Pointwise-Dense Region Queries in Spatio-temporal Databases

Jinfeng Ni and Chinya V. Ravishankar
Department of Computer Science and Engineering
University of California, Riverside
Riverside, CA 92521, USA

Abstract

Applications such as traffic management and resource scheduling for location-based services commonly need to identify regions with high concentrations of moving objects. Such queries are called dense region queries in spatio-temporal databases, and desire regions in which the density of moving objects exceeds a given threshold. Current methods for addressing this important class of queries suffer from several drawbacks. For example, they may fail to find all dense regions, provide ambiguous answers, impose restrictions on size, or lack a notion of local density.

We address these issues in this paper, starting with a new definition of dense regions. We show that we are able to answer dense region queries completely and uniquely using this definition. Dense regions in our approach may have arbitrary shape and size, as well as local density guarantees. We present two methods, the first, an exact method, and the second, an approximate method. We demonstrate through extensive experiments that our exact method is efficient and is superior to current approaches. Our approximate method runs orders of magnitude faster than our exact method, at the cost of a tolerable loss of accuracy.

1 Introduction

Continuing developments in computing, communications and positioning technologies have given rise to new applications, including vehicle fleet tracking, watercraft and aircraft navigation, and emergency E911 service for cellular phones. Such applications have triggered new research towards supporting location-based services in mobile environments. Current work in this area focuses mainly on the modeling and indexing of moving objects, aiming to optimize spatio-temporal range and aggregation queries, k-nearest neighbors queries, or selectivity estimation.

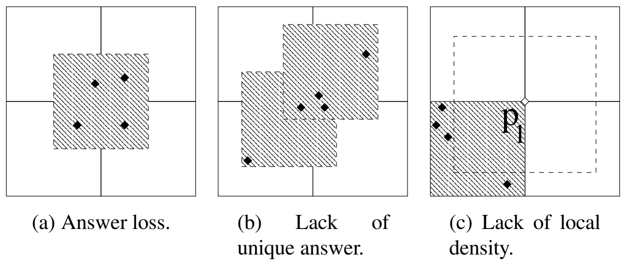
In this paper, we study dense region queries, which are important, but have received attention only recently [4, 7]. A region is *dense* if it has a high concentration of moving objects. Identifying dense regions is important for numerous applications, including traffic control, resource scheduling, and collision probability evaluation [4]. For example, traffic congestion in large cities may be alleviated if traffic databases were enhanced with the ability to *predict* dense regions that might develop in the near future, so commuters could choose their routes to avoid and avoid jams.

Hadjieleftherious et al. [4] introduced the problem of dense-region queries, and presented several techniques to evaluate such queries. Recently, Jensen et al. [7] have defined *effective density queries*. In both cases, a dense region is defined using the notion of *region density*. The density of a region in a 2-dimensional space containing moving objects is defined as the ratio of the number of objects in this region to its area. A dense-region query desires the regions with density no lower than some user-specified threshold.

1.1 Problems with previous work

Unfortunately, these proposals suffer from one or more of the following drawbacks, rendering them inappropriate for many real world applications. The first of these is *answer loss*. The authors argue in [4] that it is very difficult to evaluate general dense-region queries, since they require the computation of a *count* aggregate over every subregion in the space, whose number is unfortunately infinite. Instead, they have addressed the much simpler problem of *dense cell queries*. They partition the space into disjoint cells and look for cells with density beyond the threshold. This simplification, however, leads to the problem of *answer loss* [7], since it only reports dense cells, completely ignoring dense regions which span several cells. For example, in Figure 1(a), let each grid cell be a unit square and the density threshold be 4. Clearly, none of the grid cells is dense, and the work in [4] will not report dense regions. However, the dashed square contains four objects, and is hence a dense region, but is completely missed.

The second problem is ambiguity in query results. Jensen et al. [7] propose the notion of *effective density query* (EDQ), to address the problem of *answer loss* in [4]. Instead



(a) Answer loss. (b) Lack of unique answer. (c) Lack of local density.

Fig. 1. Problems with current methods.

of looking for only dense grid cells, [7] also reports dense regions which span several cells. For instance, the dashed square in 1(a) would be included in the query results.

