def}{=} C_{\mathcal{F}_{\mathcal{S}}^k}$ and the out-circle for $\mathcal{S}$ is $C_{\mathcal{S}}^{out} \stackrel{def}{=} C_{\mathcal{V}_{\mathcal{S}}^k}$. (See Figure 5.)*

Let $r_{\mathcal{S}}^{in}$ and $r_{\mathcal{S}}^{out}$ be the radii of $C_{\mathcal{S}}^{in}$, and $C_{\mathcal{S}}^{out}$, respectively, so that $r_{\mathcal{S}}^{in} \le r_{\mathcal{S}} \le r_{\mathcal{S}}^{out}$. We will show that the distance from the center of $C_{\mathcal{S}}^{in}$ to any point in $\mathcal{S}$ is at most $r_{\mathcal{S}}^{in}\left(1 + O\left(k^{-2}\right)\right)$. The values $r_{\mathcal{S}}^{in}$ and $r_{\mathcal{S}}^{out}$ are also very useful in optimizing decision queries, since we can immediately respond YES when $r_q > r_{\mathcal{S}}^{out}$ and NO if $r_q < r_{\mathcal{S}}^{in}$.

4.3 Constructing the Smallest Enclosing Circle $C_{\mathcal{S}}$

If neither $r_q > r_{\mathcal{S}}^{out}$ nor $r_q < r_{\mathcal{S}}^{in}$ holds, we must construct $C_{\mathcal{S}}$ explicitly. We now show how to construct this with minimal overhead.

Definition 3. *(Convex hull) The convex hull $\mathcal{H}_{\mathcal{S}}$ of a given set $\mathcal{S}$ is the minimal convex region enclosing $\mathcal{S}$.*

We denote the set of points forming the vertices of $\mathcal{H}_{\mathcal{S}}$ by $\mathcal{V}_{\mathcal{H}_{\mathcal{S}}} \subseteq \mathcal{S}$.

Lemma 4.1. *If $\mathcal{H}_{\mathcal{S}}$ is the convex hull for a set $\mathcal{S}$, then $C_{\mathcal{V}_{\mathcal{H}_{\mathcal{S}}}} = C_{\mathcal{S}}$.*

Proof: $\mathcal{H}_{\mathcal{S}}$ is a convex region enclosing all points of $\mathcal{S}$. The minimal circle $C_{\mathcal{V}_{\mathcal{H}_{\mathcal{S}}}}$ enclosing $\mathcal{H}_{\mathcal{S}}$ encloses this convex region, and therefore all points in $\mathcal{S}$. □

Symbol	Meaning	Symbol	Meaning
$\mathcal{S}$	A set of 2D points	$\mathcal{H}_\mathcal{S}$	Minimal convex region enclosing $\mathcal{S}$
$\mathcal{P}_\mathcal{S}^k$	Minimal k-polygon bounding $\mathcal{S}$	$\mathcal{F}_\mathcal{S}^k$	Frontier: Points of $\mathcal{S}$ on edges of $\mathcal{P}_\mathcal{S}^k$
$\mathcal{V}_\mathcal{S}^k$	Vertices of $\mathcal{P}_\mathcal{S}^k$	h_i	Heap corresponding to unit vector d_i
$C_\mathcal{S}$	Smallest circle enclosing $\mathcal{S}$	$r_\mathcal{S}$	Radius of $C_\mathcal{S}$
$C_\mathcal{S}^{in}$	In-circle: smallest circle enclosing $\mathcal{F}_\mathcal{S}^k$	$r_\mathcal{S}^{in}$	Radius of $C_\mathcal{S}^{in}$
$C_\mathcal{S}^{out}$	Out-circle: smallest circle enclosing $\mathcal{V}_\mathcal{S}^k$	$r_\mathcal{S}^{out}$	Radius of $C_\mathcal{S}^{out}$

Table 1. Symbols used in this paper.

Clearly, $\mathcal{H}_\mathcal{S} \subseteq \mathcal{P}_\mathcal{S}^k$, since $\mathcal{P}_\mathcal{S}^k$ may include dead space beyond $\mathcal{H}_\mathcal{S}$. Our approach is to identify a subset $\mathcal{S}' \subseteq \mathcal{S}$ such that $\mathcal{H}_\mathcal{S} \subseteq \mathcal{S}'$. We will find $C_{\mathcal{S}'}$, which will give us $C_{\mathcal{V}_{\mathcal{H}_\mathcal{S}}}$, and consequently $C_\mathcal{S}$. This approach is efficient, since we expect $\mathcal{S}'$ to be much smaller than $\mathcal{S}$.

4.4 The Algorithm

The quality of the algorithm depends on k, the number of directional vectors used. The algorithm maintains the frontier point in each direction. 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 $\mathcal{P}_\mathcal{S}^k$, 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 $\vec{p}$ in the current set that maximizes $\vec{d}_i \cdot \vec{p}$, which will be the point $\vec{p}$ 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. 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 $O(\log n)$ time per unit vector i.e. $O(k \log n)$ time for k vectors. The set $\mathcal{F}_\mathcal{S}^k$ consists of the points at the heap roots. The set $\mathcal{V}_\mathcal{S}^k$ is computed from the intersection of adjacent edges of $\mathcal{P}_\mathcal{S}^k$.

Algorithm 3 BuildHeaps $(\mathcal{S}, H, d)$

```
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 ∀d_i ∈ d do
    for ∀p_j ∈ S do
        h_i[j] = d_i · p_j
    end for
    heapify(h_i)
end for
```

Algorithm 3 describes the initial processing. The dot product between each vector and each point is calculated in line 6. One heap is built with the values of dot products for each vector. The heapify operation at line 8 builds the heap, which takes $O(n \log n)$. The dot product calculation for each vector in lines 5—7 is $O(n)$. As a result, the complexity of this pre-processing 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 $\vec{p}_j$ and/or its dot product $\vec{d}_i \cdot \vec{p}_j$ with the vector $\vec{d}_i$ we need to know where is this value in the heap h_i. For example we need to access a point if it is to be deleted. Therefore 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 4 UpdateHeap ($\mathcal{S}, H, d, p$)

1: $\mathcal{S}$: streaming window, $\{\vec{p}_1, \vec{p}_2, \ldots, \vec{p}_n\}$.
2: H : k heaps, $\{h_1, h_2, \ldots, h_k\}$, one for each direction being tracked.
3: d : k directions, $\{\vec{d}_1, \vec{d}_2, \ldots, \vec{d}_k\}$.
4: $\vec{p}$: next point to be added in $\mathcal{S}$.
5: **for** $\forall \vec{d}_i \in d$ **do**
6: delete $LT[\vec{d}_i][\vec{p}_1]$ from h_i
7: $val = \vec{d}_i \cdot \vec{p}$
8: insert val into h_i
9: **end for**
10: delete $\vec{p}_1$ from $\mathcal{S}$
11: add $\vec{p}$ to the end of $\mathcal{S}$

After building the heaps we must update them when inserting and deleting points from $\mathcal{S}$; Algorithm 4 describes the update step. At each update step, a point $\vec{p}$ is added to the window $\mathcal{S}$ as the most recent point, while the least recent point, $\vec{p}_1$, is deleted from $\mathcal{S}$. This requires deleting the record for $\vec{p}_1$ from each heap (table LT is used for this purpose). Next, the dot product between $\vec{p}$ and each vector $\vec{d}_i$ is calculated and the result inserted in the corresponding heap. Since insertion and deletion in a heap take $O(\log n)$, the update is a $O(k \log n)$ operation.

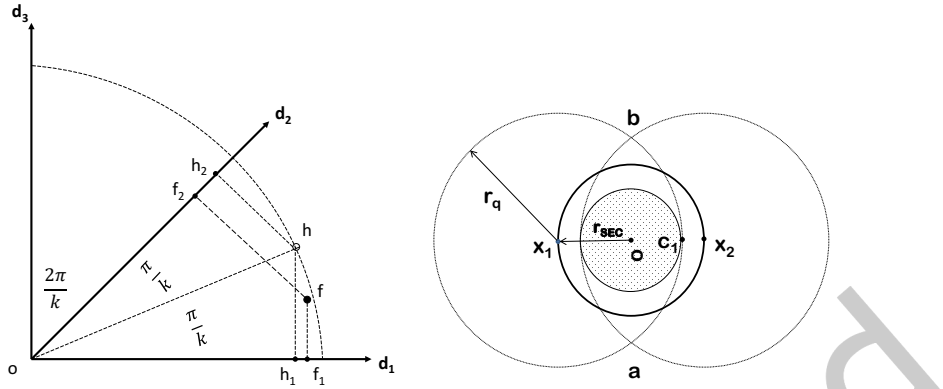
4.5 Determining $C_{\mathcal{S}}$

We have described the data structures used to maintain the upper and lower bounds and to approximate the radius of $C_{\mathcal{S}}$. 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, with heap h_i corresponding to the vector $\vec{d}_i$. The root of heap h_i is a frontier point, since it maximizes the extent of $\mathcal{P}_{\mathcal{S}}^k$ in $\vec{d}_i$'s direction. Recall that the bounding k-polygon $\mathcal{P}_{\mathcal{S}}^k$ for $\mathcal{S}$ has vertices $\mathcal{V}_{\mathcal{S}}^k$ and frontier points $\mathcal{F}_{\mathcal{S}}^k$.

To get $r_{\mathcal{S}}^{in}$ and $r_{\mathcal{S}}^{out}$, we use Welzl's algorithm to compute $C_{\mathcal{V}_{\mathcal{S}}^k}$ and $C_{\mathcal{F}_{\mathcal{S}}^k}$ (see Definition 2). As already noted, $r_{\mathcal{S}}^{in}$ and $r_{\mathcal{S}}^{out}$ can be used to answer decision queries without actually computing $C_{\mathcal{S}}$. However, when $r_{\mathcal{S}}^{in}$ and $r_{\mathcal{S}}^{out}$ are insufficient for this purpose (i.e., when $r_q < r_{\mathcal{S}}^{out}$ and $r_q > r_{\mathcal{S}}^{in}$) we must compute $C_{\mathcal{S}}$.

4.5.1 Efficient Computation of $C_{\mathcal{S}}$. We now show how to compute $C_{\mathcal{S}}$ using only a small subset $\mathcal{S}' \subset \mathcal{S}$. Our central idea is to identify a set of points $\mathcal{S}'$ that includes all points on the convex hull $\mathcal{H}_{\mathcal{S}}$. In Figure 6a, let O be the center of $C_{\mathcal{S}}$, and consider the angular sector between vectors $\vec{d}_1$ and $\vec{d}_2$. Let $\vec{f}$ be the point in this sector with the maximal projection on to $\vec{d}_1$, so $\vec{f}$ is the frontier point in the direction of $\vec{d}_1$.



(a) (b)

Fig. 6. (a) Computing $\mathcal{S}'$. Point f is a frontier point in the direction of $\vec{d}_1$. Point h is farther from O than f, so it should lie on the convex hull. We include all points whose projections exceed $|Of_1| \cos\left(\frac{\pi}{k}\right)$ in $\mathcal{S}'$, and (b) The shaded region is $\mathcal{R}^-$.

However, $\vec{f}$ need not be the furthest point in this sector from O. As Figure 6a shows, there could be a non-frontier point $\vec{h}$ 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 $\vec{h}$ lies on the convex hull $\mathcal{H}_S$, 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 $\vec{h}$ in $\mathcal{S}'$. We first note that this situation arises because the angular distance of $\vec{f}$ from $\vec{d}_1$ is less than that of $\vec{h}$. (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 $\vec{f}$ on $\vec{d}_1$ will be largest when $\vec{f}$ 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 $\vec{h}$ on both $\vec{d}_1$ and $\vec{d}_2$, and maximizes the likelihood that point $\vec{h}$ will not be a frontier point.) Let $\vec{f}$ lie on $\vec{d}_1$ and rotate the line Of so that it coincides with the bisector Oh. The projection of $\vec{f}$ 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 $\vec{f}$ is, and remain candidates for $\mathcal{H}_S$. To get $\mathcal{S}'$, we proceed as follows:

(1) Place all frontier points $\vec{f}_i \in \mathcal{F}_S^k$ into $\mathcal{S}'$.
(2) Let $\vec{f}_i$ 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 4.2. $C_S = C_{S'}$.

Proof: Since the convex hull $\mathcal{H}_S$ is the maximal convex region enclosing $\mathcal{S}$, all frontier points $\vec{f}_i \in \mathcal{F}_S^k$ are in $\mathcal{H}_S$. Step 1 above places $\mathcal{F}_S^k$ into $\mathcal{S}'$. However, not all points in $\mathcal{H}_S$ are in $\mathcal{F}_S^k$. Convexity of $\mathcal{H}_S$ ensures that such points must be farther from the center of C_S than the frontier points. Step 2 above places all such points into $\mathcal{S}'$. □

By Theorem 4.2, we need consider only points in $\mathcal{S}'$, which is much smaller than $\mathcal{S}$. Then, we can compute the SEC with these smaller number of points in $\mathcal{S}'$ from Algorithm 1. As a result, decision queries now run much faster.

