2011 12th IEEE International Conference on Mobile Data Management

Finding Regions of Interest from Trajectory Data

Md Reaz Uddin, Chinya Ravishankar, Vassilis J. Tsotras
University of California, Riverside, CA, USA
{uddinm,ravi,tsotras}@cs.ucr.edu

***Abstract*— We show how to find regions of interest (ROIs) in trajectory databases. ROIs are regions where a large number of moving objects remain for at least a given time interval. Previous techniques use somewhat restrictive definitions for ROIs, and are parameter-dependent. They require sequential scanning of the entire dataset to find ROIs when the ROI parameters change. Our approach is parameter independent, so that the user can quickly identify ROIs under different parametric definitions without rescanning the whole database. We also generalize ROIs to be regions of arbitrary shape of some predefined density. We have tested our methods with large real and synthetic datasets to test the scalability and verify the output of our methods. Our methods give meaningful output and scale very well.**

I. Introduction

The widespread use of GPS-enabled devices has enabled many applications that generate and maintain data in the form of *trajectories* (e.g., [1], [2], [3], [4]). Novel applications [5], [6], [7], [8] allow users to manage, store, and share trajectories in the form of GPS logs, and find travel routes, interesting places, or other people interested in similar activities.

Other research efforts have been geared towards understanding and extracting people's activities from trajectory databases. Examples include querying for a certain sequence of activities during traveling [9], mining similarity between travelers based on their activity sequence [10], inferring popular locations from GPS traces [11], etc. Typically, these queries assume that information about the locations of specific activities is provided as a set of Regions of Interest.

There is recent work on discovering Regions of Interest (ROIs) from trajectory databases [10], [11], [12]. However, these methods are aimed at using ROIs to identify user travel similarity, top-k interesting locations, etc. They identify "stay points" (equivalent to ROIs), where a stay point is an (x, y) average of the points of a subtrajectory in which the object moves less than a prespecified distance threshold δ and takes longer than a prespecified time threshold τ. If either δ or τ changes, the entire trajectory database must be re-scanned. In contrast, our work removes this important limitation.

It is more intuitive to define ROIs in terms of speed. If an object takes at least time τ to travel at most distance δ, it maintains an average speed no more than $\frac{\delta}{\tau}$ for at least time τ. In our framework, we actually use a speed *range* to define ROIs, as this leads to a more generic definition. Further, we introduce the notion of *trajectory density* to define ROIs. A region is *dense* if the number of objects per unit area is no less than a pre-specified threshold. In summary, our ROI definition uses (1) a range of speed that an object maintains while in an ROI (2) a minimum duration of staying in an ROI area and (3) the density of objects in that area.

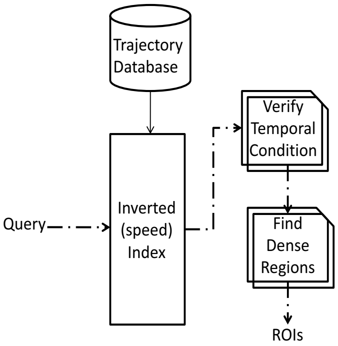


Fig. 1. Framework for Discovering ROIs

We build an index on object speeds to avoid scanning the whole database. Given a range or a particular speed, we first retrieve trajectory segments with that speed using this index. We then verify the minimum stay duration condition. Objects that fulfill the speed and duration condition are *candidate objects*. Finally, we identify dense regions of candidate objects. For this we extend the pointwise density method [13]. Figure 1 shows our proposed framework to discover ROIs.

Our contributions are summarized as follows. We provide:

- a generalized ROI definition for trajectories,
- a framework and several approaches to find ROIs efficiently, and
- an extensive experimental evaluation of the proposed methods using one synthetic and three real datasets.

The remainder of the paper is organized as follows: Section II presents the basic definitions and formal description of the regions of interest while Section III provides an overview of related works. Section IV describes the framework for storing trajectories in order to efficiently find ROIs; the proposed methods to find regions of interest are described in Section V. Section VI presents the experimental evaluation and Section VII concludes the paper.

II. Background

We begin with some definitions.

Definition 1: A ***trajectory*** is a finite sequence of (x_i, y_i, t_i) triples. The $x_i, y_i \in \mathbb{R}^2$ are spatial coordinates, and the $t_i \in \mathbb{R}^+$, are timestamps, with $t_i < t_{i+1}$ for $i = 0, 1, \ldots, n-1$.

978-0-7695-4436-6 2011

DOI 10.1109/MDM.2011.12

Each (x_i, y_i) pair represents the position recorded of a moving object at time t_i (typically from a GPS enabled device). Each trajectory has a unique trajectory ID (TID).

Definition 2: A ***trajectory segment*** is a straight line between two consecutive tuples $(x_i, y_i, t_i), (x_{i+1}, y_{i+1}, t_{i+1})$ of the same trajectory, where $i \in \mathbb{N}_0$.

Definition 3: A **subtrajectory** of length m of a trajectory $T = (x_0, y_0, t_0), \ldots, (x_n, y_n, t_n)$, is a subsequence $T' = (x_i, y_i, t_i), \ldots, (x_{m+i-1}, y_{m+i-1}, t_{m+i-1})$, of m contiguous trajectory segments, where $i \geq 0, m \leq n$.

A single trajectory segment is a subtrajectory of length one. In the rest of the paper we use *trajectory segment*, *line segment* or *line* interchangeably.

A. Defining Regions of Interest

Conceptually, an ROI is intended to be a region where moving objects pause or wait in order to complete activities that are difficult or impossible to carry out while in motion. Examples of ROIs are restaurants, museums, parks, places of work, and so on. Generally, individual trajectories display idiosyncrasies, so ROIs are best defined in terms of collective behaviors of a collection of trajectories. That is, a collection of trajectories is needed to identify a location as an ROI.

One simplistic approach to define ROIs is to consider places where many trajectories intersect. However, not all intersections may be ROIs. For example, it may not be appropriate to declare a busy road intersection as an ROI. The duration of an object's stay in a location is important in filtering out spurious ROIs, so we will require a *minimum stay duration* for objects at ROIs. Nevertheless, if an object spends a long time in a large spatial region, a city, say, then that large region should not be considered as an ROI either. Hence, we must also consider the geographic extent of the object's movement, that is, the maximum area within which an object remains (or the maximum distance traveled by an object) during the minimum stay duration.