Nevertheless, as acknowledged in [7], there might be multiple overlapping dense regions. Jensen et al. [7] suggest reporting only a set of non-overlapping dense regions to an EDQ. This strategy, however, could lead to *completely different* query results under different reporting strategies, even keeping the dataset and query parameters the same. Consider Figure 1(b), where each of the dashed squares contains 4 objects and is dense regions according to the criterion. Under the the definition of EDQ, either one is a valid answer; the other one is excluded from the query result. This could be confusing to users. Further, even when both dashed squares are equally important and useful, as argued by Jensen et al. [7], it might be better to report all dense regions, ignoring overlap, and to let users decide which are appropriate, according to preference or need.

The third problem is that these methods restrict dense regions to have either a fixed size or a fixed shape. Thus, [4] was only able to identify dense grid cells with fixed size, while [7] identified dense regions as squares of fixed size. This restriction could diminish the value of these two approaches in real world applications, where dense region may have arbitrary shape and size.

The final problem with current approaches is the lack of local density guarantees for points in the dense region. The *local density* for a point, defined as the ratio between the number of objects in the point's neighborhood and the area of the neighborhood, specifies how dense the objects are around the point. In contrast, *region density*, as used in earlier proposals is an aggregated and gross measurement, and does not reflect the details of the density distribution. Since the distributions of objects could be highly skewed, a dense region may contain points with local density far lower than the query threshold. For example, in Figure 1(c), the shadowed unit square contains 4 objects and is dense. However, if we consider point p at the right corner of this square, we find a 1.5-unit-square neighborhood centered on p entirely devoid of objects.

2 Our Approach: Pointwise-Dense Regions

We argue that these difficulties arise since [4, 7] do not use an appropriate definition of dense regions. We propose the use of *Pointwise-Dense Regions (PDRs)*, a new definition of density, that helps avoid *all* the aforementioned problems. Under this definition, we are able to answer dense-region queries uniquely, reporting all dense regions, regardless of shape and size. Further, PDR queries represent a more general class of dense region queries than the *dense-cell queries* of [4], and the *effective dense region queries* in [7], since under reasonable conditions, the answer to a PDR query includes the answer to those queries.

We will define *point density* as a unique attribute associated with each point p in the region, representing the density of moving objects around p. In contrast, under the definition of *region density*, point p could belong to an infinite number of regions, so p could itself not be associated with a unique density, making it hard to decide whether p should belong to a dense region or not. It is, in fact, the lack of a unique notion of density for a point p that causes the difficulties in the proposals of [4, 7].

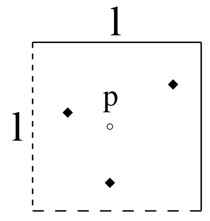


Fig. 2. Point density.

Definition 1. *The* l**-square neighborhood** S_p^l *of a point* p *is the square centered at* p *with edge length* l*, including the right and top edges, but excluding the left and bottom edges.*

l could be an arbitrary user-specified parameter. We assume $l \geq l_{min}$, where l_{min} is some predetermined value.

Definition 2. *The* **point density** $d_t(p)$ *at time* t *of point* p *is the value* $n_t(S_p^l)/l^2$*, where* $n_t(S_p^l)$ *is the number of moving objects located within* S_p^l *at time* t*.*

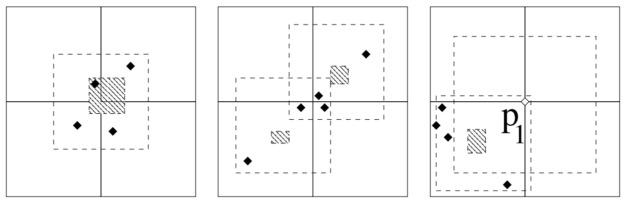
In Figure 2, point p's l-square neighborhood contains 3 moving objects at timestamp t. Therefore, $d_t(p) = \frac{3}{l^2}$.

It is now natural to define dense regions as sets of points whose *point densities* are all no lower than the query threshold. The problem of finding dense regions is now reduced to the problem of finding all dense points.