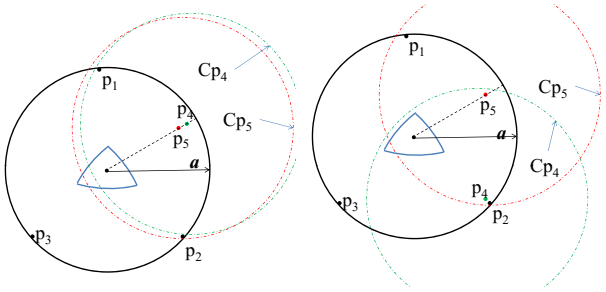


Fig. 7. Showing the importance of points near the boundary to compute the dwell region $\mathcal{R}$ when the are at (a) the same angular position (b) a different angular position.

5 ONLINE 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 $r_{\mathcal{S}} = r_q$, the circles centered at the points on the perimeter of $C_{\mathcal{S}}$ will share only its center. In that case, $\mathcal{R}$ will consist of only one point, namely, the center of $C_{\mathcal{S}}$.

Next, consider the case when $r_q \geq r_{\mathcal{S}}$. Let C_x be a circle of radius r_q centered at a point x on the circumference of $C_{\mathcal{S}}$. Consider the region $\mathcal{R}^- = \bigcap_x C_x$. (See shaded region in Figure 6b.) If we move the center of C_x along the circumference of $C_{\mathcal{S}}$, the intersection of the resulting circles will be a disk $\mathcal{R}^-$ of radius $\delta = r_q - r_{\mathcal{S}}$ centered at the center of $C_{\mathcal{S}}$.

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 the 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 $\mathcal{R}_m$, as the remaining circles are less likely to have an effect on $\mathcal{R}_m$.

Intuitively, points closer to the boundary of $C_{\mathcal{S}}$ 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 [8].

Figure 7a shows $C_{\mathcal{S}}$ for some 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 $C_{\mathcal{S}}$. 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, respectively. Consider two other points p_4, p_5 inside $C_{\mathcal{S}}$ lying on the same radial vector $\vec{a}$. Let p_5 be closer to the center of $C_{\mathcal{S}}$ 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}.

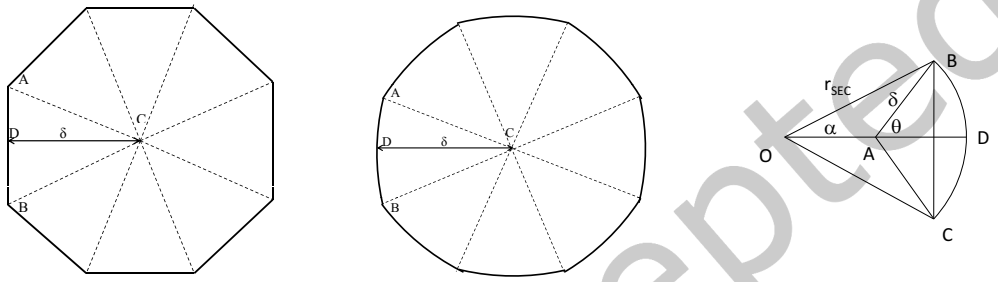
This illustration shows the importance of points closer to the boundary among the points that are at a same angular position in the circle. However, if p_4 and p_5 had different angular position then p_5 might have trimmed $\mathcal{R}_m$ more than p_4, as in Figure 7b. Nevertheless, if the points are uniformly distributed, considering them in the 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 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 $C_{\mathcal{S}}$ 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 6a) can be considered.

[8] 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, heuristics 2, 3, 5 do not need to be considered. 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 [8] 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 4.2.



(a) (b) (c)

Fig. 8. Replacing the bounding arcs of the dwell region with straight lines: (a) actual region, (b) bounding arcs replaced with straight lines; and (c) Approximating $\mathcal{R}^+$.

5.1 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 $\delta = (r_q - r_{\mathcal{S}})$. 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 8a shows $\mathcal{R}^+$ with eight bounding arcs. Each of the sectors in this figure is equivalent to the sector ABDC in Figure 8c. We obtain the area of this sector as follows.

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

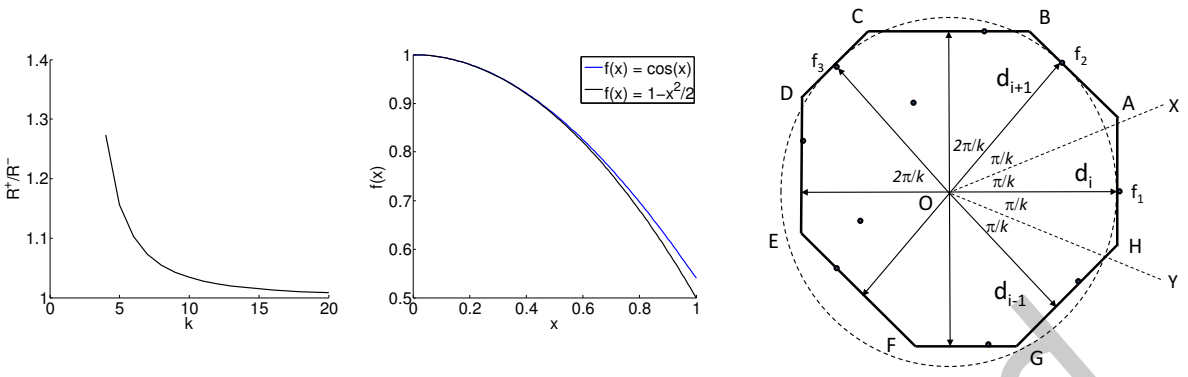
$$\alpha = \arcsin\left(\frac{|\mathrm{BC}|}{2r_{\mathcal{S}}}\right) = \arcsin\left(\frac{\delta\sin(\theta)}{r_{\mathcal{S}}}\right). \tag{1}$$

Standard formulas yield the area of the circular segment

$$\mathrm{BCDB} = \frac{(r_{\mathcal{S}})^2}{2}\left(2\alpha - \sin(2\alpha)\right).$$

Now, using elementary trigonometry and simplifying,

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



(a) (b) (c)

Fig. 9. Trigonometric functions, (a) The ratio $\frac{\mathcal{R}^+}{\mathcal{R}^-} = \frac{k \tan\theta}{\pi}$ as k changes, (b) Approximating $\cos x$ with $1 - \frac{x^2}{2!}$; and (c) Proving Theorem 5.1. $\mathcal{P}_{\mathcal{S}}^k$ = ABCDEFGH.

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

$$\frac{\mathcal{R}^+}{\mathcal{R}^-} = \frac{k\left(\frac{1}{4}\delta^2 \sin(2\theta) + \frac{(r_{\mathcal{S}})^2}{2}(2\alpha - \sin(2\alpha))\right)}{\pi\delta^2} = \frac{k}{\pi}\left(\frac{1}{4}\sin(2\theta) + \left(\frac{r_{\mathcal{S}}}{\delta}\right)^2\left(\alpha - \frac{1}{2}\sin(2\alpha)\right)\right)$$

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

The above analysis gives an exact value for this ratio, but the following approximation yields a better intuition for how it changes with k. In Figure 8a, consider the circular sector CAB, where C is the center of the $C_{\mathcal{S}}$. Recall that the centers of the bounding arcs are on the boundary of $C_{\mathcal{S}}$. The closest point from C on arc AB is D, with $|CD| = \delta$. Let the angle ACD be denoted by θ. Since $\lim_{\theta\to 0}(\sin(\theta)/\theta) = 1$, we can approximate the circular arc between A and D with the line segment AD, for sufficiently small θ. By increasing k, the number of arcs defining the boundary of $\mathcal{R}^+$ can be increased. This in turn will result in smaller arc lengths and smaller θ.

Proceeding thus, we replace the circular sectors which collectively define $\mathcal{R}^+$ with triangles, obtaining a polygon with k sides. Figure 8b shows the polygon with eight sides which replaces the area of Figure 8a. Now, AD = $\delta \tan\theta$, so $\triangle$ CAB = $2 * (\triangle$ CAD$) = \delta^2 \tan\theta$. Since $\mathcal{R}^- = \pi\delta^2$, we get

$$\frac{\mathcal{R}^+}{\mathcal{R}^-} = \frac{k\delta^2 \tan\theta}{\pi\delta^2} = \frac{k\tan\theta}{\pi} = \frac{k}{\pi}\tan\frac{2\pi}{k} = \frac{\sin\frac{2\pi}{k}}{\frac{\pi}{k}} \cdot \frac{1}{\cos\frac{2\pi}{k}}$$

We know that $\lim_{x\to 0}\frac{\sin 2x}{x} = \lim_{x\to 0}\cos 2x = 1$. Since $\frac{2\pi}{k} \to 0$ as $k \to \infty$, $\frac{\mathcal{R}^+}{\mathcal{R}^-} \to 1$. Figure 9a shows that this ratio approaches 1 very rapidly as k increases.

Finally, we prove that $r_{\mathcal{S}} \approx (1 + O(1/k^2))r_{\mathcal{S}}^{in}$. That is, we can get arbitrarily good approximations to the radius by maintaining only a constant number k of direction vectors, in $O(\log n)$ time per update.

Theorem 5.1. *Let $\mathcal{S}$ be the current set of points in the streaming window, and $\mathcal{F}_{\mathcal{S}}^k$ be the set of frontier points, so that $r_{\mathcal{S}}^{in}$ is the radius of the minimal enclosing circle for $\mathcal{F}_{\mathcal{S}}^k$. Then, $r_{\mathcal{S}} \le r_{\mathcal{S}}^{in}(1 + O(1/k^2))$.*

Proof: Figure 9c shows a set of points $\mathcal{S}$, their corresponding $\mathcal{P}_{\mathcal{S}}^{k}$ = ABCDEFGH, the set $\mathcal{F}_{S}^{k}$ of frontier points in the directions of vectors $\vec{d_1}, \vec{d_2}, \ldots$, as well as the minimal enclosing circle $C_{\mathcal{S}}^{in}$ for the frontier points f_1, f_2, and f_3. Let O be the center of $C_{\mathcal{S}}^{in}$. Clearly, $|\mathrm{O}f_1| = r_{\mathcal{S}}^{in}$. 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 trigonometry, all points in this isosceles triangle, and hence all points in the sector of $\mathcal{P}_{\mathcal{S}}^{k}$ defined by OX and OY lie within distance $|\mathrm{O}f_1|/\cos\left(\frac{\pi}{k}\right) = r_{\mathcal{S}}^{in}/\cos\left(\frac{\pi}{k}\right)$ of the point O.

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

Hence, $r_{\mathcal{S}} \leq r_{\mathcal{S}}^{in}/\cos\left(\frac{\pi}{k}\right)$. Using a Taylor series expansion, $\cos\left(\frac{\pi}{k}\right) = 1 - \frac{\pi^2}{2k^2} + O\left(\frac{\pi^4}{k^4}\right)$, 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. Hence $r_{\mathcal{S}} \leq r_{\mathcal{S}}^{in}\left(1 + O\left(\frac{1}{k^2}\right)\right)$. □

6 IDENTIFYING DWELL REGIONS OVER ARCHIVAL DATA

So far, we have described an online algorithm to find dwell regions for streaming data. To identify dwell regions from archival data, the most straightforward way would be to apply the online algorithm to the archival dataset. This would be inefficient, but we use this method as the *baseline* standard for evaluating our approach for archival data. No other benchmark is available or appropriate, since this is the first paper to address the dwell region problem for archival data.

Algorithm 5 BaselineApproach (τ_q, r_q)

```
1:  for each trajectory T in the dataset do
2:      for k ← 0 : length(T) do
3:          S ← (x_k, y_k, t_k), τ ← 0, e ← k + 1
4:          while S satisfies (τ_q, r_q) and e < length(T) do
5:              append (x_e, y_e, t_e) to S, τ ← τ + (t_e − t_{e−1}), e ← e + 1
6:              Find S′ from S and compute C_S
7:              if r_S > r_q then
8:                  remove (x_e, y_e, t_e) from S, τ ← τ − (t_e − t_{e−1})
9:                  Find S′ from S and compute C_S
10:                 if r_S ≤ r_q and τ ≥ τ_q then
11:                     add C_S to the result-set R          ▷ R : the set with resultant dwell regions
12:                 end if
13:                 break
14:             end if
15:         end while
16:     end for
17: end for
18: return R
```

Algorithm 5 gives the pseudocode for this baseline approach. Given a dwell region query (τ_q, r_q), the $C_{\mathcal{S}}$ is computed using the online algorithm (Algorithm 1) at Line 6 for each subtrajectory $\mathcal{S}$ of a trajectory T starting at each point of T. We continue adding points to $\mathcal{S}$ until $r_{\mathcal{S}} > r_q$. We then remove the last added point from $\mathcal{S}$, compute the $C_{\mathcal{S}}$ again (Line 9), and add $\mathcal{S}$ to the result-set $\mathcal{R}$ if $\mathcal{S}$ satisfies the query (Lines 7–14).