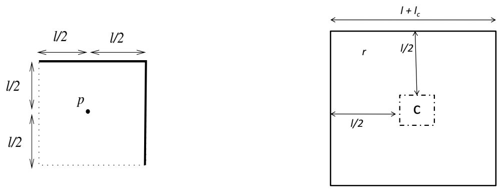
Since the number of objects visiting a potential ROI is also important, we consider the density of candidate objects in such a region. The problem of finding ROIs can then be viewed as that of finding dense regions of candidate objects.

B. Identifying Point-Wise Dense Regions

We identify dense regions adapting the *point-wise dense region* approach of [13]. In this approach, a region $R \subset \mathbb{R}^2$ is dense if every point in R has a neighborhood which contains a sufficient number of objects. This density approach removes various anomalies (e.g., answer loss, lack of local density guarantee, etc.) that other density computation methods, [14], [15], have. We thus adapt the following definitions from [13]:

Definition 4: The ***l*-square neighborhood** of a point $p \in \mathbb{R}^2$ is a square with edge length l centered at p, including top and right edges, but excluding bottom and left edges. Figure 2(a) shows the l-square neighborhood of a point p. We assume $l \geq l_{\min}$, where $l_{\min}$ is predefined.

Since we must find the density around trajectory segments, we extend this definition by defining l-neighborhoods around line segments and rectangles.



(a) l-square neighborhood of a point p.

(b) l-square neighborhood of the rectangle c.

Fig. 2.

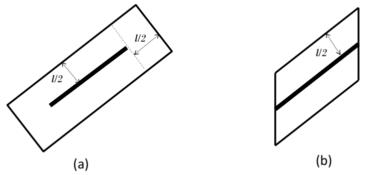


Fig. 3. l-neighbor regions.

Definition 5: The ***l*-rectangle neighborhood** r of a rectangle c with lower-left and upper-right corners at (x_l, y_b) and (x_r, y_t) is the rectangle (Figure 2(b)) with left-bottom corner ($x_l - \frac{l}{2}$, $y_b - \frac{l}{2}$) and right-top corner ($x_r + \frac{l}{2}$, $y_t + \frac{l}{2}$). If c is a square, r is the l-square neighborhood of c.

Definition 6: The ***l*-rectangle neighborhood of a line segment** is the rectangle with edge lengths l and $l + |L|$. The edges parallel with the line L are $l/2$ apart L and have length $l + |L|$ while the other edges are $l/2$ apart from the end points of L having length l.

Figure 3(a) illustrates this idea. Figure 3(b) shows a variant of this definition that is useful in simplifying the evaluation of certain integrals while computing dense regions. Here, two rectangle boundaries are parallel to the coordinate axes.

Definition 7: A point p in a region is ***dense*** if at least N different trajectories pass through the l-square neighborhood of p. The thresholds l and N are specified by the ROI query.

Given thresholds l, N, τ, we consider R to be an ROI if at least N objects remain for time τ in the l-square neighborhood of every point $p \in R$.

We note that the behavior of an object within a region of interest can be described in terms of its speed and duration of stay. If the object remains within the l-square neighborhood of a point for time at least τ, the net speed of the object (in terms of its net displacement) can not exceed $\sqrt{2}l/\tau$ during the time interval τ. In this paper, therefore, we define a region of interest in terms of speed.

Definition 8: A region R is a ***region of interest*** if every point $p \in R$ has an l-square neighborhood containing segments from at least N distinct trajectories with object speeds in the range $[s_1, s_2]$, and each such object remains in R for at least time τ before leaving R. The parameters l, N, τ, s_1, s_2 are user-defined.

Our definition also supports timestamps, i.e. weekends or weekdays, lunch or dinner time, etc. This allows the user to distinguish between ROIs with different semantics. ROIs found with long stay duration on the weekends have different semantics than those found on weekdays with short stay duration.

III. Related Work

Retrieving semantic information from trajectory databases has attracted much research attention. In [9], [16] ROI information is given in a relational database and a join operation between trajectory and ROI relations is performed to evaluate activity sequence queries. [16] and [9] assume 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 Δ, then the portion of the trajectory inside region Δ is considered as a *stop* area in that trajectory. These stops are similar to ROIs. Nevertheless, these approaches do not discover new ROIs as they consider only the application specified regions.

Recently, various works on discovering ROIs have appeared, [10], [11], [12], [17], [18], and are discussed below.

In [10], [11] the notion of "stay point" is presented using a maximum distance threshold D_{threh} and a minimum duration threshold T_{threh}. In particular subtrajectories are identified that take at least T_{threh} duration to travel no more than D_{threh} distance. A fixed pair of values (e.g., D_{threh} = 200m and T_{threh} = 30min) is considered for finding these subtrajectories. The (x, y) points of these subtrajectories are then averaged to identify stay points (one stay point for each subtrajectory). Note that, these stay points might not be on a trajectory (Figure 4). Density based clustering methods are then applied to group spatially collocated stay points. Each cluster is called a "stay region". These stay regions are then used to find similar travel sequences [10], top n interesting locations [11], [12]. However, reducing a subtrajectory into a particular point (possibly not on the trajectory) leads to possible loss of information. For example, if density based methods are applied on stay points to identify stay regions they might generate false negatives. Stay points (solid circles) shown in Figure 4 are obtained by taking the average of points in the low speed part of each trajectory. These stay points are too far from each other to form a cluster although there is a dense region (the grey region) of slow moving objects.

[12] takes the first of the two contiguous trajectory points that are logged more than T_{threh} time apart (the empty circles in Figure 4). These stay points are always on the trajectory but still subtrajectories are reduced to a single point and thus this approach also suffers the above problem. In addition, [12] does not consider the situation when an object is moving slowly or stopped but the GPS is frequently recording its positions.

In our approach we find dense regions considering the whole low speed subtrajectories instead of particular points and thus overcome the above problems. We achieve this by identifying as dense points, those points which have a certain number of trajectories in their predefined neighborhood.

In [17] ROIs are discovered for each individual trajectory instead of considering all trajectories and identifying commonly interesting places. The DBSCAN method [19] 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).

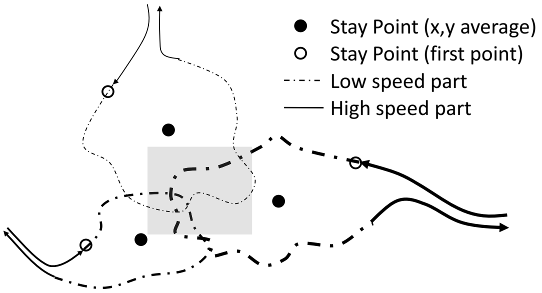


Fig. 4. Problem of density based clustering methods for trajectories.