Definition 3. *Given* $\rho \geq 0$*, point* p *is* ρ**-dense** *with respect to an* l*-square neighborhood at time* t*, if* $d_t(p) \geq \rho$*. A region is* ρ*-dense with respect to* l*-square neighborhoods at time* t *if every point inside the region is* ρ*-dense at time* t*.*

We will consider the following query types.

Definition 4. *Given a set of moving objects, an edge length* l*, a density threshold* ρ*, and a timestamp* q_t*, the* **snapshot PDR query** (ρ, l, q_t) *requests all regions that are* ρ*-dense with respect to* l*-square neighborhoods at time* q_t*.*



(a) No loss. (b) Uniqueness. (c) Local density.

Fig. 3. Solution using PDRs

Definition 5. *Given a set of moving objects, the* **interval PDR query** $(\rho, l, [q_{t1}, q_{t2}])$ *requests the union of the responses to the snapshot PDR queries* (ρ, l, q_t) *across all timestamps* $q_t \in [q_{t1}, q_{t2}]$.

2.1 Answering PDR Queries

We describe two methods to efficiently evaluate PDR queries. Our first method uses a filtering-refinement strategy, and finds exact answers to PDR queries. The filtering step uses a histogram, and the refinement step finds dense regions using plane-sweeping.

Our second method uses a completely different approach. It approximates the density distribution using a polynomial function, and uses the approximation to identify dense regions. While this method provides approximate answers, it runs much faster than our first method. We demonstrate the accuracy and efficiency of both methods through extensive experiments.

To see the superiority of the PDR approach, consider an l-square neighborhood of the same size as the grid cell in [4], and the square used in [7]. In Figure 3(a), Figure 3(b) and Figure 3(c), the filled rectangles are dense under PDR. We note the following under PDR.

No answer loss: Unlike [4], which only reports dense grid cells, dense regions under PDR include the centers of all squares containing at least ρl^2 objects. Therefore, it does not miss any dense region. Figure 3(a) shows the dense regions under PDR.

No ambiguity: Unlike [7] which chooses *one* of the dashed squares in Figure 1(b), PDR reports *all* the points whose l-square neighborhoods have at least ρl^2 objects. Hence, both dashed squares are returned in the query response, as shown in Figure 3(b).

Regions with arbitrary shape or size: Figures 3(a), 3(b) and 3(c) clearly demonstrate that regions dense under PDR may have arbitrary rectangular shape and size. We will give algorithms to evaluate PDR queries for arbitrary l.

Local density guarantees: Under PDR, all points in a dense region must have at least ρl^2 objects in its l-square neighborhood. Hence, point p in Figure 1(c) is excluded in the query result, as shown in Figure 3(c).

2.1.1 Generality of PDR

We note that PDR query results are actually a superset of the results under the methods of [4, 7]. The centers of the dense regions of [4, 7] are ρ-dense under PDR, and are hence included in the PDR query results, as shown in Figures 1 and 3. From the PDR perspective, [4, 7] results are equivalent to a subset of discretized ρ-dense points (the center of the dense region in their definition). PDR queries are required to report all ρ-dense points, and are therefore a more general and challenging form of dense region query. To the best of our knowledge, this is the first work which address how to efficiently find a complete and unique answer to the dense region query.

3 Related Work

Previous work has been in a number of related areas. The work in [3] models moving objects. The TPR-tree [16], TPR*-tree [18], STRIPE [14], and the B^x-tree [6] focus on indexing of predicted trajectories of moving objects. Work on indexing of historical trajectory includes TB-tree [15], MVR-tree [5], SEB-tree [17], SETI [2] and PA-tree [11]. Index structures are proposed in such work to support range queries, nearest neighbor or reverse nearest neighbor queries.

Spatio-temporal aggregations [12, 13] and selectivity estimation range queries [19] are related to dense region queries, since both must compute an aggregated value within some range. However, aggregation and selectivity estimation queries require a user-specified spatial range. In contrast, *no* spatial query predicates are specified for a dense region query, and the dense region query must *identify* the ranges that satisfy the density threshold. One naive approach to the dense region query is to use the selectivity estimator for all possible ranges. However, this approach is unrealistic, since the number of regions over which we have to execute an aggregation query is prohibitively high.