As noted in Section 7, this baseline approach is too inefficient to be useful in practice. We now show how to extend the ideas we developed for online computation of dwell regions to computing them on archival data. We

address the archival case in two phases. First, we present the *ρ-Index*, a natural extension of the online case, which creates an index using query radius ranges only. Based on an evaluation of the ρ-Index, we refine our approach to develop the *τ-Index*, which indexes subtrajectories using both query radius ranges and dwell durations. These indexes allow us to focus on small portions of the archival dataset to identify dwell regions. We show that queries using these indexes can be an order of magnitude faster than queries run without them. These indexes are built in three steps:

(1) **Partitioning:** We partition the allowable query radius and duration ranges using a partitioning scheme.
(2) **Extraction:** We extract subtrajectories matching the ranges in each partition by running the online query algorithm over the archival trajectories.
(3) **Storage:** We store these extracted subtrajectories into our index structure.

The one-dimensional ρ-Index is created by partitioning the range of possible query radii, extracting subtrajectories from the dataset that match this radius, and storing them in *bins*. Each bin in the ρ-Index corresponds to a fixed query radius, and each bin entry is a subtrajectory that satisfies a dwell region query of that radius and some duration.

The τ-Index is built by partitioning both query radius and duration, and is an $M \times N$ grid. Each entry in a *cell* of the τ-Index contains a subtrajectory that matches the radius and duration ranges corresponding to that cell. The semantics of ρ-Index bins are different from those for τ-Index cells, so the query algorithms for the two indexing methods are different.

In Sections 6.1 and 6.2, we elaborate on how we preprocess archival data to build the ρ-Index and τ-Index respectively, and how these indexes speed up the evaluation of dwell region queries on archival data.

6.1 The One-Dimensional ρ-Index

Let r_{min} and r_{max} be the minimum and maximum permissible query radii, so every query radius r_q satisfies $r_{min} \leq r_q \leq r_{max}$. We partition the range $[r_{min}, r_{max}]$ into $N-1$ discrete intervals $r_{min} = r_0, r_1, \ldots, r_{N-1} = r_{max}$ using a partitioning scheme. In this section, we assume a linear partitioning scheme where all intervals are equal in size and $r_i = r_0 + i\left(\frac{r_{max}-r_{min}}{N-1}\right)$. We shall discuss other partitioning schemes in Section 6.3. The ρ-Index is a sequence of bins $\rho_{min}, \ldots, \rho_{max}$ where each bin ρ_i collects all subtrajectories that satisfy an online dwell region query of radius r_i and arbitrary duration. We call r_i the *bin radius* of bin ρ_i.

6.1.1 Preprocessing and building the ρ-Index. We say subtrajectory $\mathcal{S} = (x_k, y_k, t_k), \ldots, (x_j, y_j, t_j)$ is *r_i-maximal* if $\mathcal{S}$ can be enclosed in a circle of radius r_i, but not if extended either with $(x_{k-1}, y_{k-1}, t_{k-1})$ or $(x_{j+1}, y_{j+1}, t_{j+1})$. For each bin radius r_i, we iterate over trajectories in our dataset and extract all r_i-maximal subtrajectories. We store them in bin ρ_i sorted by duration.

Algorithm 6 presents the pseudocode for updating the r_i-maximal bins of the ρ-Index for each trajectory T of the dataset. We first build the ρ-Index with N sorted r_i-maximal bins, where $r_i \in \{r_0, r_1, \ldots, r_{max}\}$. We extract r_i-maximal subtrajectories from a trajectory T in our dataset as follows: We initialize $\mathcal{S}$ to a single point (x_k, y_k, t_k) in T, and add the remaining points of T to $\mathcal{S}$, one at a time, in temporal order. We find $r_{\mathcal{S}}^{in}$ and $r_{\mathcal{S}}^{out}$ using the online algorithm after each addition. We continue adding points to $\mathcal{S}$ as long as $r_{\mathcal{S}}^{out} \leq r_i$. When $r_{\mathcal{S}}^{out} > r_i$, we find $r_{\mathcal{S}}$ and check if $r_{\mathcal{S}} \leq r_i$.

If $r_{\mathcal{S}} > r_i$, we discard the last added point from $\mathcal{S}$, and if $r_{\mathcal{S}} \leq r_i$, we enter $\mathcal{S}$ into ρ_i. We now consider the next subtrajectory by moving the starting point from (x_k, y_k, t_k) to $(x_{k+1}, y_{k+1}, t_{k+1})$, and repeat the above procedure with the same bin radius r_i. After processing T for r_i, we process T for r_{i+1} and so on.

6.1.2 Structure of ρ-Index Bins. Let $s_i^k \in \rho_i$ and $s_{i+1}^k \in \rho_{i+1}$ be two subtrajectories of T appearing as entries in ρ-Index bins. Let these subtrajectories start at point (x_k, y_k, t_k) of T, with durations τ_i^k and τ_{i+1}^k respectively. Clearly, $r_i^k \leq r_{i+1}^k$, $\tau_i^k \leq \tau_{i+1}^k$, and $s_i^k \subseteq s_{i+1}^k$. We say s_{i+1}^k is the *extension* of s_i^k in ρ_{i+1} (we term this as the

Algorithm 6 UpdateRhoIndex (ρ-Index, r_i, T)

```
1: for k ← 0 : length(T) do
2:     S ← (x_k, y_k, t_k), τ ← 0, e ← k + 1
3:     while S satisfies r_i and e < length(T) do
4:         append (x_e, y_e, t_e) to S, τ ← τ + (t_e − t_{e−1}), e ← e + 1
5:         compute r_S^in and r_S^out
6:         if r_S^out > r_i then
7:             compute r_S
8:             if r_S > r_i then
9:                 remove (x_e, y_e, t_e) from S, and compute r_S
10:                if r_S ≤ r_i then
11:                    add S to the bin of ρ-Index with r_i
12:                end if
13:                break
14:            end if
15:        end if
16:    end while
17: end for
```

subsequence property). Let the last point of s_{i+1}^k be (x_ℓ, y_ℓ, t_ℓ). To facilitate query evaluation, we store ℓ and τ_{i+1}^k with the entry s_i^k.

Each entry s_i^k for a subtrajectory $\mathcal{S}$ in the bin ρ_i is a 5-tuple $\langle T, k, \tau, \ell, \tau_{i+1}^k \rangle$, where T is the trajectory ID of $\mathcal{S}$ in the archival dataset, k is the start index of $\mathcal{S}$ in T, τ is the duration of $\mathcal{S}$, ℓ is the last point index of s_{i+1}^k in ρ_{i+1}, and τ_{i+1}^k is the duration of s_{i+1}^k in ρ_{i+1}.

Figure 10 shows the ρ-Index structure with N bins $\rho_{min} = \rho_0, \rho_1, \ldots, \rho_{N-1} = \rho_{max}$. The dots represent r_i-maximal subtrajectory entries, sorted by duration.

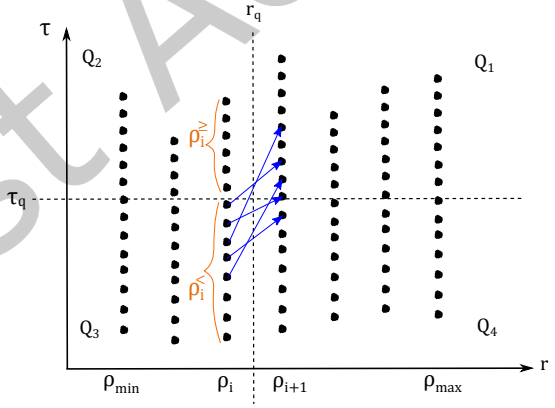


Fig. 10. Evaluation of a dwell region query (τ_q, r_q) using the ρ-Index.

6.1.3 Query Evaluation. Let a query arrive with query radius r_q and duration τ_q. If r_q equals the radius r_i corresponding to a bin ρ_i, the subtrajectory entries in ρ_i with duration $\tau \geq \tau_q$ satisfy the query, and are added to the result set. Entries in ρ_i are r_i-maximal, so we can ignore the members of $\rho_j, j > i$. Also, because s_i^k is a subsequence of s_{i+1}^k, that is, $s_i^k \subseteq s_{i+1}^k$, for $s_i^k \in \rho_i$, and $s_{i+1}^k \in \rho_{i+1}$, we automatically capture all elements of ρ_j, $j < i$.

Now consider the case where r_q falls between bin radii, so $r_i < r_q < r_{i+1}$. Figure 10 can be viewed as a plot of the bin radii versus the duration of maximal subtrajectories. The query radius r_q and duration τ_q divide the space into four quadrants Q_1, Q_2, Q_3 and $Q4$. However, because of the subsequence property alluded to above, we need only consider the entries of ρ_i and ρ_{i+1} in these quadrants. All entries $s^k_{i+1} \in \rho_{i+1}$ in quadrant Q_4 have $r^k_{i+1} > r_q$ and $\tau^k_{i+1} < \tau_q$. These fail the query criteria, and can be disregarded. We can add all entries $s^k_i \in \rho_i$ in quadrant Q_2 to the query result, as they have $r^k_i < r_q$ and $\tau^k_{i+1} \geq \tau_q$.

Let ρ_i be partitioned as $\rho_i = \rho^<_i \cup \rho^\geqslant_i$ such that every entry $s^k_i \in \rho^<_i$ has duration $\tau^k_i < \tau_q$, and every entry $s^k_i \in \rho^\geqslant_i$ has duration $\tau^k_i \geq \tau_q$. Similarly, ρ_{i+1} can be partitioned as $\rho_{i+1} = \rho^<_{i+1} \cup \rho^\geqslant_{i+1}$. We have already shown that all entries $s^k_i \in \rho^\geqslant_i$ in quadrant Q_2 are in the query result, but the entries $s^k_{i+1} \in \rho^<_{i+1}$ in quadrant Q_4 are not. We are now left with $\rho^<_i$ and $\rho^\geqslant_{i+1}$.

The entries $s^k_i \in \rho^<_i$ in quadrant Q_3 have radii $r^k_i \leq r_q$ and durations $\tau^k_i < \tau_q$. Hence we can extend each s^k_i in quadrant Q_3 maximally with points in T to get $s^k_{i^>}$ such that $\tau^k_{i^>} \geq \tau_q$ and $r^k_{i^>} \leq r_q$. Conversely, the entries $s^k_{i+1} \in \rho^\geqslant_{i+1}$ in quadrant Q_1 have radii $r^k_{i+1} > r_q$ and durations $\tau^k_{i+1} \geq \tau_q$. We can shorten each s^k_{i+1} in quadrant Q_1 minimally to arrive at $s^k_{i+1^<}$, which ensures both $\tau^k_{i+1^<} \geq \tau_q$ and $r^k_{i+1^<} \leq r_q$. As s^k_i and s^k_{i+1} both start at (x_k, y_k, t_k) in T, s^k_{i+1} is the extension of s^k_i in ρ_{i+1}. We can hence arrive at s^k_i by shortening s^k_{i+1}, or we can arrive at s^k_{i+1} by extending s^k_i. If we choose to shorten s^k_{i+1}, we do not need to extend s^k_i, as both will give us the same results, and vice versa.

To make a decision between shortening or extending subtrajectory entries, we use a heuristic based on the difference between r_q and r_i or r_{i+1}. If $|r_q - r_i| < |r_q - r_{i+1}|$, we choose to extend the entries in $\rho^<_i$; otherwise, we choose to shorten the entries in $\rho^\geqslant_{i+1}$. We extend or shorten the entries as long as they do not satisfy the given query, and then add them to the result set.

Extending or shortening a subtrajectory. Figure 11a illustrates extending or shortening a subtrajectory. Let $S_1 = \{p_1, p_2, \ldots, p_5\}$ be an entry in $\rho^<_i$ and $S_2 = \{p_1, p_2, \ldots, p_9\}$ be an entry in $\rho^\geqslant_{i+1}$. A and B depict circles with radius r_i and r_{i+1}, enclosing S_1 and S_2 respectively. Let C represent the dwell region with radius r_q that encloses the subtrajectory $S = \{p_1, p_2, \ldots, p_7\}$.

To answer a query with radius r_q, we can use the above heuristic and either extend S_1 or shorten S_2. We can extend S_1 by adding points starting with p_6, computing the resulting smallest enclosing circles, until we find S. Similarly, we might shorten S_2 by removing points one by one, until we find S. However, these iterative operations are expensive.

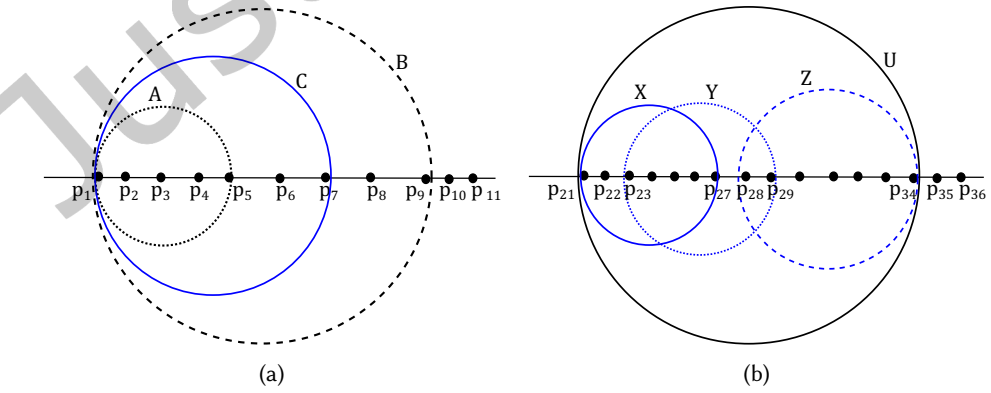


Fig. 11. (a) An example of extending or shortening a subtrajectory, and (b) An example showing the importance of storing multiple subtrajectory entries instead of one entry in the τ-Index.