The above methods do not index the data and so they need to rescan the whole database for every different set of values of parameters. If these approaches wanted to index the data to retrieve subtrajectories with arbitrary combination of maximum distance traveled and minimum stay duration then they would have needed an index with all possible combinations of values of D_{threh} and T_{threh}, which is not practical. We instead define stay points in terms of the speed of the object and index the trajectory database for speed values. With the speed index, we do not need to access the whole database for every different query: only the low speed (as specified by the query) trajectory segments are accessed.

[18] presents an approach to mine common sequence of locations, ROIs, visited with similar travel times between them. An example of such a sequence is 'Railway Station $\longrightarrow^{15min}$ Book Store $\longrightarrow^{30min}$ University', which says Railway Station to Book Store to University is a common travel sequence with travel time between them 15min and 30min respectively. In this work ROI implies dense regions which are visited by a certain fraction (i.e. 10%) of all trajectories. Dense regions are identified by discretization of the space into grids, which can also introduce false negatives [13] i.e., when a dense region spans over multiple cells but none of those cells are individually dense. Moreover, considering all segments of all trajectories will identify places which are not ROIs, i.e. road intersections. Other than density, this method is different from ours because it does not identify ROIs with query specified parameter values.

To summarize, our approach has the following characteristics: 1) it does not assume any a priori knowledge about ROIs, 2) it can identify ROIs for arbitrary values of parameters without rescanning the whole database 3) it identifies commonly interesting places by also considering the number of objects that visited the place.

IV. Indexing Trajectory Segments by Speed

Typically, objects in an ROI will maintain very low (or zero) speed. Hence, if we can quickly retrieve and analyze low speed trajectory segments, we can reduce query costs significantly.

Algorithm 1 BuildIndex($\mathcal{T}$:Dataset, $\mathcal{R}$:Ranges)

```
1:  for each trajectory $T \in \mathcal{T}$ do
2:      $\rho_{prev} = -1$
3:      $start = 1$
4:
5:      for each segment $(p_i, p_{i+1} \in T)$ do
6:          $\sigma$ =Speed($p_i, p_{i+1}$)
7:          $\rho$ =Range($\sigma$, $\mathcal{R}$)
8:          if $\rho \neq \rho_{prev}$ then
9:              $length = i - start$
10:             $ptr$ =MakePtr()
11:             $e$ =indexEntry($T.ID$, $start$, $length$, $ptr$)
12:             insertIntoBucket($\rho$, $e$)
13:
14:             $start = i$
15:         end if
16:         $\rho_{prev} = \rho$
17:     end for
18: end for
```

Let $s_{\max}$ and $s_{\min}$ be the maximum and minimum speeds specifiable in an ROI query. We partition the speed values into *index ranges* $\mathcal{R} = [s_{\min}, s_1), [s_1, s_2), \ldots, [s_{n-1}, s_{\max})$. These ranges can be of arbitrary length. We maintain one bucket for each index range, with bucket B_i holding trajectory segments with speed range $[s_i, s_{i+1})$.

We consider the segments for trajectory sequentially, and compute speeds assuming linear motion between two successive timestamps. If a series of consecutive segments fall within the same speed range, we combine them into one subtrajectory, and insert it into the index as one entry. Thus each entry in an index bucket points to a subtrajectory all of whose segments fall into within the speed range of the bucket. Figure 5 illustrates how subtrajectories are assigned to different buckets. The dotted and dashed lines show the subtrajectories which are contiguous parts of the same trajectory. The pseudo code for building the speed index appears in algorithm 1. Lines 6 and 7 calculate the speed of a segment and decide which of the index ranges contains it. If the speed range is same as that of the previous segment then we proceed to the next segment and so on. Otherwise, a new entry, e, is created which points to the last subtrajectory with same speed range; e is then inserted into the appropriate bucket (lines 8 to 15).

We assume trajectories are sorted according to TID, so that subtrajectories in the buckets are also sorted according to TID. Having TID sorted entries in the buckets allows to perform a merge join to reconstruct trajectories from these buckets. When new trajectories are added to the database the index can easily be updated using the above algorithm.

Finding trajectory segments having speed within range $[s_1, s_2)$ is straightforward. Every bucket whose speed range overlaps with the range $[s_1, s_2)$ is accessed. If the speed range of a bucket B_i is completely contained within the query speed range then all subtrajectories of B_i are considered. If there is partial overlap, the subtrajectories of B_i are checked for containment of speed within the query range.

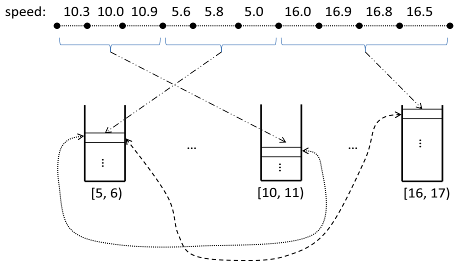


Fig. 5. Index Structure.

V. Finding Regions of Interest

We find ROIs in three steps. First, we retrieve the appropriate buckets from the index. In the second step, we collect subtrajectories spanning multiple buckets by performing a merge-join, and check the stay durations. In the third step, we find regions with line segment density N/l^2, where each of N segments has to be from different trajectories.

It is straightforward to retrieve the segments falling into a given speed range $[s_1, s_2)$ using the speed index. No further discussion is needed.

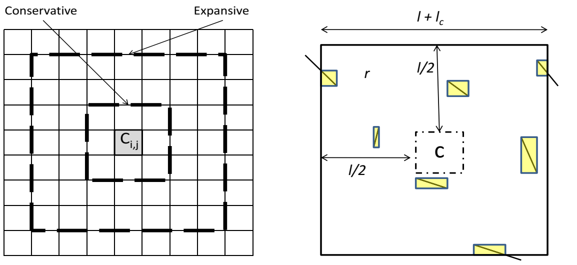
A. Step 2: Verifying the Duration Condition

In this step, we consider only the buckets obtained from the previous step. To verify the duration condition for each trajectory we must join subtrajectories with same TID from different buckets. Let the query speed range include buckets B_i and B_j, and let $S_i \in B_i$ and $S_j \in B_j$ be subtrajectories. Let the start and end timestamps for S_i and S_j be $[t_{i1}, t_{i2}]$ and $[t_{j1}, t_{j2}]$ respectively. If S_i and S_j have the same TID and $t_{i2} = t_{j1}$ or $t_{i1} = t_{j2}$, then S_i and S_j should be merged into a single subtrajectory. The object's stay duration is the interval between the first and the last timestamps of the merged subtrajectory. We discard all subtrajectories with stay duration less than τ after merge, since they do not fulfil the stay duration condition. Since we have a TID-sorted list in each bucket we need one pass over every bucket entry. The segments that belong to a subtrajectory with stay duration τ or more are input to the next step.