The problem of clustering moving objects is also related to dense region queries. Clustering algorithms [20, 9, 8] typically operate based on some optimization criteria, so each cluster represents a group of objects located within some ranges. However, these techniques are not directly useful to us, since clusters do not guarantee densities over the desired threshold.

The work most closely related is the on-line discovery of dense regions by Hadjieleftheriou et. al [4] and the *Effective Density Query* by Jensen et. al [7]. However, as discussed in Section 1, these two proposals suffer from the problems of answer loss, query answer ambiguity, size or shape restrictions on dense regions, and the lack of local density guarantees.

4 Problem Statement and Assumptions

We assume a set of n objects moving in a $L \times L$ region in the XY-plane. Each object is modeled as a point, and reports its current location (x, y) and velocity (v_x, v_y) to a central server. We adopt a linear motion model for simplicity and for consistency with previous work, including [4, 7]. The position (x_t, y_t) of a moving object at timestamp $t \geq t_{now}$ is hence $x_t = x + (t - t_{now})v_x$, and $y_t = y + (t - t_{now})v_y$. Several indexing methods have been proposed for linear movement, which we can adopt in our framework.

As suggested by Jensen et al. [7], finding dense regions for *a period of time* appears to be less useful than finding dense regions at some timestamp. Therefore, as in [7], in this paper, we focus on snapshot version of PDR queries.

We assume that a moving object reports its new location to the server within a time period of length U, called the *maximum update time* [7]. Further, as in [4, 7], a PDR query is a predictive query which asks for the dense regions no more than W time instants into the future. W is called the length of prediction window. Finally, we define the *time horizon* $H = U + W$ as the maximum duration of time between which a PDR query will be targeted at [7].

We assume all object data are maintained and updated in an index structure. Without loss of generality, we use a TPR-tree [16] to index the moving objects. As in [7], we do not count the overhead of maintaining and updating the index when we evaluate the efficiency of the proposed methods, since this index may be maintained for other queries, such as spatio-temporal range or nearest-neighbor queries.

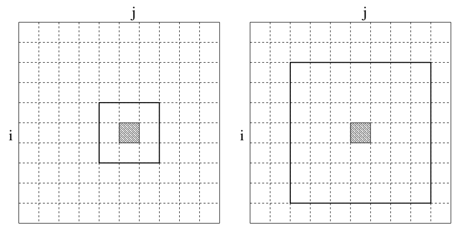
5 A Filtering-Refinement Approach

Our filtering-refinement approach to PDR queries maintains a data structure called a *density histogram (DH)* [7], for each timestamp $t \in [t_{now}, t_{now} + H]$. The region is divided into an $m \times m$ grid, and a density histogram at time t maintains a counter of the number of moving objects in each grid cell. We denote the cell at row i and column j by $c_{i,j}$.

We choose a grid size m such that the cell edge length $l_c = L/m \leq \frac{l_{min}}{2}$ for a suitable l_{min}. This strategy allows us to evaluate PDR queries with respect to any l-square with $l \geq l_{min}$. Query processing consists of two steps:

1. **Filtering step:** We use density histograms to identify cells that are guaranteed dense (*accepts*), the cells that are guaranteed not dense (*rejects*), and the *candidate* cells for the refinement step.

2. **Refinement step:** For each candidate cell, we execute a spatio-temporal range query over the TPR-tree, and propose an efficient plane-sweep algorithm to determine the dense regions within each candidate cell.



(a) Conservative. (b) Expansive.

Fig. 4. Neighborhoods. ($\eta_l = 2$, $\eta_h = 3$).

5.1 Maintaining Density Histograms

All counters in a density histogram are initialized to be 0. The histogram is updated as objects send location updates in the form of insertions or deletions at each timestamp t_{now}.

The insertion update $(t_{now}, x_1, y_1, v_1^x, v_1^y)$ inserts an object which starts moving from location (x_1, y_1) with velocity (v_1^x, v_1^y), starting t_{now}. The deletion update $(t_2, t_{now}, x_2, y_2, v_2^x, v_2^y)$ deletes a previous movement at t_{now}, which started moving from position (x_2, y_2) at timestamp t_2 with velocity (v_2^x, v_2^y).