These operations can be especially expensive if we have to extend or shorten a subtrajectory entry of the ρ-Index to find multiple dwell regions during query evaluation. Consider the trajectory shown in Figure 11b, and let $S' = \{p_{21}, p_{22}, \ldots, p_{27}\}$, $S'' = \{p_{23}, p_{24}, \ldots, p_{29}\}$ and $S''' = \{p_{28}, p_{29}, \ldots, p_{34}\}$ be the dwell regions that satisfy a given query. Assume that these are not stored in the ρ-Index. Let X, Y and Z be the smallest enclosing circles of S', S'' and S''' respectively. Let the subtrajectory $S_3 = \{p_{21}, p_{22}, \ldots, p_{34}\}$ be an entry in $\rho^{\geqslant}_{i+1}$ and U be the circle with radius r_{i+1}, enclosing S_3. Now, to find S' by shortening S_3, we must remove the points $p_{34}, \ldots, p_{28}$, one by one, each time performing the operations we marked as expensive. Similarly, to find S'' and S''', we must remove points one by one from both ends of S_3 and from the beginning of S_3 respectively. Instead, if we had stored S', S'', S''', S_3 in an index, we would not need to extend or shorten S_3. Another observation about the ρ-Index is that the subsequence property of subtrajectory entries $s^k_i \in \rho_i$ allows us to answer queries by examining just the two bins ρ_i and ρ_{i+1}. While this speeds up query evaluation, every point in a trajectory will create an entry in every bin (and this happens for all trajectories in the archival dataset). We will address these shortcomings of the ρ-Index by introducing the τ-Index which avoids extending or shortening subtrajectories during query evaluation time and removes the subsequence dependency among the subtrajectory entries stored in the index.

6.2 The Two-Dimensional τ-Index

An analysis of the performance of the ρ-Index reveals that while the binning idea is effective, it results in bins that are quite large, since the time dimension remains unrestricted. We now show how to partition the dataset along both distance and time dimensions, a further refinement of the ideas behind the ρ-Index. We will show that this strategy is very effective, and yields the τ-Index, which is a two-dimensional index with performance far superior to that of the ρ-Index. The τ-Index is useful for both decision and dwell region queries.

Let the query radius r_q and duration τ_q be bounded, with $r_{min} \leq r_q \leq r_{max}$, and $\tau_{min} \leq \tau_q \leq \tau_{max}$. The τ-Index is effectively an $M \times N$ grid, whose cells are obtained by partitioning these ranges. Let $\tau_{min} = \tau_0 < \ldots < \tau_i < \ldots < \tau_{M-1} = \tau_{max}$ and $r_{min} = r_0 < \ldots < r_j < \ldots < r_{N-1} = r_{max}$. A *cell* (τ_i, r_j) in the τ-Index contains all maximal subtrajectories matching queries (τ_q, r_q) with $\tau_i \leq \tau_q \leq \tau_{i+1}$ and $r_{j-1} < r_q \leq r_j$. We discuss various partitioning schemes in Section 6.3.

6.2.1 Structure of the τ-Index. Each entry in a τ-Index for a subtrajectory $\mathcal{S}$ in a cell (τ_i, r_j) is a 6-tuple $\langle T, k, e, \tau, r_{\mathcal{S}}, c_{\mathcal{S}} \rangle$, where T is the trajectory ID of $\mathcal{S}$ in the archival dataset, k and e are, respectively, the start and end indexes of $\mathcal{S}$ in T, τ is the duration of $\mathcal{S}$, $r_{\mathcal{S}}$ is the radius of the smallest enclosing circle $C_{\mathcal{S}}$, and $c_{\mathcal{S}}$ is the center of $C_{\mathcal{S}}$. We will have $r_{j-1} < r_{\mathcal{S}} \leq r_j$ and $\tau_i \leq \tau \leq \tau_{i+1}$. The entries in each cell are sorted by radius $r_{\mathcal{S}}$. Unlike the ρ-Index an, τ-Index entries contain no information regarding subsequent subtrajectories. Figure 12 shows the τ-Index structure.

In principle, we could build the τ-Index from the ρ-Index by applying the defined duration partitions to the subtrajectories in the ρ-Index bins. However, placing these subtrajectories into the appropriate cells of the τ-Index requires extending or shortening them, which can be expensive (as discussed in Section 6.1). Instead of obtaining the τ-Index cells by segmenting the ρ-Index bins, we build the τ-Index directly by partitioning both query radius and time durations. We discuss the efficient preprocessing steps in Section 6.2.2.

6.2.2 Preprocessing. To build a τ-Index of $M \times N$ sorted grid cells (τ_i, r_j), we precompute dwell regions from each trajectory T for each duration τ_i, and store a subtrajectory of T in to a cell (τ_i, r_j) in the τ-Index if it satisfies $r_{j-1} < r \leq r_j$ and $\tau_i \leq \tau \leq \tau_{i+1}$ (see Algorithm 7). We preprocess all trajectories T for all duration partitions τ_i, where $0 \leq i \leq (M-1)$. The steps to preprocess a trajectory T, and store its qualified subtrajectories $\mathcal{S}$ in the cells (τ_i, r_j) of the index are as follows:

Step 0: Initialize a subtrajectory $\mathcal{S}$ to a single point (x_k, y_k, t_k) in T.
Step 1: Add the remaining points of T to $\mathcal{S}$ in temporal order until its duration $\tau \leq \tau_i$.

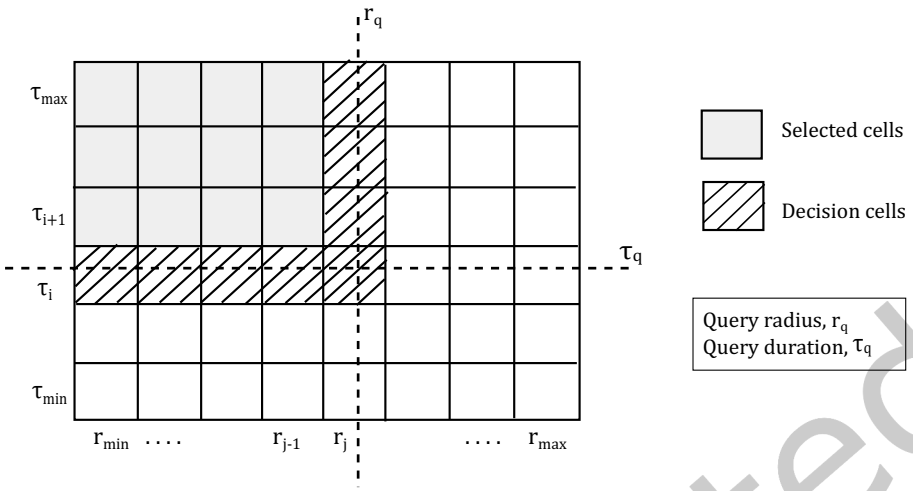


Fig. 12. The τ-Index and its use in answering dwell region queries.

Algorithm 7 UpdateTauIndex (τ-Index, τ_i, r_j, T)

```
1: k ← 0
2: while all points in T not considered do
3:     S ← (x_k, y_k, t_k), τ ← 0, e ← k + 1          ▷ S : current subtrajectory, k : start point index of S
4:     while τ ≤ τ_i do
5:         append (x_e, y_e, t_e) to S, τ ← τ + (t_e − t_{e−1}), e ← e + 1
6:     end while
7:     compute r_S^in and r_S^out
8:     if r_S^out < r_S^in or r_S^out > r_j then
9:         k ← k + 1, continue
10:    end if
11:    compute r_S
12:    if r_{j−1} < r_S ≤ r_j then
13:        while (x_e, y_e, t_e) is inside C_S and τ_i ≤ τ ≤ τ_{i+1} do
14:            append (x_e, y_e, t_e) to S, τ ← τ + (t_e − t_{e−1}), e ← e + 1
15:        end while
16:        add E = ⟨T, k, e, τ, r_S, c_S⟩ to the cell (τ_i, r_j) of τ-Index
17:    end if
18:    k ← k + 1
19: end while
```

Step 2: Compute $r_{\mathcal{S}}^{in}$ and $r_{\mathcal{S}}^{out}$. If $r_{\mathcal{S}}^{out} < r_{\mathcal{S}}^{in}$ or $r_{\mathcal{S}}^{out} > r_j$, discard $\mathcal{S}$ and go to Step 4.
Step 3: Compute $C_{\mathcal{S}}$.
(a) If $r_{\mathcal{S}} > r_j$, discard $\mathcal{S}$ for the cell (τ_i, r_j) and go to Step 4.
(b) If $r_{j-1} < r_{\mathcal{S}} \leq r_j$, add more points of T to $\mathcal{S}$ in temporal order as long as the newly added points can be enclosed by the same $C_{\mathcal{S}}$ and $\tau_i \leq \tau \leq \tau_{i+1}$. Then add an entry $E = \langle T, k, e, \tau, r_{\mathcal{S}}, c_{\mathcal{S}} \rangle$ for $\mathcal{S}$ in the cell (τ_i, r_j).
Step 4: Slide the window forward to find the next subtrajectory starting at $(x_{k+1}, y_{k+1}, t_{k+1})$ and go to Step 1.

6.2.3 Query Evaluation. Figure 12 shows how to evaluate a dwell region query using the τ-Index. Let τ_q and r_q be the duration and the radius of a dwell region query, where $\tau_i \leq \tau_q < \tau_{i+1}$ and $r_{j-1} < r_q \leq r_j$. To build the result set of dwell regions, we choose the candidate cells from the τ-Index, and find the resultant dwell regions for the given query from these cells. These candidate cells can be either selected cells or decision cells (as shown in Figure 12). We can find the dwell regions satisfying the given query from the τ-Index as follows:

1. **Choose candidate cells:** The subtrajectory entries $\mathcal{S}$ in the white cells in Figure 12 have either radii $r_{\mathcal{S}} > r_q$, or duration $\tau < \tau_q$, and do not meet the query criteria. We discard them from the query results. Since $\tau_i \leq \tau_q < \tau_{i+1}$ and $r_{j-1} < r_q \leq r_j$, we can locate the query results to be in cells $(\tau_{\geq i}, r_{\leq j})$. These cells are the candidate cells.
2. **Classify candidate cells:** We classify the chosen candidate cells of the τ-Index as follows:
   (i) *[Selected Cells]* All the subtrajectory entries $\mathcal{S}$ in the candidate cells $(\tau_{>i}, r_{<j})$ have $\tau > \tau_q$ and $r_{\mathcal{S}} < r_q$, and satisfy the given query. We call these cells *Selected cells*. We retrieve the smallest enclosing circles from all entries in these cells and add them to the result set.
   (ii) *[Decision Cells]* We call the candidate cells $(\tau_{\geq i}, r_j)$ and $(\tau_i, r_{\leq j})$ *Decision cells*. We add the smallest enclosing circles of the subtrajectory entries of these cells that meet the query criteria to the result set, and discard the others.
3. **Explore decision cells:** The subtrajectory entries in each cell of the τ-Index are sorted by their radius values. Depending on the duration and the radius, the subtrajectory entries of the decision cells may or may not meet the query criteria.
   (a) From the decision cell (τ_i, r_j), we can consider the entries as long as their radii $r_{\mathcal{S}} \leq r_q$, and add the smallest enclosing circles of these entries to the result set if and only if their duration $\tau \geq \tau_q$.
   (b) From the decision cells $(\tau_{>i}, r_j)$, we can add the smallest enclosing circles of the entries to the result set as long as their radii $r_{\mathcal{S}} \leq r_q$, as their duration $\tau > \tau_q$.
   (c) From the decision cells $(\tau_i, r_{<j})$, we can add the smallest enclosing circles of the entries to the result set if and only if their duration $\tau \geq \tau_q$, as their radii $r_{\mathcal{S}} < r_q$.

6.3 Partitioning schemes for ρ-Index and τ-Index

The choice of the number and size of partitions can have a significant effect on performance of the indexes in both preprocessing and query stages. If the chosen partitions are too fine, the ρ-Index will have a large number of ρ bins, which could make index construction prohibitively expensive in both time and space. On the other hand, having too few partitions could result in high query times, as it increases the number of shorten and extend operations for most queries. The same trade-off applies to the τ-Index. Having too many cells increases the preprocessing time and space significantly, while having too few cells would increase the query times, possibly defeating the purpose of preprocessing altogether.

Choosing a suitable partitioning scheme for a range of the input parameters involves reasoning about precise application for which dwell region queries are being used, the ranges of permissible input query parameters, and the expected distribution of entries in the bins or cells in the dataset. Ideally, all bins or cells in the index would have roughly equal numbers of entries, as a skewed distribution can slow down index lookups. Now, say that we seek to determine if some person is engaged in surveillance. Here, we might expect most dwell region queries to have a small r_q value, since while surveilling an area, humans might not want to stray too far from it. We use this criterion in our experiments using the GeoLife dataset [59], which comprises trajectories of individuals collected over approximately two years and within a small geographic region (city), and the TDrive dataset [60] that consists of taxi trajectories collected for one week within a small geographic area.