In addition to minimum stay duration, our implementation also supports other temporal conditions, such as time intervals and weekdays/weekends. For example, ROIs during any weekday with $\tau = 15$ to 30 minutes, carry different semantics than those found in the afternoon or evening of any weekend, with a few hours of stay duration.

B. Step 3: Finding Dense Regions

This step involves finding points p whose l^2-neighborhood contains at least N distinct trajectories. For our purpose we extend the Pointwise Dense Region (PDR) method [13] which was originally presented for point objects. We extend those techniques here for line segments. The work in [13] describes two variations: (1) an exact, and (2) an approximate method.



(a) Conservative and Expansive neighborhood of a cell $c_{i,j}$.

(b) MBRs in the l-square neighborhood of the cell c.

Fig. 6.

1) Exact PDR Method: The spatial region is assumed to be a $L \times L$ square area. This space is partitioned into $m \times m$ grid, with cell width $l_c = \frac{L}{m}$. Here, m must be such that $l_c \leq \frac{l_{min}}{2}$ where $l_{min} \leq l$. For each cell, $c_{i,j}$ where $1 \leq i, j \leq m$, we maintain a histogram. Initially all histogram values are set to zero. The histogram value for each cell is increased by one, for each distinct overlapping trajectory. We index the trajectory segments obtained from the previous step using an R*-tree [20]. The cost for building such an R*-tree is included in the query cost, which is, according to our experimental evaluations, quite small. For each cell we perform an R*-tree search to determine the number of overlapping trajectories. Histogram values for all the cells form an $m \times m$ matrix, which we call a *histogram matrix*.

PDR [13] is designed for moving objects, and evaluates predictive queries about dense regions at a future timestamp. Hence, the numbers and positions of objects could be different for different queries. As a result, histogram values must be computed for every different query. However, we evaluate the query on historical data (GPS traces), and can calculate histogram values once, and use to evaluate queries. That is, to speed up queries, we pre-compute a histogram matrix for every index range and then use these to answer queries with any arbitrary speed range. As new trajectories are added to the database, the histogram values and the speed index are both updated. We assume that the data is always up-to-date.

Calculating the histogram matrix for a query speed range requires adding histogram matrices whose speed ranges overlap with the query range. If the upper and lower limits of the query range exactly match the limits of index ranges then no histogram values are calculated. Otherwise we have to calculate one or two histogram matrices for a very small size of data. The following example illustrates the idea.

Let the query range be $[s_{q1}, s_{q2})$, and the index ranges be $[s_0, s_1), \ldots, [s_i, s_{i+1}), \ldots, [s_j, s_{j+1}), \ldots, [s_{n-1}, s_n)$. If $s_{q1} = s_i$ and $s_{q2} = s_j$ where $0 \leq i \leq n-1$, $1 \leq j \leq n$, and $i < j$, then we add up the histograms for the ranges $[s_i, s_{i+1}), \ldots, [s_{j-1}, s_j)$. However, if $s_{i-1} < s_{q1} < s_i$, $s_j < s_{q2} < s_{j+1}$, where $1 \leq i \leq n-2$, $2 \leq j \leq n-1$, and $i < j$, then we have to calculate histogram values for the ranges $[s_{q1}, s_i)$ and $[s_j, s_{q2})$ in addition to adding up histogram matrices for the speed ranges $[s_i, s_{i+1})$, $\ldots$, $[s_{j-1}, s_j)$.

a) The Filtering Step: Let $\eta_l = \lfloor \frac{l}{2l_c} \rfloor$, $\eta_h = \lceil \frac{l}{2l_c} \rceil$. We use the following definitions from [13]:

Definition 9: The ***conservative neighborhood*** of a cell $c_{i,j}$ is the union of grid cells $c_{u,v}$ such that $i - \eta_l < u < i + \eta_l$ and $j - \eta_l < v < j + \eta_l$.

The l-square neighborhood of any point inside $c_{i,j}$ fully contains its conservative neighborhood, $C_{i,j}$.

Definition 10: The ***expansive neighborhood*** of a cell $c_{i,j}$ is the union of grid cells $c_{u,v}$ such that $i - \eta_h \leq u \leq i + \eta_h$ and $j - \eta_h \leq v \leq j + \eta_h$.

The l-square neighborhood of any point inside $c_{i,j}$ is fully contained in its expansive neighborhood, $E_{i,j}$. Figure 6(a) shows the conservative and expansive neighborhood for a cell $c_{i,j}$, with $\eta_l = 2$ and $\eta_h = 3$.

If the conservative neighborhood of any cell overlaps with N trajectories, then all points inside cell $c_{i,j}$ are guaranteed to be dense, and $c_{i,j}$ is accepted as a dense cell. On the other hand, if the expansive neighborhood of any cell overlaps less than N line segments, then no point in cell $c_{i,j}$ is dense, and $c_{i,j}$ is rejected. Cells that are neither accepted nor rejected are candidate cells. We use the *FilterQuery* algorithm from [13] to identify accepted, rejected and candidate cells. In our case $\rho l^2 = N$ and there is no query timestamp, q_t. Candidate cells are further analyzed to identify dense points inside them.

b) Refinement Step: To identify dense points in a candidate cell $c_{i,j}$ first, we will find all segments that overlap with the l-square neighborhood, r, of the cell $c_{i,j}$.

We do an R*-tree search to retrieve segments overlapping with r. We retrieve only the portion of a line that overlaps with r. Each segment is represented by its MBR. Figure 6(b) shows the region r and the MBRs of the overlapping line segments. We sort the MBRs of line segments according to the x coordinate of their left-bottom corner.

In the refinement step, an *l-band* is swept along the X axis for each candidate cell $c_{i,j}$. An l-band is a rectangle with width l and height $l + y_t - y_b$. The position of the l-band is identified by the position of its vertical median. The plane sweep algorithm along the X-axis starts with the l-band's vertical median at the left edge of $c_{i,j}$ and stops when it touches the right edge of $c_{i,j}$. Let L be the sorted list of MBRs that overlap the *l-band* at the beginning. As the l-band is swept, when the right edge of the *l-band* touches the left edge of an MBR we insert it into the list L. When the left edge of the *l-band* touches the right edge of an MBR we delete it from L. The x coordinates of the vertical center line when any edge of l-band touches any edge of an MBR are the stopping points. Instead of sweeping through all the x coordinates it is sufficient to consider only the stopping points.