Insertion and deletions are handled differently. For an object insertion $(t_{now}, x_1, y_1, v_1^x, v_1^y)$, we compute the new trajectory of the object during $[t_{now}, t_{now} + H]$ using the velocity (v_1^x, v_1^y). We increment by 1 the counter for each cell that intersects the object's trajectory. For an object deletion $(t_2, t_{now}, x_2, y_2, v_2^x, v_2^y)$, we compute the old trajectory of the object during $[t_{now}, t_2 + H]$ using the velocity (v_2^x, v_2^y). We decrement by 1 the counter for each cell that intersects with the object's old trajectory.

The storage overhead for the histograms is Hm^2.

5.2 Query Processing: Filtering Step

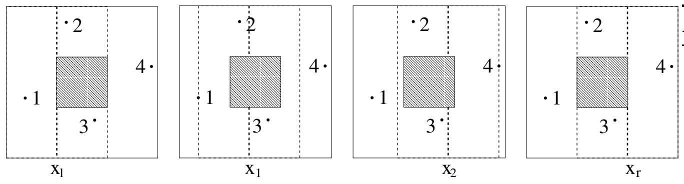
Let $l \times l$ be the size of the neighborhood and $l_c \times l_c$ be the size of grid cells. Let $\eta_l = \lfloor \frac{l}{2l_c} \rfloor$ and $\eta_h = \lceil \frac{l}{2l_c} \rceil$. Let $|R|$ denote the number of objects in region R.

Definition 6. *(See Figure 4(a).) The conservative neighborhood $C_{i,j}$ for grid cell $c_{i,j}$ is the union of grid cells $c_{u,v}$ for which $i - \eta_l < u < i + \eta_l$ and $j - \eta_l < v < j + \eta_l$.*

We note that any point $p \in c_{i,j}$ is *at most* distance $l/2$ away from the edges of $C_{i,j}$, so that p's l-square neighborhood *completely contains* $C_{i,j}$. Therefore, if $|C_{i,j}| \geq \rho l^2$, cell $c_{i,j}$ is surely dense, and is *accepted*.

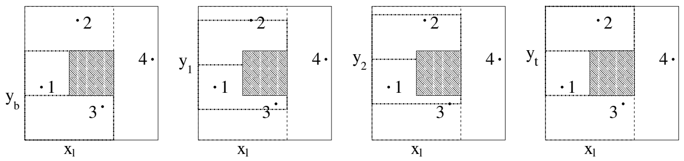
Definition 7. *(See Figure 4(b).) The expansive neighborhood $E_{i,j}$ for grid cell $c_{i,j}$ is the union of grid cells $c_{u,v}$ for which $i - \eta_h \leq u \leq i + \eta_h$ and $j - \eta_h \leq v \leq j + \eta_h$.*

We note again that any point $p \in c_{i,j}$ is *at least* distance $l/2$ away from the edges of $E_{i,j}$, so that p's l-square neighborhood is *completely contained* in $E_{i,j}$. Therefore, if $|E_{i,j}| < \rho l^2$, cell $c_{i,j}$ can not be dense, and is *rejected*.



(a) $\mathcal{L}_x = \{1, 2, 3\}$. (b) $\mathcal{L}_x = \{2, 3\}$. (c) $\mathcal{L}_x = \{2, 3, 4\}$. (d) $\mathcal{L}_x = \{2, 3, 4\}$.

Fig. 5. l-band sweep along X-dimension.



(a) $\mathcal{L}_y = \{1, 3\}$. (b) $\mathcal{L}_y = \{1, 2, 3\}$. (c) $\mathcal{L}_y = \{1, 2\}$. (d) $\mathcal{L}_y = \{1, 2\}$.

Fig. 6. l-square sweep along Y-dimension.

We can now use these criteria to identify accepts, rejects and refinement candidates during the filter step, using Algorithm 1.