In contrast, animals tend to stay in small regions for long times and move steadily for food. Hence, we consider extended time windows during experiments with the ElkDeer dataset [42], which contains trajectories of deer,

elk and cattle collected to document the habitat use of these animals in the presence of habitat changes and human activities in an experimental forest setting. We note that similar queries have been found useful for widely different applications from locating galactic habitable zones between stars [16], and finding regions of overlap from wildlife tracking data [31]. Had we used dwell region queries for one of these other semantics, our current assumptions would not hold and we would have to change them as appropriate.

Given a continuous range $[l, h]$ where $l, h \in \mathbb{R}$, a *partitioning scheme* divides $[l, h]$ into n ranges and returns a sequence $l = s_0, s_1, s_2, \ldots, s_{n-1} = h$, where $s_i \in \mathbb{R}$ and an s_i denotes the boundaries of a partition. We use three partitioning schemes:

(i) A *linear partitioning scheme* divides the given range into partitions of equal size, and therefore $s_i = s_0 + i\left(\frac{s_{max}-s_{min}}{n-1}\right)$.
(ii) A *quadratic partitioning scheme* divides the input range into smaller ranges of sizes that increase at a quadratic rate, so $s_i = (i + c)^2$, where $c \geq 1$.
(iii) An *exponential partitioning scheme* divides the input range into smaller ranges of sizes that increase as an exponential of a given constant x, so $s_i = ax^i + c$, where $a \neq 0, c \geq 0$.

Table 2 shows the partitioning schemes used to build the ρ-Index and τ-Index for radius and duration dimensions in this paper. We use the linear partitioning scheme for the radius dimension while constructing both ρ-Index and τ-Index, and show experiments with linear, quadratic and exponential partitioning schemes in the duration dimension of the τ-Index.

Dimension	**Partitioning Scheme**	
	ρ-Index	**τ-Index**
Radius	Linear	Linear
Duration	N/A	Linear, Quadratic and Exponential

Table 2. Partitioning schemes used for ρ-Index and τ-Index in our experiments.

We choose quadratic and exponential schemes for only the duration dimension because all of our datasets contain trajectories that span a much longer duration and only cover a small geographic area. In an exponential partitioning scheme for the radius dimension, the distance span of different cells would have been widely variant, ranging from a cell covering only a small neighborhood, to another covering a large section of the city. Such highly skewed partitions are generally undesirable as they can lead to large differences between the number of entries per cell, which may affect query performance. We can make the same argument for the quadratic partitioning scheme.

Assuming constant speed and fixed sampling frequency, an object travels about the same distance in each sampled interval. A quadratic or exponential growth in the radii of bins or cells results in a skewed distribution in the number of entries. Further, the subtrajectory entries in bin ρ_{i+1} can be very large compared to the corresponding subsequent subtrajectory entries in bin ρ_i. During query evaluation, we must perform extend or shorten operations on the bin entries, so that a quadratic or exponential partitioning scheme increases the number of shorten or extend operations during query evaluation. We therefore use linear partitioning for the radius dimension in all experiments.

For the duration dimension of the τ-Index, we use linear, quadratic and exponential partitioning schemes in the experiments. Since the cells with the linear partitioning scheme span equal ranges of duration, for a fixed number of partitions and a fixed τ_{max}, it sets relatively higher duration for cells than that of the corresponding cells with quadratic and exponential partitioning schemes. To build the τ-Index, these higher durations in the cells can affect the τ-Index performance (considering both preprocessing and query evaluation). We will observe its effect in the experiments (see Section 7.2).

7 EXPERIMENTS

We evaluate dwell region query performance both on streaming (Section 7.1) and archival data (Section 7.2). We conduct our experiments using three real-world datasets, namely Geolife, ElkDeer and TDrive (described below); we also run scaling experiments using much larger synthetic datasets (see Section 7.3). The GeoLife dataset [59] contains public activity data, such as shopping, dining, sightseeing, hiking and cycling in Beijing, China. The dataset contains 8,970 trajectories for 165 objects and a total of 24,778,552 spatial points. The sampling frequency of the points in trajectories is $2 \sim 5$ seconds. The ElkDeer dataset [42] contains trajectories of deer, elk and cattle in the Starkey Experimental Forest and Range in Oregon, USA. It contains 253 object trajectories and $> 287,000$ spatial points. The dataset was collected within 40 square miles at an average sampling frequency of ~ 1 hour. The TDrive dataset [60] contains one-week trajectories of 10,375 taxis and about 17 million spatial points. The sampling frequency of points in trajectories is $2 \sim 6$ minutes. Table 3 summarizes the properties of the real-world datasets used. These datasets are chosen as their sampling frequencies range from seconds to hours, capturing object movement in different practical contexts. Further, this allows us to better evaluate the effect of the duration dimension in the τ-Index.

Dataset	Location	No. of trajectories	No. of spatial points	Sampling frequency
GeoLife	Beijing, China	8,970	24,778,552	$2 \sim 5$ seconds
ElkDeer	Starkey, Oregon, USA	253	$> 287,000$	~ 1 hour
TDrive	Beijing, China	10,375	~ 17 million	$2 \sim 6$ minutes

Table 3. Summary of real-world datasets used in the experiments.

All experiments were run on an Intel Xeon E5-2430 processor with a base clock of 2.5 GHz running CentOS 7 with Linux kernel 3.10.0 and 32GB of DDR3 main memory. Our C++ implementation is publicly available[2].

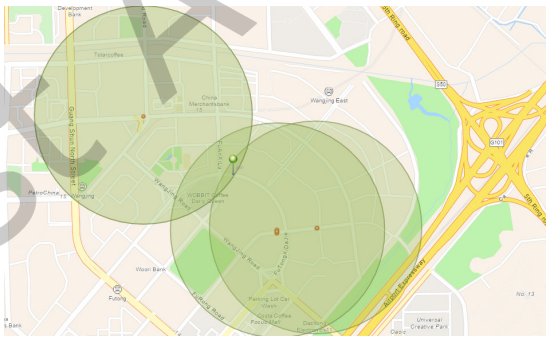


Fig. 13. A dwell region $\mathcal{R}$ (the intersection of the circles) identified from the TDrive dataset, and the pinpoint represents the center of $C_{\mathcal{S}}$.

Figure 13 visualizes a dwell region $\mathcal{R}$ identified from the TDrive dataset with the SEC radius of 0.42 miles. The circles are drawn centering at the trajectory points on $C_{\mathcal{S}}$ and considering the radius. The intersection of these circles is identified as the dwell region $\mathcal{R}$. The pinpoint represents the center of $C_{\mathcal{S}}$ which is inside $\mathcal{R}$, as expected.

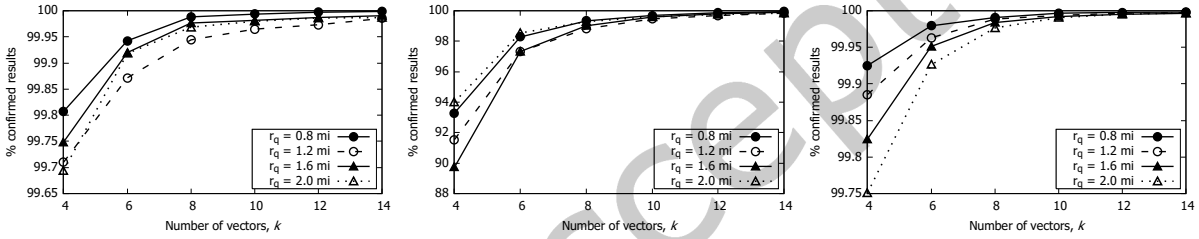
[2]https://github.com/PayasR/DwellRegions-open

7.1 Performance of Online Dwell Region Queries

We ran a series of experiments to demonstrate the pruning power of our method in Section 4 for computing C_S. First, we fixed the streaming window size to 4000 points, and then ran a set of dwell region queries with $\tau_q = 20000$ seconds and $r_q \in \{0.8, 1.2, 1.6, 2.0\}$ miles over all trajectories in GeoLife and TDrive datasets. However, the ElkDeer dataset is much smaller and has a low sampling frequency, we fixed the streaming window size to 120 points, so $\tau_q = 450000$ seconds (=~ 5.2 days) for each dwell region query.

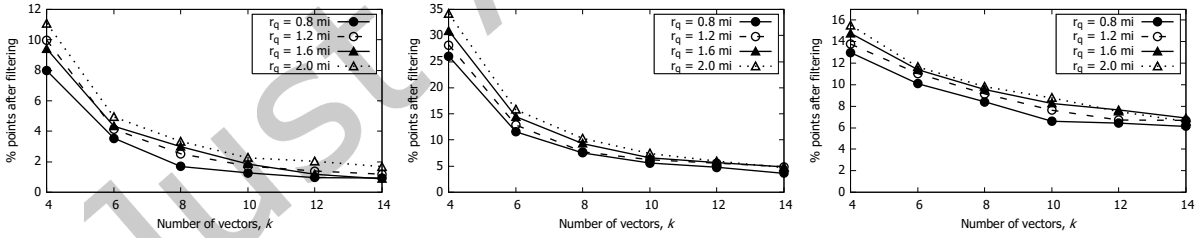
When $r_q \geq r_S^{out}$ or $r_q \leq r_S^{in}$, we can answer decision queries without computing C_S. This feature allows rapid response to decision queries, and is an important benefit of our method. Figure 14 shows the percentage of subtrajectories S where computing C_S is unnecessary. We use this metric to show how well our methods allow answering decision queries quickly using just r_S^{in} and r_S^{out}.

However, if $r_S^{in} < r_q < r_S^{out}$, we must compute C_S. As the number of vectors k increases, we need to compute C_S significantly fewer times, since with higher k, the minimum bounding polygon $\mathcal{P}_S^k$ better resembles C_S, which in turn makes our bounds r_S^{in} and r_S^{out} closer to C_S. We note that C_S needs to be computed for less than 0.3% of queries even with k as low as 4 for both GeoLife and TDrive datasets, and 2% with k as low as 8 for the ElkDeer dataset.



(a) GeoLife Data (b) ElkDeer Data (c) TDrive Data

Fig. 14. Fraction of results where r_S^{in} and r_S^{out} suffice to answer dwell region queries.

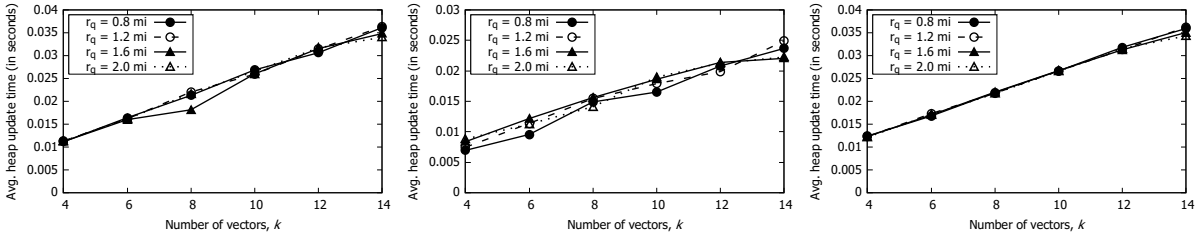


(a) GeoLife Data (b) ElkDeer Data (c) TDrive Data

Fig. 15. Fraction of points selected to calculate the smallest enclosing circle.

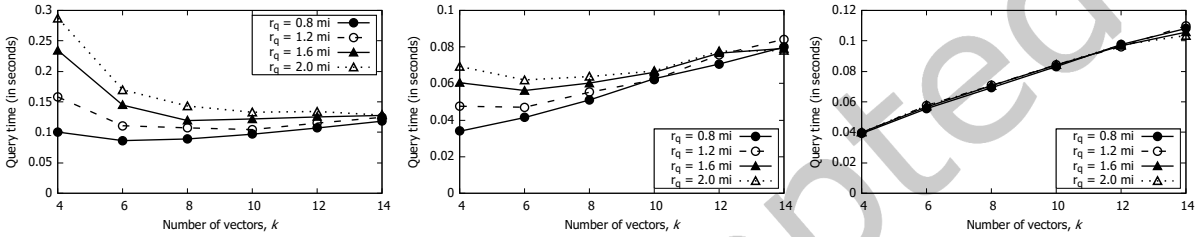
When decision queries cannot be answered only using bounds r_S^{in} and r_S^{out}, points lying between C_S^{in} and C_S^{out} can be used to compute C_S. Figure 15 shows the fraction of such points in subtrajectories. We see that the fraction of selected points decreases quadratically with increase in k. This can be attributed to our data structure becoming increasingly selective with the increase in k, as the $\mathcal{P}_S^k$ better approximates C_S.

Since the online algorithm operates on a stream of points, we now measure the average time required to add an incoming point to the heaps. The results appear in Figure 16. As the number of vectors is equal to the number



(a) GeoLife Data (b) ElkDeer Data (c) TDrive Data

Fig. 16. Average heap update time for an incoming point per dwell region query.