Algorithm 2 describes the plane sweep along the X-axis. In this algorithm the l-band is placed at each stopping point and the left and right edges of the candidate cell . If the l-band overlaps at least N objects then SweepY is called to identify dense regions, where l-square neighborhood of each point overlaps N lines. Plane sweep along Y axis proceeds in the same way. An l-square is swept instead of an l-band.

Algorithm 2 RefineQuery(N, $c_{i,j}$)

```
$S$ = MBRs in $l$-square neighborhood of $c_{i,j}$
sort $S$ according to the left x-coordinate, $x'_l$, of MBRs
stopX = $x_l$, $x_r$
for each MBR($x'_l$, $y'_b$, $x'_r$, $y'_t$) in $S$ do
    insert $x'_l - \frac{l}{2}$ if $x'_l - \frac{l}{2} \in [x_l,x_r]$
    insert $x'_r - \frac{l}{2}$ if $x'_r - \frac{l}{2} \in [x_l,x_r]$
    insert $x'_l + \frac{l}{2}$ if $x'_l + \frac{l}{2} \in [x_l,x_r]$
    insert $x'_r + \frac{l}{2}$ if $x'_r + \frac{l}{2} \in [x_l,x_r]$
end for

$L$ = MBRs inside $l$-band at the initial position
$n$ = #elements in $L$.
for $i$ = 1 to $n$ do
    $x_i = L[i]$
    delete MBR whose $x'_r$ is $x_i - \frac{l}{2}$ from $L$
    insert MBR whose $x'_l$ is $x_i + \frac{l}{2}$ into $L$
    if $|L| \geq N$ then
        SweepY($L$, $N$)
        for each dense segment $[y_j, y_{j+1})$ do
            $[x_i, x_{i+1}) \times [y_j, y_{j+1})$ is a dense region.
        end for
    end if
end for
```

2) Approximate PDR Method: The exact PDR method requires to run the plane sweep algorithm for all candidate cells, which can be a costly operation. The number of plane sweeps required depends on the number of candidate cells. If most of the cells cannot be accepted or rejected during the filtering step then a large number of plane sweeps is needed. This will significantly increase the query execution time. However in practice the querying application or the user can accept some loss of accuracy and identifying dense regions exactly is not necessary.

The work in [13] therefore presents an alternate method using Chebyshev polynomials of the first kind to approximate the density function $D(x, y)$ of the two dimensional space. l is assumed to be fixed. We adapt this method we adapt to our case. Our experimental results mirror that of [13], and show that the approximation method is very fast, so that approximations for different l can be computed on the fly.

a) Chebyshev Polynomials: The Chebyshev polynomial $T_k(x)$ of the first kind is a polynomial in x of degree k and defined by the relationship $T_k(\cos\theta) = \cos(k\theta)$. When $x \in [-1, 1]$, then $\theta \in [0, \pi]$. These polynomials obey the recurrence

$$T_k(x) = 2xT_{k-1}(x) - T_{k-2}(x),\ k \geq 1$$
$$T_0(x) = 1,\ T_1(x) = x.$$

The approximation $\hat{f}(x, y)$ of a function $f(x, y)$ with Chebyshev polynomials of degree k is given by:

$$\hat{f}(x, y) = \sum_{i=0,j=0}^{i+j\leq k} a_{i,j} T_i(x) T_j(y)$$

The Chebyshev coefficients $a_{i,j}$ are computed using the following formula:

$$a_{i,j} = \frac{c}{\pi^2} \int_{-1}^{1} \int_{-1}^{1} \frac{f(x, y) T_i(x) T_j(y)}{\sqrt{1-x^2}\sqrt{1-y^2}} dxdy$$

where,

$$c = \begin{cases} 4 \text{ when } i \neq 0, j \neq 0 \\ 2 \text{ when } i = 0, j \neq 0 \text{ or } i \neq 0, j = 0 \\ 1 \text{ when } i = 0, j = 0 \end{cases}$$

The x and y coordinates are normalized to between $+1$ and -1, with the bottom-left corner at $(-1, -1)$ and top-right corner at $(1, 1)$. In [13], Chebyshev coefficients are updated for the l-square neighborhood of each point as objects are added, so that that the density of the l-square neighborhood of a point is increased by $1/l^2$ for this point.

We will use a similar approach, considering the line segments obtained from step 2 one by one, and updating the coefficients. We first describe how to update coefficients for a l-square neighborhood of a point and then focus on updating coefficients for a line segment.

For each point p we need to update each coefficient $a_{i,j}$, for $i = 0, j = 0$ to $i + j \leq k$, so that the density of l-square neighborhood of p is increased by $1/l^2$. It has been shown in [13] that for each point if we can calculate the increment $a^{\delta}_{i,j}$ of coefficient $a_{i,j}$, then the updated coefficient $a'_{i,j}$ is:

$$a'_{i,j} = a_{i,j} + a^{\delta}_{i,j}.$$

$a^{\delta}_{i,j}$ is calculated as follows.

$$\begin{aligned} a^{\delta}_{i,j} &= \frac{c}{\pi^2} \int_{x_l}^{x_r} \int_{y_b}^{y_t} \frac{\frac{1}{l^2} T_i(x) T_j(y)}{\sqrt{1-x^2}\sqrt{1-y^2}} dxdy \\ &= \frac{c}{\pi^2 l^2} \int_{x_l}^{x_r} \frac{T_i(x)}{\sqrt{1-x^2}} dx \int_{y_b}^{y_t} \frac{T_j(y)}{\sqrt{1-y^2}} dy \\ &= \frac{c}{\pi^2 l^2} \int_{x_l}^{x_r} \frac{\cos(i \arccos(x))}{\sqrt{1-x^2}} dx \int_{y_b}^{y_t} \frac{\cos(j \arccos(y))}{\sqrt{1-y^2}} dy \\ &\text{[using the trigonometric representation of } T_i(x)] \qquad (1) \end{aligned}$$

y_b, y_t, x_l, x_r, are bottom, top, left, and right boundaries respectively of the l-square.