Algorithm 1 FilterQuery(q_t, ρ, l)

```
Require: $l_c \leq l/2$
  for each grid cell $c_{i,j}$ do
    $N_c = \sum |c_{u,v}|$ at $q_t$, where $c_{u,v} \in C_{i,j}$.
    $N_e = \sum |c_{u,v}|$ at $q_t$, where $c_{u,v} \in E_{i,j}$.
    if $N_c \geq \rho l^2$ then
      Mark cell $c_{i,j}$ as dense.
    else if $N_e < \rho l^2$ then
      Mark cell $c_{i,j}$ as not dense.
    else
      Mark cell $c_{i,j}$ as a candidate.
    end if
  end for
```

5.3 Query Processing: Refinement Step

The filtering step outputs a set of candidate cells. To identify the dense region inside a candidate cell $c_{i,j}$, we must first count all objects which appear in the l-square neighborhood of some point $p \in c_{i,j}$.

Let candidate cell $c_{i,j}$ have left-bottom corner (x_l, y_b) and right-top corner (x_r, y_t). Let S be a square whose left-bottom corner is $(x_l - l/2, y_b - l/2)$, and right-top corner is $(x_l + l/2, y_b + l/2)$. Clearly, S contains all objects that appear in the l-square neighborhood of some point $p \in c_{i,j}$.

We now execute a spatio-temporal range query over the TPR-tree, using spatial range S and temporal range q_t, and retrieve all the objects located within S at timestamp q_t.

Algorithm 2 RefineQuery((ρ, q_t, l), cell $c_{i,j}$).

```
  $\Omega = \{x_l, x_r\}$
  for Each object at location $(x, y) \in S$ do
    Insert $x - l/2$ into $\Omega$, if $x - l/2 \in [x_l, x_r]$
    Insert $x + l/2$ into $\Omega$, if $x + l/2 \in [x_l, x_r]$
  end for
  Sort $\Omega$ into increasing order.
  $\mathcal{L}_x$ = {objects inside the $l$-band at $x_l$ }.
  while $\Omega \neq \emptyset$ do
    $x_i$ = ExtractMin($\Omega$). {Find & remove min from $\Omega$}.
    $x_{i+1}$ = Min($\Omega$). {Find min in $\Omega$}
    Delete object whose x-coordinate is $x_i - l/2$ from $\mathcal{L}_x$.
    Insert object whose x-coordinate is $x_i + l/2$ into $\mathcal{L}_x$.
    if $|\mathcal{L}_x| \geq \rho l^2$ then
      SweepY($\mathcal{L}_x$, $(\rho, q_t, l)$).
      for Each dense segment $[y_j, j_{j+1})$ returned from SweepY. do
        Output $[x_i, x_{i+1}) \times [y_j, j_{j+1})$ as dense region.
      end for
    end if
  end while
```

5.3.1 l-bands and Plane-sweeping

We will use a plane-sweeping algorithm to identify the dense regions within cell $c_{i,j}$. The algorithm sweeps an *l-band* with width l and height $l + y_t - y_b$ along the X-dimension (see Figure 5). The l-band is identified by its vertical center line, which sweeps right, from x_l to x_r. Let $\mathcal{L}_x$ denote the set of objects inside the l-band. $\mathcal{L}_x$ is initialized with the objects in the l-band when at x_l.

Let a *stopping event* be the moment when the l-band's left edge or right edge *touches* some object within S. Let $x_l < x_1 < x_2 < \cdots < x_r$ be the X-coordinates of the l-band's center line for the stopping events. Figure 5(a), 5(b), 5(c) and 5(d) show the l-band for four stopping events. When the left edge touches some objects, we remove them from $\mathcal{L}_x$. When the right edge touches some objects, we insert them into $\mathcal{L}_x$.

Plane sweeping along the Y-dimension proceeds exactly as for the X-dimension, with the X and Y axes interchanged. Figures 6(a), 6(b), 6(c) and 6(d) show the l-square for all the four stopping events along the Y-axis. As the bottom edge touches objects, we remove them from $\mathcal{L}_y$. As the top edge touches objects, we insert them into $\mathcal{L}_y$.

Algorithms 2 and 3 illustrate the plane-sweep method.