(a) GeoLife Data (b) ElkDeer Data (c) TDrive Data

Fig. 17. Query evaluation time per trajectory.

of heaps, we see that the average heap update time increases linearly with the increase in the number of vectors, but remains within 0.04 seconds. Since the error in the approximated radius is $O(\frac{1}{k^2})$, we can approximate the radius of SEC to within 1% of the actual radius using $k = 10$.

Finally, Figure 17 plots the average time required to evaluate dwell region queries per trajectory in the datasets. We observe that with GeoLife and ElkDeer datasets, the query evaluation time first decreases with the increase of the number of vectors and subsequently increases, and it increases with the increase of the number of vectors while using the TDrive dataset. This behavior can be explained as follows: as the number of vectors increases, our data structure becomes more selective, and we need to process far fewer points to compute C_S. However, the cost of inserting a point in the heaps increases linearly (as we can see in Figure 16). The dip and rise pattern we see in Figures 17a and 17b is the combined effect of the two behaviors.

7.2 Performance of Offline Dwell Region Queries

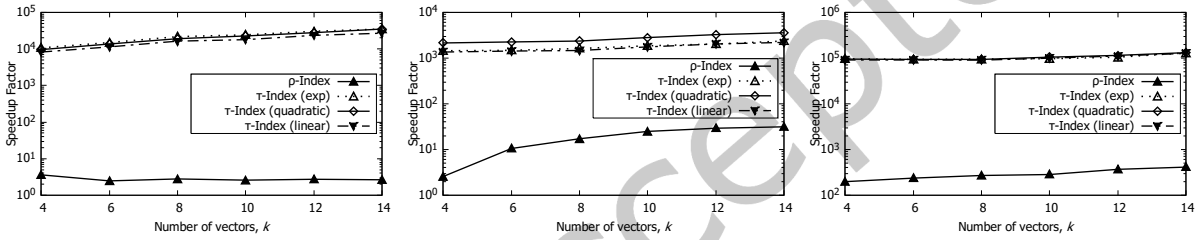
To evaluate the offline preprocessing methods for the archival data presented in Section 6, we built both the ρ-Index and τ-Index for $k \in \{4, 6, 8, \ldots, 14\}$. As discussed in Section 6.3, we choose the linear partitioning scheme in the radius dimension of both of ρ-Index and τ-Index to avoid the highly skewed radius partitions. Also, since our trajectories are constrained to relatively small geographic regions, so we set $r_{min} = 0.6$ miles and $r_{max} = 2.0$ miles for both indexes. While building the τ-Index, we experimentally evaluate linear, quadratic, and exponential partitioning schemes in the duration dimension, setting the number of temporal partitions to $M \in \{3, 4, 5 \ldots, 7\}$. Table 4 shows the parameter set for different partitioning schemes in the duration dimension used for the τ-Index. Since the trajectories in all of our real datasets are collected at a variable sampling frequency and span a much longer duration, we set the durations in hours for GeoLife and TDrive datasets. We increased the durations from hours to days due to the very low sampling frequency of the ElkDeer dataset. We ran queries with

$r_q \in \{0.7, 0.9, 1.1, \ldots, 1.9\}$ miles for each $\tau_q \in \{1, 3, 5, \ldots, 31\}$ hours while using GeoLife and TDrive datasets, and for each $\tau_q \in \{1, 3, 5, \ldots, 31\}$ days while using the ElkDeer dataset.

Partitioning Scheme	GeoLife & TDrive		ElkDeer	
	τ_{min}	τ_{max}	τ_{min}	τ_{max}
Linear	1 hour	64 hours	1 day	64 days
Quadratic	1 hour	49 hours	1 day	49 days
Exponential	1 hour	64 hours	1 day	64 days

Table 4. Parameters for different partitioning schemes in the temporal dimension used for the τ-Index.

We note that the offline dwell region queries operate on entire trajectories and not just on points within a sliding window of fixed size. Therefore, without using indexes, we expect offline dwell region queries to take much longer to execute than their online counterparts.

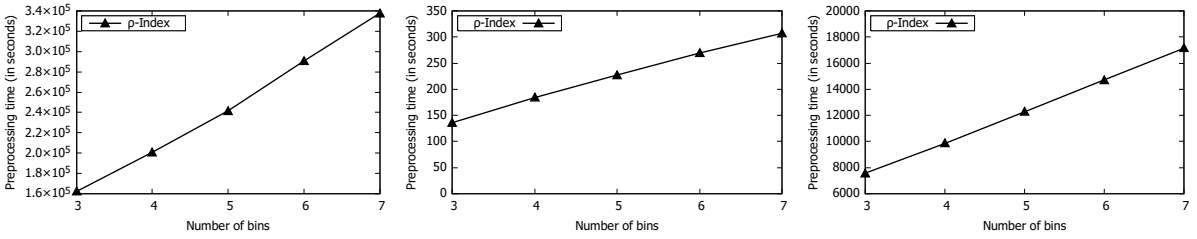


(a) GeoLife Data (b) ElkDeer Data (c) TDrive Data

Fig. 18. Speedup factor of the query evaluation time compared to the baseline approach (i.e., without any indexes) varying the number of vectors k, for queries with $r_q = \{0.7, 0.9, \ldots, 1.9\}$ miles, $\tau_q \in \{1, 3, 5, \ldots, 31\}$ hours for GeoLife and TDrive datasets, and $\tau_q \in \{1, 3, 5, \ldots, 31\}$ days for the ElkDeer dataset.

Dwell region queries run without using indexes serve as the *baseline approach* with which we compare the performance of our indexes. In this case, the online algorithm is directly applied to the trajectories in the archival data. This is reasonable since ours is the first work to address this problem for archival data. Figure 18 shows the speedups in query evaluation time over the baseline when using ρ-Index and τ-Index with linear, quadratic, and exponential partitioning schemes in the temporal dimension. Note the logarithmic scale. When using the ρ-Index, the evaluation algorithm looks up preprocessed subtrajectory entries of the index and extends or shortens as required, using the subsequence property. Therefore, as the number of vectors k increases, the queries get slower as all the points in a trajectory need to be processed. Since the extension and shortening are cheaper operations than processing whole trajectories, the ρ-Index achieves an order of magnitude speedup over the baseline on GeoLife and ElkDeer datasets and is two orders of magnitude faster on the TDrive dataset. Further, as the τ-Index does not use the subsequence property at all during query evaluation, a simple lookup into the index suffices to evaluate queries. As a result, increasing the value of k has a negligible effect on the query evaluation time. The τ-Index speeds up dwell region queries by four orders of magnitude on GeoLife and TDrive datasets and three orders of magnitude faster on the ElkDeer dataset.

Figure 19 shows that the preprocessing time for the ρ-Index increases linearly with the number of bins. This is because the ρ-Index construction requires running queries on the archival data using bin radii as the query radii, and the number of such preprocessing queries grows linearly with the number of bins. When considering



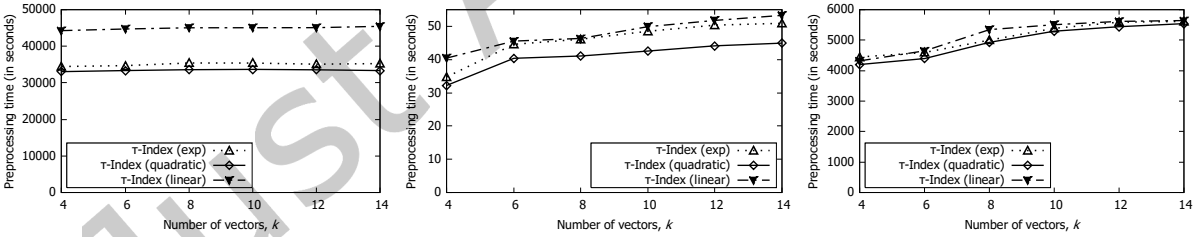
(a) GeoLife Data (b) ElkDeer Data (c) TDrive Data

Fig. 19. Preprocessing time for the ρ-Index for varying the number of bins.

the speedup in query evaluation time (Figure 18) and the preprocessing time (Figure 19) for the ρ-Index, we see that both times are an order of magnitude higher for the GeoLife than the TDrive dataset. This is simply because the GeoLife dataset has a higher sampling frequency and contains 1.5 times more spatial points than the TDrive dataset (see Table 3), and our queries use the same time window for both datasets. We can expect the same behavior for the τ-Index preprocessing time in Section 7.2.1.

Since using the τ-Index results in *significantly* lower query evaluation and preprocessing times than the ρ-Index (Figures 19 and 20), we focus our discussion on the τ-Index for the remainder of this section. This is expected because unlike the two-dimensional τ-Index, the one-dimensional ρ-Index maintains the subsequence property while preprocessing and requires subtrajectory entries to be extended or shortened while evaluating queries (as we discussed in Sections 6.1.3 and 6.2).

7.2.1 Effect of partitioning schemes on τ-Index build time. The time needed to build a τ-Index depends on two factors: the number of partitions of the temporal and radius dimensions, and the partitioning schemes used for each dimension. As discussed in Section 6.3, under our settings, we expect most dwell regions to be of short radii and durations. Hence, cells that store entries with low values of r_i and τ_i have more entries than other cells. The τ-Index does not utilize the subsequence property.

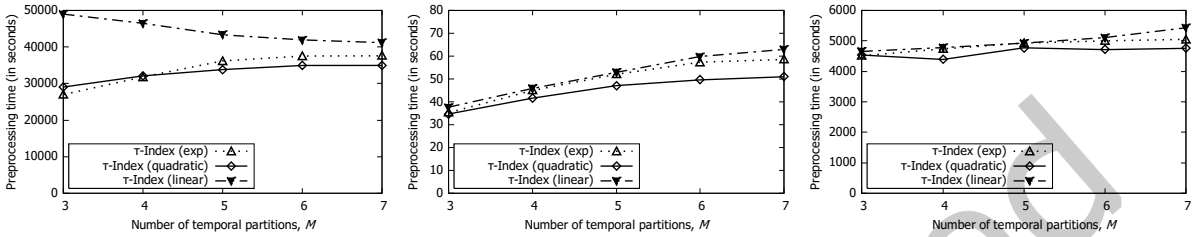


(a) GeoLife Data (b) ElkDeer Data (c) TDrive Data

Fig. 20. Average preprocessing time for the τ-Index for varying the number of vectors k, averaged $\forall M \in \{3, 4, \ldots, 7\}$.

Figure 20 shows the average time (averaged over $M \in \{3, 4, \ldots, 7\}$) to build a τ-Index using all three partitioning schemes for the duration dimension, with varying number of vectors k. Two observations follow: first, we see that with increase in k, the preprocessing time for each partitioning scheme remains nearly constant while using the GeoLife dataset (Figure 20a), and increases very slowly for ElkDeer and TDrive datasets (Figures 20b and 20c). Although the heap data structure becomes more selective as k increases, and fewer points need to be processed when computing C_S, a large number of dwell region queries still need to be run to build the index. Overall, the higher selectivity does not suffice to make a measurable difference in the index build time. Next, in the GeoLife

dataset, we observe that the linear partitioning scheme results in a higher preprocessing time than the quadratic or the exponential schemes. This is because the cost of sorting entries per cell affects the preprocessing time more than the increase in k. In the linear scheme, the cells span equal temporal ranges, and with higher M, the number of entries per cell increases, in turn increasing the sorting overhead and thereby the average preprocessing time relative to the other two schemes. However, the rate of increase is linear in M.



(a) GeoLife Data (b) ElkDeer Data (c) TDrive Data

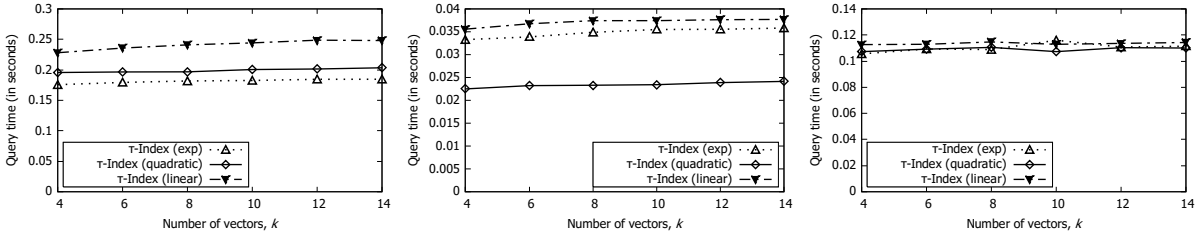
Fig. 21. Average preprocessing time for the τ-Index for varying the number of temporal partitions M, averaged $\forall k \in \{4, 6, \ldots, 14\}$.

Figure 21 shows the variation in τ-Index preprocessing time with the number of temporal partitions M using linear, quadratic and exponential partitioning schemes. The τ-Index build time first increases, and then becomes nearly constant for $M \geq 6$ for Elkdeer and TDrive datasets. However, on the GeoLife dataset while linear partitioning, the preprocessing time is high for lower values of M, decreases thereon, and then for $M \geq 6$, becomes nearly constant.