However, computing the above integrals for the l-rectangle region of a line segment is complicated, since the boundaries of a line segment's l-rectangle are not fixed values. This causes the y-limits to become functions of x. We will simplify the integrals by assuming that two boundaries of the l-rectangle neighborhood are parallel to y axis, as in Figure 3(b). Now x-limits are fixed values and y-limits are linear functions $f_1(x, x_l, x_r)$ and $f_2(x, x_l, x_r)$. The integrals now assume the

Location	Time of collection	Number of trajectories	Number of spatial points	Sampling frequency	Description
Beijing, China	Apr 2007 to Aug 2009	165	24778552	2-5 sec	**GeoLife Data:**
San Francisco, USA	2008-05-17 to 2008-06-10	536	11219955	10 sec	**TaxiCab Data:**
Starkey, Oregon, USA	April to August of 1993-1996	253	>287,000	1hr	**DeerElk Data:**

TABLE I
DESCRIPTION OF REAL DATA SET.

form

$$\begin{aligned} a_{i,j}^{\delta} &= \frac{c}{\pi^2 l^2}\int_{x_r}^{x_l}\frac{\cos(i\arccos(x))}{\sqrt{1-x^2}}dx \\ &\quad\times\int_{y_b=f_1(x,x_l,x_r)}^{y_t=f_2(x,x_l,x_r)}\frac{\cos(j\arccos(y))}{\sqrt{1-y^2}}dy \qquad (2)\\ &= \frac{c}{\pi^2 l^2}\int_{x_r}^{x_l}\frac{\cos(i\arccos(x))}{\sqrt{1-x^2}} \\ &\quad\times\frac{\sin(jf_1(x,x_l,x_r))-\sin(jf_2(x,x_l,x_r))}{j}dx \\ &= \frac{c}{\pi^2 l^2 j}[I_1 - I_2] \end{aligned}$$

where

$$I_k = \int \frac{\cos(i\arccos(x))\sin(jf_k(x,x_l,x_r))}{\sqrt{1-x^2}}, \; k = 1,2 \qquad (3)$$

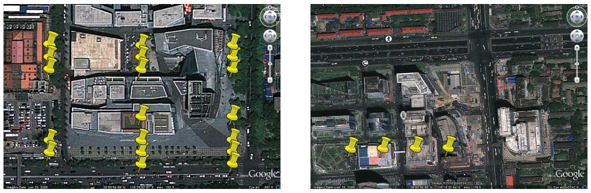
Unfortunately, using linear $f_1()$ and $f_2()$ in (2) results in an elliptical integral, which has no closed form. Numerical integration is expensive, and will not allow us to achieve our goal of quickly approximating the density function.

Since updating the Chebyshev coefficients for the l-rectangle neighborhood of a line segment will not be efficient, we proceed using a simpler region around line segments. Our final goal is to approximate the dense regions quickly. We take the middle point p_m of each segment, and update the coefficient for the l-square neighborhood of p_m. This makes the approximation method simple and quick, although it might harm the goodness of the approximation. Usually trajectory segments are much smaller than grid cells, and fully contained within a cell. If a line segment is bigger than grid cell width, l_c, then we segment the line into multiple lines of length l_c, except the last segment, which might be smaller.

VI. EXPERIMENTAL EVALUATION

In our experiments we used three real and one synthetic datasets. All the experiments were run in an Intel Xeon 3.0GHz processor running Linux 2.6.18 with 8GB of main memory. We used the disk manager and R*-tree implementation of the spatial index library [21] with page size 16KB.

Table I provides the description of the real datasets. The GeoLife dataset [22] contains public activity data (i.e. shopping, dining, sightseeing, hiking, cycling etc.) in Beijing, China. The TaxiCab dataset was collected from GPS equipped taxi cabs in San Francisco, USA [23]. The DeerElk data contains the trajectories of deer, elk and cattle in the Starkey Experimental Forest and Range in Oregon, USA [24]. The synthetic dataset contains two hundred thousand trajectories, each of length 250 recordings, generated for the Chicago metropolitan area road network.



(a) (b)

Fig. 7. ROIs identified Beijing with long stay duration in weekends.

We first consider identifying ROIs in the real datasets. For experiments we used the exact PDR method. Table II shows the temporal conditions used for the experiments on the GeoLife data. The results of our algorithms were then validated using Google Maps. Note that GeoLife data comes from Microsoft Asia employees, visitors, etc. Using a short stay duration (15 to 30 min) we found bus stops, railway and subway stations, the Tsinghua University canteen, etc. We then considered weekends and a longer stay duration (1.5 to 4 hr). This resulted in ROIs in (1) the Sanlitun area which houses many malls, bars and is a very popular place, (2) the Wenhua square which contains churches, theaters, and other entertainment places, and (3) Zhongguancun, referred to as 'China's Silicon Valley', having a lot of IT and electronics markets. Figure 7(a) and (b) shows Sanlitun and Zhongguancun area respectively in Beijing. When considered lunch and dinner time we found places that contain many restaurants. Interestingly ROIs found at lunch time contain regions near the Microsoft China head quarters which are absent in dinner time ROIs. Finally we identified ROIs on each individual day from April 2007 to August 2009. These resulted in (1) the Olympic media village, the Olympic sports center stadium during the Olympics 2008, (2) Peking University when the 'Regional Windows Core Workshop 2009 - Microsoft Research' was taking place in the PKU campus, (3) areas near the Great Wall in a weekend, (4) the Beijing botanical gardens, (5) the Celebrity International Grand Hotel, Beijing, etc.

Figure 8(a) shows all the ROIs found using the TaxiCab dataset. We further zoom in to ROIs and these are shown in figure 8(b) The San Francisco international airport, (c) a car rental, (d)the main downtown, union square (e) hotels: Star Wood, Westin, Mariott, (f) hotel Radisson (g) Ramada Plaza hotel (h) Embarcadero, Regency hotel, (i) San Francisco Caltrain station (j) the yellow cab access road. These were found for short stay duration of 10 minutes. When the stay