As M increases, the τ-Index preprocessing times are affected by two factors. First, the number of dwell region queries required to compute the τ-Index increases, as a query needs to be run per partition of the duration dimension. This increases the preprocessing time. Second, the number of entries in each cell decreases with M, thereby decreasing the cost of sorting and storing the entries in the cells. When using the quadratic and the exponential partitioning schemes, in the GeoLife dataset, the first factor dominates and the preprocessing time increases with the number of partitions as the cells of larger duration are added to the index. However, since the dataset has far fewer dwell regions of higher durations, corresponding cells also have fewer entries, so for $M \geq 6$, the additional entries in the these cells do not significantly add to the index build times, and we see the plateau. When using the linear partitioning scheme to build the τ-Index, the cells span equal ranges of duration. Having more smaller cells decreases the total preprocessing time slightly as the cost of sorting and storage decreases due to fewer entries.

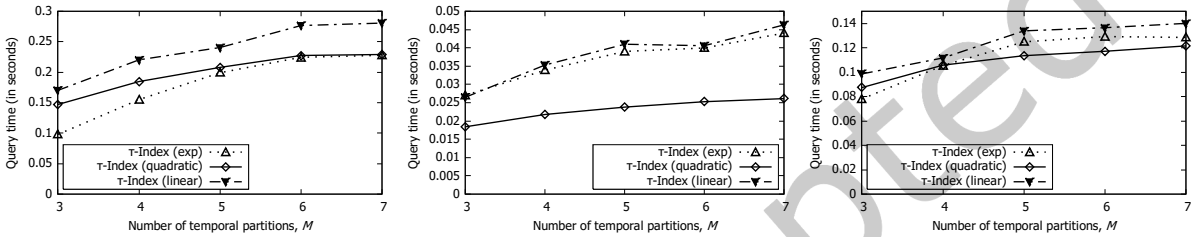
Conversely, the sampling frequency for ElkDeer and TDrive datasets is relatively low. Therefore, these datasets contain more dwell regions as the query duration increases, which increases the number of entries in the higher-duration cells. As the cost of sorting and storing cell entries increases, so does the preprocessing time.

7.2.2 Effect of partitioning schemes on τ-Index query evaluation time. The query evaluation time when using a τ-Index highly depends on the relative number of the entries in the cells of the τ-Index. Figure 22 shows the change in query time with change in the number of vectors, k. We notice that quadratic and exponential partitioning schemes perform better than the linear partitioning on the GeoLife dataset, the quadratic partitioning scheme performs better on the ElkDeer dataset, and all schemes perform similarly on the TDrive dataset. We also observe that the query evaluation time for all partitioning schemes remains nearly constant for any value of k. This is because the bulk of the query processing cost when using a τ-Index is disk I/O, so the better pruning of points afforded by high values of k does not help in answering queries faster.



(a) GeoLife Data (b) ElkDeer Data (c) TDrive Data

Fig. 22. Query evaluation time using the τ-Index and varying the number of vectors k, for queries with $r_q = \{0.7, 0.9, \ldots, 1.9\}$ miles, $\tau_q \in \{1, 3, 5, \ldots, 31\}$ hours for GeoLife and TDrive datasets, and $\tau_q \in \{1, 3, 5, \ldots, 31\}$ days for the ElkDeer dataset.



(a) GeoLife Data (b) ElkDeer Data (c) TDrive Data

Fig. 23. Query evaluation time using the τ-Index and varying the number of temporal partitions M, for queries with $r_q = \{0.7, 0.9, \ldots, 1.9\}$ miles, $\tau_q \in \{1, 3, 5, \ldots, 31\}$ hours for GeoLife and TDrive datasets, and $\tau_q \in \{1, 3, 5, \ldots, 31\}$ days for the ElkDeer dataset.

Figure 23 shows the query evaluation time for the τ indexes with all three partitioning schemes for different values of M. We noted in Section 7.2.1 that an increase in number of cell entries is directly proportional to the preprocessing and query times. The effect is clearly visible in our experiments, with increase in number of M, the query processing time increases, irrespective of partitioning scheme used. However, quadratic and exponential partitioning schemes using the GeoLife dataset perform better than the linear scheme with approximately 50% faster queries. This is because both of quadratic and exponential partitioning create more cells that span smaller durations, closer to the low duration dimension values than the linear partitioning. This means smaller cells hold roughly the same number of entries, and therefore can be used to answer the dwell region queries faster.

7.3 Scaling the Number of Trajectories

Having established the performance advantages of the τ-Index in our earlier experiments, we now evaluate its scalability as the number of trajectories grows to 500K. Real datasets of such size are generally unavailable, so we chose the road network of Riverside County, and generated synthetic datasets with 100K, 300K, and 500K trajectories, each covering a duration of one week, using the trajectory generator of [3]. This road network covers an area of 7,304 sq mi (166 mi $\times$ 44 mi), and has 92,607 nodes and 58,546 edges.

Dwell region query parameters must be carefully chosen to get meaningful dwell regions, and depend on factors such as object speed and sampling frequency. If the sampling frequency is low and the object is moving rapidly, two successively sampled points on its trajectory are likely to be far away from each other. Dwell regions of small query radius may be missed. For example, say that an object moving at an average speed of 20 mph is sampled every 20 seconds. The object travels about half a kilometer ($\approx$ 0.33 miles) in a minute, and we get 3 points per minute on the trajectory, with an average separation of 0.11 miles. If a dwell region query has $r_q = 0.33$

miles and $\tau_q = 1$ minute, we are likely to find some dwell regions, since successive object positions are within 0.33 miles during this minute. On the other hand, if the same trajectory is sampled every 90 seconds, the distance between two subsequent points can exceed $r_q = 0.33$ miles. No dwell region will be found for the same trajectory.

There is also the risk of generating too many dwell regions. Continuing with the same example, if the object moves at 20 mph along a straight line, and the sampling frequency is 20 seconds, every set of three successively sampled points will remain within 0.33 miles of each other, so that every point on the trajectory gives rise to a dwell region. These parameters identify too many dwell regions, all of them likely meaningless, as they simply reflect the default motion of the object.

Based on such considerations, we set the sampling frequency in our trajectory generator to 10 seconds, and the maximum speed to 80 mph. On average, with these parameters, the generated trajectories had speeds of ~ 20 mph, and between 168 and 170 points (see Table 5).

No. of trajectories	No. of spatial points	Average trajectory size
100K	~ 16 million	168
300K	~ 50 million	170
500K	~ 82 million	169

Table 5. Summary of synthetic trajectory datasets generated on the Riverside county road network with sampling frequency 10 seconds.

We used a quadratic partitioning scheme for the time dimension in the following experiments, since the results from Section 7.2.1 show that it has better preprocessing time than the other schemes on real datasets. We built the τ-Index using $k = 8$ vectors, and the number of temporal partitions, $M = 5$. In the real datasets the preprocessing time increases very slowly when $k \geq 8$ and $M \geq 5$. Given the relatively high sampling frequency (10 seconds), we consider the duration dimension for the τ-Index in minutes.

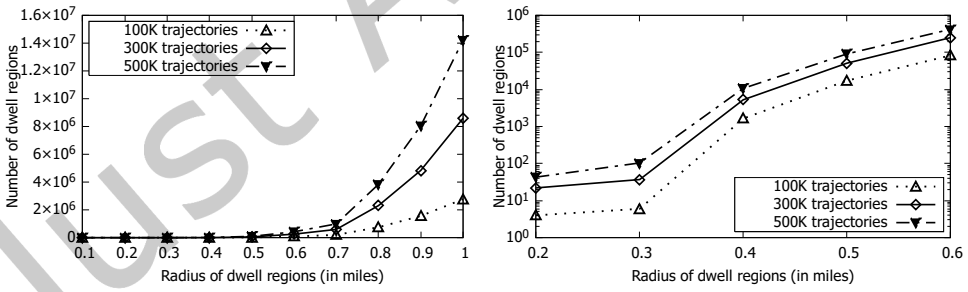


Fig. 24. Synthetic trajectory datasets. (a) Number of dwell regions for queries with $\tau_q = 9$ minutes and $0.1 \leq r_q \leq 1.0$ miles. (b) Drilling down for $0.2 \leq r_q \leq 0.6$ miles (The y-axis scale is logarithmic).

7.3.1 Number of dwell regions. We first examine the effect of query radius r_q on the number of dwell regions. For a given τ_q, a dwell region subsumes all dwell regions of smaller radius. Figure 24a shows the number of dwell regions for queries with $\tau_q = 9$ minutes, for $0.1 \leq r_q \leq 1.0$ miles. For a fixed τ_q, the number of dwell regions increases exponentially with query radius. Figure 24b drills down on Figure 24a, focusing on $0.2 \leq r_q \leq 0.6$ miles, where the same exponential increase is apparent (the y-axis scale is logarithmic). This result highlights

the importance of choosing a meaningful query radius (e.g. 0.3 miles), based on the sampling frequency and the number of trajectories in the dataset. Clearly, most of the dwell regions identified are uninteresting, arising from the relation between object behavior and query parameters, as we have discussed earlier.

7.3.2 τ-Index build time. We next examine the effect of the number of trajectories on the τ-Index build time. We built each τ-Index for the synthetic datasets with $k = 8$ and $M = 5$, using $\tau_{min} = 4$ minutes and $\tau_{max} = 36$ minutes. Each τ-Index incorporated 10 radii, following the linear partitioning scheme from $r_{min} = 0.1$ miles to $r_{max} = 1.0$ miles. We chose the linear partitioning scheme in the radius dimension so as to avoid highly skewed radius partitions (see Section 6.3).

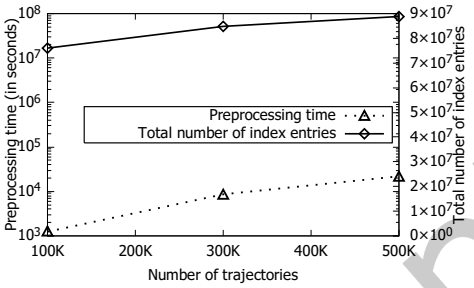


Fig. 25. Synthetic trajectory datasets. τ-Index preprocessing time, with $k = 8$ vectors, $M = 5$ temporal partitions and total number of index entries in the τ-Index.

Figure 25 shows that the τ-Index preprocessing time grows quadratically with the number of trajectories in the synthetic datasets. The number of trajectory points increases linearly with the number of trajectories, which are all sampled at the same frequency (Table 5). However, the index stores dwell regions, not trajectory points. Therefore, we examine how the number of index entries grows with the number of trajectories. Figure 25 also shows the total number of index entries in the τ-Index. The number of index entries increases with the increase in number of trajectories, which justifies the increasing preprocessing time with the increase in number of trajectories.

8 CONCLUSIONS

In this paper we have considered the dwell region query in spatio-temporal databases. This query identifies regions where a moving object stays within a certain distance for at least a certain duration. We have reduced the problem of identifying dwell regions to computing smallest enclosing circles (SEC) of subtrajectories of the moving object. We have also examined the related decision query that determines whether a dwell region (or dwell behavior) exists or not for a moving object's trajectory. We have first considered the online version of the problem and proposed a method that evaluates the decision query in logarithmic time, by maintaining the upper and lower bounds for the radius of the SEC of a subtrajectory. This also significantly reduces the number of spatial points that should be considered in order to approximate the radius of the SEC. In all experiments, we have observed that the upper and lower bounds suffice to answer on average 97.1% – 99.9% decision queries without computing the SEC. Moreover, to approximate the radius of the SEC we have to consider only 4.5% – 13.5% spatial points on average. As a result, our approach is capable of evaluating the online version of dwell region queries on hundreds of thousands of moving objects per second. We have also examined the offline version of the problem using archival datasets of trajectories. We have introduced two indexes that evaluate dwell region queries by looking into few candidate subtrajectories: the ρ-Index that indexes subtrajectories by radius ranges,

and the τ-Index further refines the approach to index subtrajectories by radius and duration ranges. Experiments have shown that using ρ-Index and τ-Index, we achieve at least an order of magnitude speedup and at least three orders of magnitude speedup respectively, over a baseline approach, for finding dwell regions. Moreover, the τ-Index offers orders of magnitude better preprocessing and query evaluation performance over the ρ-Index. Finally, experiments using synthetic datasets where the number of trajectories goes up to 500K, have validated the scalability of the τ-Index.

An interesting future research direction is to detect common dwell regions for groups of trajectories in both streaming and archived environments.

ACKNOWLEDGMENTS

We would like to thank Neal E. Young and Claire Mathieu for their useful suggestions and comments. This work was partially supported by NSF grants IIS-1527984, SES-1831615 and IIS-1901379.