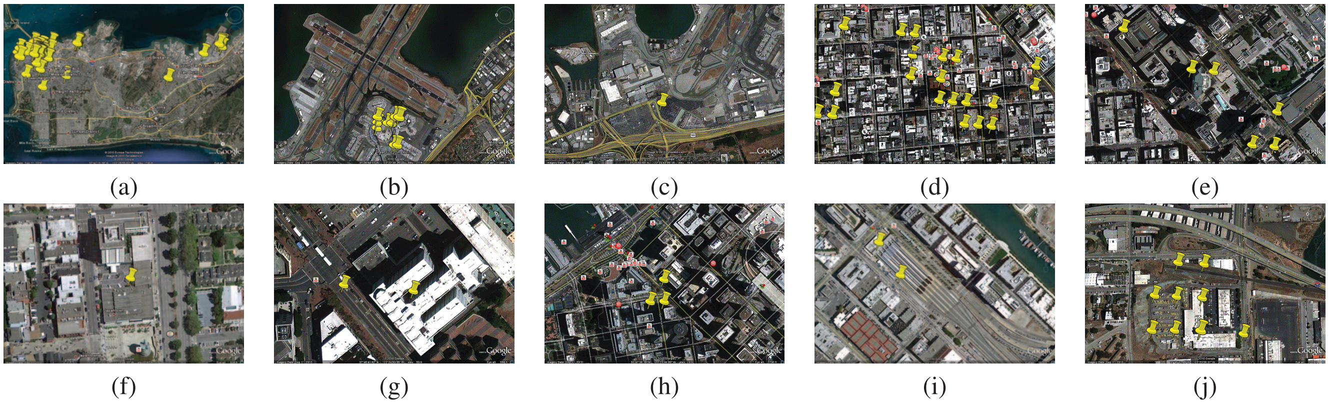


Fig. 8. ROIs identified for the TaxiCab data.

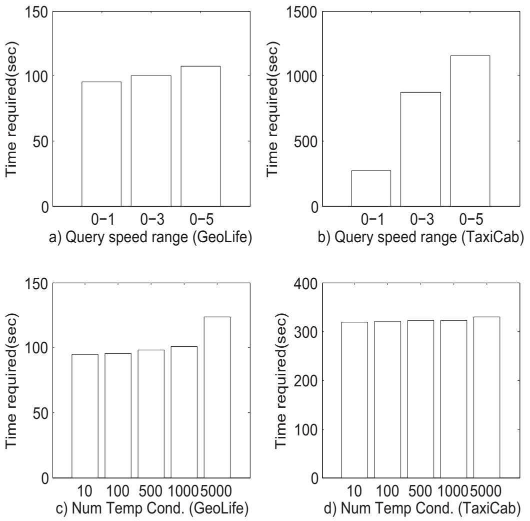


Fig. 9. Query parameters vs Time.

duration was increased to 12 hours we found only yellow cab access road, while for $2 - 3$ hours of stay duration we also found the airport.

	Time Period	Duration
1	Any day	15-30 min
2	Weekends	1.5-4 hr
3	Lunch time	0.5-1.5 hr
4	Dinner time	0.5-1.5 hr
5	Any day	1-4 hr

TABLE II
TEMPORAL ATTRIBUTES FOR THE GEOLIFE DATA EXPERIMENTS.

Finally, when using the DeerElk dataset, the ROIs found tend to be near a valley, with the largest ROI being close to a big water body (0.56 miles in length).

Figure 9 shows the query evaluation time for different values of parameters on real data. We do not consider the DeerElk data because there is not much variation of speed in this dataset. As we increase the speed range the query evaluation time increases very slowly for the GeoLife data. However, for the TaxiCab data the evaluation time increases sharply (the reason behind which is explained in the next paragraph). On the other hand, the effect of the number of the temporal conditions on query evaluation time is very small. This is because evaluating the temporal condition requires a sort merge join which is very fast. While varying the number of temporal conditions the amount of selected data from the index was kept the same, which ensures that the subsequent parts of the algorithm after verifying the temporal condition processed the same data.

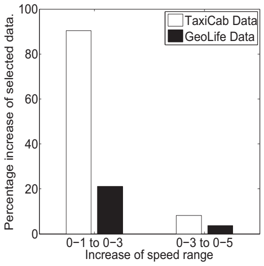
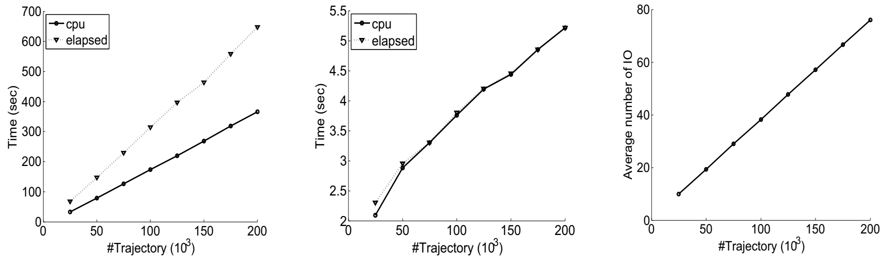


Fig. 10. Increase in fraction of selected data vs speed range.

Figure 10 shows the percentage increase in the selected data from the speed index as the query speed range increases from $0 - 1$mph to $0 - 3$mph and from $0 - 3$mph to $0 - 5$mph. Increasing the query speed range results in much higher increase of selected data in case of the TaxiCab data than that in the GeoLife data. This explains why the performance of query evaluation in the TaxiCab data is more affected by the query speed range than that in the GeoLife Data.

We also ran experiments on the synthetic dataset to determine the algorithm's behavior over large datasets. Figure 11(a) shows the time required to build the speed index. Here we report index construction times for indexing subtrajectories with speeds between 0 and 100, (although we need only low speed segments, i.e. 0 to 5, to answer typical ROI queries). Our index ranges are $[0, 1), [1, 2), \ldots, [99, 100)$. The cost of



(a) (b) (c)

Fig. 11. (a) Index building cost vs Database size. (b) Histogram computation time vs Database size. (c) Histogram computation IO vs Database size.

building the index is linear with the data size. The difference between CPU time and elapsed time is slowly increasing which is due to the increasing number of disk IO as the data size increases.

Figure 11(b) shows the time required for the histogram computation for each index range as a preprocessing step. Computing histogram matrices requires accessing the speed index and building an R*-tree with the selected data. Note that the elapsed time is very close to CPU time. This is because the histogram matrices are typically small and can fit into main memory. Moreover, the data indexed by the R*-tree is also small. Figure 11(c) shows the average number of IOs required for computing each histogram matrix.