REFERENCES

[1] Luis Otavio Alvares, Vania Bogorny, Bart Kuijpers, Jose Antonio Fernandes de Macedo, Bart Moelans, and Alejandro Vaisman. 2007. A Model for Enriching Trajectories with Semantic Geographical Information. In *GIS*. ACM, Article 22, 8 pages.
[2] Daniel Ashbrook and Thad Starner. 2003. Using GPS to Learn Significant Locations and Predict Movement across Multiple Users. *Personal Ubiquitous Comput.* 7, 5 (2003), 275–286.
[3] Thomas Brinkhoff. 2002. A Framework for Generating Network-Based Moving Objects. *GeoInformatica* 6, 2 (2002), 153–180.
[4] Maike Buchin, Anne Driemel, Marc van Kreveldz, and Vera Sacristán. 2010. An Algorithmic Framework for Segmenting Trajectories based on Spatio-Temporal Criteria. In *GIS*. ACM, 202–211.
[5] Xin Cao, Gao Cong, and Christian S. Jensen. 2010. Mining Significant Semantic Locations From GPS Trajectory. *Proc. VLDB Endow.* 3, 1–2 (2010), 1009–1020.
[6] Yang Cao, Jingling Yuan, Song Xiao, and Qing Xie. 2019. TPM: A GPS-based Trajectory Pattern Mining System. In *BESC*. IEEE, 1–4.
[7] Sanjay Chawla, Yu Zheng, and Jiafeng Hu. 2012. Inferring the root cause in road traffic anomalies. In *ICDM*. IEEE, 141–150.
[8] Muhammad Aamir Cheema, Ljiljana Brankovic, Xuemin Lin, Wenjie Zhang, and Wei Wang. 2010. Multi-guarded safe zone: An effective technique to monitor moving circular range queries. In *ICDE*. IEEE, 189 – 200.
[9] Lisi Chen, Shuo Shang, Christian S. Jensen, Bin Yao, and Panos Kalnis. 2020. Parallel Semantic Trajectory Similarity Join. In *ICDE*. IEEE, 997–1008.
[10] Kenneth L. Clarkson. 1995. Las Vegas Algorithms for Linear and Integer Programming When the Dimension is Small. *J. ACM* 42, 2 (1995), 488–499.
[11] Maria Luisa Damiani, Hamza Issa, and Francesca Cagnacci. 2014. Extracting Stay Regions with Uncertain Boundaries from GPS Trajectories: A Case Study in Animal Ecology. In *SIGSPATIAL*. ACM, 253–262.
[12] Martin E. Dyer and Alan M. Frieze. 1989. A randomized algorithm for fixed-dimensional linear programming. *Mathematical Programming* 44, 1-3 (1989), 203–212.
[13] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. 1996. A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise. In *KDD*. AAAI Press, 226–231.
[14] Daniel Fortin, Hawthorne L. Beyer, Mark S. Boyce, Douglas W. Smith, Thierry Duchesne, and Julie S. Mao. 2005. Wolves Influence Elk Movements: Behavior Shapes a Trophic Cascade in Yellowstone National Park. *Ecology* 86, 5 (2005), 1320–1330.
[15] Fosca Giannotti, Mirco Nanni, Fabio Pinelli, and Dino Pedreschi. 2007. Trajectory Pattern Mining. In *KDD*. ACM, 330–339.
[16] Michael Gowanlock and Henri Casanova. 2014. In-Memory Distance Threshold Similarity Searches on Moving Object Trajectories. *International Journal on Advances in Software* 7, 3-4 (2014), 617–631.
[17] Antonin Guttman. 1984. R-trees: A Dynamic Index Structure for Spatial Searching. In *SIGMOD*. ACM, 47–57.
[18] Marios Hadjieleftheriou, George Kollios, Dimitrios Gunopulos, and Vassilis J. Tsotras. 2003. On-Line Discovery of Dense Areas in Spatio-temporal Databases. In *SSTD*. Springer, 306–324.
[19] Hoyoung Jeung, Man Lung Yiu, Xiaofang Zhou, Christian S. Jensen, and Heng Tao Shen. 2008. Discovery of Convoys in Trajectory Databases. *Proc. VLDB Endow.* 1, 1 (2008), 1068–1080.
[20] Jianguo Jiang, Xiao guang Lv, Yanfang Zhang, and Siye Wang. 2018. Research on Hotspots Finding in Indoor Space Based on Regression Analysis. In *ICCDE*. ACM, 98–102.
[21] Peiquan Jin, Jiang Du, Chuanglin Huang, Shouhong Wan, and Lihua Yue. 2015. Detecting Hotspots from Trajectory Data in Indoor Spaces. In *DASFAA*. Springer, 209–225.

[22] Alexandros Kontarinis, Karine Zeitouni, Claudia Marinica, Dan Vodislav, and Dimitris Kotzinos. 2021. Towards a semantic indoor trajectory model: application to museum visits. *GeoInformatica* 25, 2 (2021), 311–352.
[23] John Krumm and Eric Horvitz. 2006. Predestination: Inferring Destinations from Partial Trajectories. In *UbiComp*. Springer-Verlag, 243–260.
[24] Jae-Gil Lee, Jiawei Han, Xiaolei Li, and Hector Gonzalez. 2008. TraClass: Trajectory Classification Using Hierarchical Region-Based and Trajectory-Based Clustering. *Proc. VLDB Endow.* 1, 1 (2008), 1081–1094.
[25] Jae-Gil Lee, Jiawei Han, and Kyu-Young Whang. 2007. Trajectory Clustering: A Partition-and-Group Framework. In *SIGMOD*. ACM, 593–604.
[26] Quannan Li, Yu Zheng, Xing Xie, Yukun Chen, Wenyu Liu, and Wei-Ying Ma. 2008. Mining User Similarity Based on Location History. In *GIS*. ACM, Article 34, 10 pages.
[27] Zhenhui Li, Bolin Ding, Jiawei Han, and Roland Kays. 2010. Swarm: Mining Relaxed Temporal Moving Object Clusters. *Proc. VLDB Endow.* 3, 1-2 (2010), 723–734.
[28] Zhenhui Li, Bolin Ding, Jiawei Han, Roland Kays, and Peter Nye. 1998. Mining Periodic Behaviors for Moving Objects. In *KDD*. ACM, 1099–1108.
[29] Lin Liao, Donald J. Patterson, Dieter Fox, and Henry Kautz. 2007. Learning and Inferring Transportation Routines. *Artif. Intell.* 171, 5–6 (2007), 311–331.
[30] Yanchi Liu, Chuanren Liu, Nicholas Jing Yuan, Lian Duan, Yanjie Fu, Hui Xiong, Songhua Xu, and Junjie Wu. 2014. Exploiting Heterogeneous Human Mobility Patterns for Intelligent Bus Routing. In *ICDM*. IEEE, 360–369.
[31] Jed A Long, Stephen L Webb, Trisalyn A Nelson, and Kenneth L Gee. 2015. Mapping areas of spatial-temporal overlap from wildlife tracking data. *Moving Ecology* 3, 1, Article 38 (2015), 14 pages.
[32] Nimrod Megiddo. 1982. Linear-Time Algorithms for Linear Programming in $\mathbb{R}^3$ and Related Problems. In *FOCS*. IEEE, 329 – 338.
[33] Raul Montoliu, Jan Blom, and Daniel Gatica-Perez. 2013. Discovering Places of Interest in Everyday Life From Smartphone Data. *Multim. Tools Appl.* 62, 1 (2013), 179–207.
[34] Jinfeng Ni and Chinya V. Ravishankar. 2007. Pointwise-Dense Region Queries in Spatio-temporal Databases. In *ICDE*. IEEE, 1066–1075.
[35] Panagiotis Nikitopoulos, Aris-Iakovos Paraskevopoulos, Christos Doulkeridis, Nikos Pelekis, and Yannis Theodoridis. 2018. Hot Spot Analysis over Big Trajectory Data. In *Big Data*. IEEE, 761–770.
[36] Kyosuke Nishida, Hiroyuki Toda, and Yoshimasa Koike. 2015. Extracting Arbitrary-Shaped Stay Regions from Geospatial Trajectories with Outliers and Missing Points. In *IWCTS*. ACM, 1–6.
[37] Andrey Tietbohl Palma, Vania Bogorny, Bart Kuijpers, and Luis Otavio Alvares. 2008. A Clustering-Based Approach for Discovering Interesting Places in Trajectories. In *SAC*. ACM, 863–868.
[38] Christine Parent, Stefano Spaccapietra, Chiara Renso, Gennady Andrienko, Natalia Andrienko, Vania Bogorny, Maria Luisa Damiani, Aris Gkoulalas-Divanis, Jose Macedo, Nikos Pelekis, Yannis Theodoridis, and Zhixian Yan. 2013. Semantic Trajectories Modeling and Analysis. *ACM Comput. Surv.* 45, 4, Article 42 (2013), 32 pages.
[39] Donald J. Patterson, Lin Liao, Dieter Fox, and Henry A. Kautz. 2003. Inferring High-Level Behavior from Low-Level Sensors. In *UbiComp*. Springer, 73–89.
[40] Nikos Pelekis, Ioannis Kopanakis, Gerasimos Marketos, Irene Ntoutsi, Gennady Andrienko, and Yannis Theodoridis. 2007. Similarity Search in Trajectory Databases. In *TIME*. IEEE, 129–140.
[41] Rafael Pérez-Torres, César Torres-Huitzil, and Hiram Galeana-Zapién. 2016. Full On-Device Stay Points Detection in Smartphones for Location-Based Mobile Applications. *Sensors* 16, 10 (2016), 1693.
[42] Mary Rowland. 1998. http://www.fs.fed.us/pnw/starkey/.
[43] Raimund Seidel. 1990. Linear Programming and Convex Hulls Made Easy. In *SCG*. ACM, 211–215.
[44] Shuo Shang, Kai Zheng, Christian S Jensen, Bin Yang, Panos Kalnis, Guohe Li, and Ji-Rong Wen. 2015. Discovery of Path Nearby Clusters in Spatial Networks. *TKDE* 27, 6 (2015), 1505–1518.
[45] Arun Sharma and Shashi Shekhar. 2022. Analyzing Trajectory Gaps to Find Possible Rendezvous Region. *ACM Trans. Intell. Syst. Technol.* 13, 3, Article 36 (2022), 23 pages.
[46] M Reaz Uddin, Chinya V Ravishankar, and Vassilis J Tsotras. 2011. Finding Regions of Interest from Trajectory Data. In *MDM*. IEEE, 39–48.
[47] M Reaz Uddin, Chinya V Ravishankar, and Vassilis J Tsotras. 2012. Online Identification of Dwell Regions for Moving Objects. In *MDM*. IEEE, 248–257.
[48] Reaz Uddin, Michael N. Rice, Chinya V. Ravishankar, and Vassilis J. Tsotras. 2017. Assembly Queries: Planning and Discovering Assemblies of Moving Objects Using Partial Information. In *SIGSPATIAL*. ACM, Article 24, 10 pages.
[49] Marcos R. Vieira, Petko Bakalov, and Vassilis J. Tsotras. 2009. On-Line Discovery of Flock Patterns in Spatio-Temporal Data. In *GIS*. ACM, 286–295.
[50] Marcos R. Vieira, Petko Bakalov, and Vassilis J. Tsotras. 2010. Querying Trajectories Using Flexible Patterns. In *EDBT*. ACM, 406–417.
[51] Ling-Yin Wei and Wen-Chih Peng Yu Zheng. 2012. Constructing Popular Routes from Uncertain Trajectories. In *KDD*. ACM, 195–203.

[52] Emo Welzl. 1991. Smallest enclosing disks (balls and ellipsoids). In *New Results and New Trends in Computer Science*. Springer, 359–370.
[53] Kexin Xie, Ke Deng, and Xiaofang Zhou. 2009. From Trajectories to Activities: A Spatio-Temporal Join Approach. In *LBSN*. ACM, 25–32.
[54] Yutaka Yanagisawa, Jun ichi Akahani, and Tetsuji Satoh. 2003. Shape-Based Similarity Query for Trajectory of Mobile Objects. In *MDM*. Springer, 63–77.
[55] Jing Yuan, Yu Zheng, and Xing Xie. 2012. Discovering regions of different functions in a city using human mobility and POIs. In *KDD*. ACM, 186–194.
[56] Jing Yuan, Yu Zheng, Xing Xie, and Guangzhong Sun. 2013. T-Drive: Enhancing Driving Directions with Taxi Drivers' Intelligence. *TKDE* 25, 1 (2013), 220–232.
[57] Bolong Zheng, Nicholas Jing Yuan, Kai Zheng, Xing Xie, Shazia Sadiq, and Xiaofang Zhou. 2015. Approximate keyword search in semantic trajectory database. In *ICDE*. IEEE, 975–986.
[58] Vincent Wenchen Zheng, Yu Zheng, Xing Xie, and Qiang Yang. 2010. Collaborative Location and Activity Recommendations With GPS History Data. In *WWW*. ACM, 1029–1038.
[59] Yu Zheng. 2007. http://research.microsoft.com/en-us/projects/geolife/.
[60] Yu Zheng. 2019. T-Drive trajectory data sample. Retrieved from UCR-STAR https://star.cs.ucr.edu/?TDrive&d.
[61] Yu Zheng, Yukun Chen, Quannan Li, Xing Xie, and Wei-Ying Ma. 2010. Understanding transportation modes based on GPS data for Web applications. *ACM Trans. Web* 4, 1, Article 1 (2010), 36 pages.
[62] Yu Zheng, Lizhu Zhang, Xing Xie, and Wei-Ying Ma. 2009. Mining Correlation Between Locations Using Human Location History. In *GIS*. ACM, 472–475.
[63] Yu Zheng, Lizhu Zhang, Xing Xie, and Wei-Ying Ma. 2009. Mining Interesting Locations and Travel Sequences from GPS Trajectories. In *WWW*. ACM, 791–800.