For comparison purposes we also implemented the CB-SMOT approach [17]. CBSMOT estimates the eps parameter using a quantile function where the user has to specify the fraction of trajectory points that is expected to be in an ROI. Since in our approach we assume that the user will specify speed and distance, we can equivalently assume that the eps-distance is specified by the user. To compare CBSMOT with our method we ran it for a certain value of eps and minimum stay duration τ. Then we ran our method with speed range $[0, \frac{eps}{\tau})$ and minimum stay duration τ. Note that, CBSMOT finds ROIs for each individual trajectory but our method considers all trajectories of the dataset and identifies regions that are commonly interesting. Thus, CBSMOT identifies more ROIs than those identified by our method. To make the results of CBSMOT comparable with ours we extend it (E-CBSMOT) by applying the pointwise density method on its output. Note that, E-CBSMOT, similarly to CBSMOT, cannot evaluate temporal conditions as our approach.

Figure 12 shows the regions of interest found in the synthetic data using the exact, approximate and E-CBSMOT methods. ROIs identified by the Chebyshev approximation have more area than those by the exact method. This is due to the fact that the density coefficients are being over estimated because of considering the l-square neighborhood around the center of a line segment. The subtle difference between the result of E-CBSMOT and our method is because of the different definitions of ROIs.

Figure 13 shows the query evaluation time. As expected, the time for the approximation method is quite less than that of the exact method. Recall that the approximate method computes Chebyshev coefficients and finds dense regions for every different value of l. From the experimental results we argue that the whole approximation process is so fast that it is feasible to run the approximation for every different value of l. For the exact PDR method we experimented with two cases, when we (1) only add precomputed histogram matrices and (2) need to compute the histogram values after retrieving required subtrajectories from the speed index. In the later case we need to (i) access the speed index to retrieve the required trajectory segments (ii) build the R-Tree on them and (iii) compute the histogram values. This experiment shows the benefit of precomputing the histogram matrices and thus avoid accessing the speed index. Finally, E-CBSMOT takes much longer than our method since it has to scan the whole dataset.

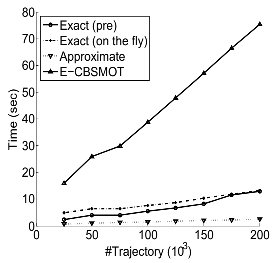


Fig. 13. Query time vs Database size.

VII. Conclusions

In this paper we address the problem of discovering regions of interest from trajectory databases. We give formal definition of ROIs in a more generic way than previous approaches and propose a framework to discover ROIs efficiently. We allow users to specify any arbitrary values for attributes defining ROI. Unlike previous approaches we do not scan the whole database to identify ROIs for a certain set of attribute values defining ROIs, neither assume any spatial information given about these regions.

We also consider a minimum number of objects must stay in an ROI for a minimum duration, which is absent in previous methods. We extend the Pointwise Density method to identify these regions with a minimum density of trajectories. Experimental results show that our proposed methods discover

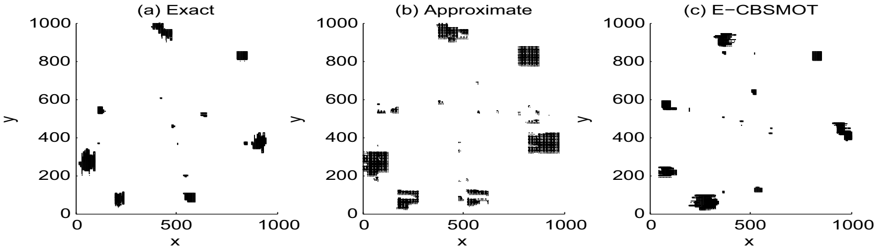


Fig. 12. ROIs identified by different methods.

ROIs efficiently and correctly. As a future work we want to address the issue of dealing with data with uncertainty e.g., noisy, low resolution data.

VIII. Acknowledgement

This work was supported in part by contract number N00014-07-C-0311 with the Office of Naval Research and NSF grant: NSF IIS 0803410.

References

[1] www.accutracking.com.
[2] tracNet24, www.isecuretrac.com.
[3] Footpath, www.pathintelligence.com.
[4] GeoChat, instedd.org/geochat.
[5] www.bikely.com.
[6] www.gpsxchange.com.
[7] www.everytrail.com.
[8] www.sports-tracker.com.
[9] K. Xie, K. Deng, and X. Zhou, "From trajectories to activities: A spatio-temporal join approach," in *LBSN'09: Proc. of the Int'l Workshop on Location Based Social Networks*, 2009, pp. 25–32.
[10] Q. Li, Y. Zheng, X. Xie, Y. Chen, W. Liu, and W.-Y. Ma, "Mining user similarity based on location history," in *ACM GIS*, 2008, pp. 1–10.
[11] Y. Zheng, L. Zhang, X. Xie, and W.-Y. Ma, "Mining interesting locations and travel sequences from gps trajectories," in *WWW*, 2009, pp. 791–800.
[12] X. Cao, G. Cong, and C. S. Jensen, "Mining significant semantic locations from gps trajectory," in *VLDB*, 2010, pp. 1009–1020.
[13] J. Ni and C. V. Ravishankar, "Pointwise-dense region queries in spatio-temporal databases," in *IEEE ICDE*, 2007, pp. 1066–1075.
[14] M. Hadjieleftheriou, G. Kollios, D. Gunopulos, and V. J. Tsotras, "On-line discovery of dense areas in spatio-temporal databases," in *SSTD*, 2003.
[15] C. S. Jensen, D. Lin, B. C. Ooi, and R. Zhang, "Effective density queries on continuously moving objects," in *ICDE*, 2006.
[16] L. O. Alvares, V. Bogorny, B. Kuijpers, J. A. F. de Macedo, B. Moelans, and A. Vaisman, "A model for enriching trajectories with semantic geographical information," in *ACM GIS*, 2007, pp. 1–8.
[17] A. T. Palma, V. Bogorny, B. Kuijpers, and L. O. Alvares, "A clustering-based approach for discovering interesting places in trajectories," in *ACM SAC*, 2008, pp. 863–868.
[18] F. Giannotti, M. Nanni, F. Pinelli, and D. Pedreschi, "Trajectory pattern mining," in *ACM KDD*, 2007, pp. 330–339.
[19] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu, "A density-based algorithm for discovering clusters in large spatial databases with noise," in *ACM SIGKDD*, 1996, pp. 226–231.
[20] N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger, "The R*-tree: An efficient and robust access method for points and rectangles," in *ACM SIGMOD*, 1990, pp. 322–331.
[21] "Spatial Index Library," http://dblab.cs.ucr.edu/spatialindexlib.html.
[22] http://research.microsoft.com/en-us/projects/geolife/.
[23] crawdad.cs.dartmouth.edu.
[24] http://www.fs.fed.us/pnw/starkey/.