Lemma 1. *The density of point (x, y), where $x \in [x_i, x_{i+1})$, is equal to the density of point (x_i, y).*

Proof. When the center line of the l-band is between $[x_i, x_{i+1})$, no objects join or leave the l-band. Therefore, density for point $p(x, y)$, where $x \in [x_i, x_{i+1})$ is same as that of point (x_i, y). □

Algorithm 3 SweepY($\mathcal{L}_x$, (ρ, q_t, l)).

```
$\Omega = \{y_b, y_t\}$
for Each object with location $(x, y)$ in $\mathcal{L}_x$ do
    Insert $y - l/2$ into $\Omega$, if $y - l/2 \in [y_b, y_t]$
    Insert $y + l/2$ into $\Omega$, if $y + l/2 \in [y_b, y_t]$
end for
Sort $\Omega$ into increasing order.
$\mathcal{L}_y$ = {objects inside the $l$-square at $y_b$ }.
while $\Omega \neq \emptyset$ do
    $y_j$ = ExtractMin($\Omega$). {Find & remove min from $\Omega$}.
    $y_{j+1}$ = Min($\Omega$). {Find min in $\Omega$}
    Delete object whose y-coordinate is $y_j - l/2$ from $\mathcal{L}_y$.
    Insert object whose y-coordinate is $y_j + l/2$ into $\mathcal{L}_y$.
    if $|\mathcal{L}_y| \geq \rho l^2$ then
        Mark $[y_j, y_{j+1})$ as dense segment.
    end if
end while
```

Lemma 1 implies that instead of computing the density for all $x \in [x_l, x_r]$, we only need to compute the density at a finite number of X-coordinates. This significantly reduces the complexity of our problem.

Lemma 2. *The density of point (x, y), where $y \in [y_i, y_{i+1})$, is equal to the density of point (x, y_i).*

During the sweeping process along Y-dimension, we compute the size of $\mathcal{L}_y$ at each stopping event, and obtain the density of point (x_i, y_j), where y_j is one of $y_b, y_1, y_t, \cdots, y_r$. If the density of point $p(x_i, y_j)$ is at least ρl^2, we know from Lemma 2 that the vertical segment with X-coordinate x_i and y-coordinate between $[y_j, y_{j+1})$ is a dense segment. Using Lemma 1, we derive from this dense segment a dense rectangle $[x_i, x_{i+1}) \times [y_j, y_{j+1})$.

5.3.2 An Example

Assume $\rho l^2 = 3$. Consider Figure 5, where we sweep the l-band along X-dimension for candidate cell $c_{i,j}$. For the l-band at x_l, we sweep an l-square, as shown in Figure 6. We compute the density at each stopping event and find only at y_1, the cardinality of $\mathcal{L}_y$ is 3 and hence segment $[y_1, y_2)$ is the only dense segment. Therefore, we identify the rectangel $[x_l, x_1) \times [y_1, y_2)$ as one dense region.

6 An Approximation-based PDR Method

The algorithm presented in Section 5 returns exact answers to PDR queries, but incurs overhead proportional to the number of objects. As suggested in [4], we are often interested in finding dense regions very quickly, tolerating some loss of accuracy.

We now present a method which returns approximate answers to PDR queries, but runs much faster. This method models the point density distribution over the XY-plane as a function $d_t(x, y)$ of (x, y). We can evaluate a PDR query by first computing the density function $d_t(x, y)$ for $t \in [t_{now}, t_{now} + H]$. We can then identify the regions in R^2 where $d_t(x, y)$ is beyond the threshold ρ.

We approximate the density function $d_t(x, y)$ with a 2-dimensional Chebyshev polynomial $\hat{d}_t(x, y)$. Chebyshev polynomials are widely used, and have a rich theory. They also make it easy to compute the ρ-dense regions, which are the regions where $\hat{d}_t(x, y)$ is no less than the threshold. We can also compute contour lines for the approximated distribution in explicit form, which provide a clear overview of the distribution of moving objects. Finally, a Chebyshev polynomial approximation is very accurate, and incurs low storage overhead due to its nice *min-max* property [10], which has proven very useful for similarity search in time series databases [1], or spatio-temporal range queries [11].

Unlike the filtering-refinement algorithm which allows that the l-square neighborhood have varied size, the approximated method assumes that l is predetermined. This limitation may be justified by the fact that the approximated method runs much faster and is able to compute the contour line of distribution.

6.1 Chebyshev Approximations

We start with a brief review of Chebyshev polynomials.

Definition 8. *The k-degree Chebyshev polynomial $T_k(x)$ is $T_k(x) = cos(k\theta)$, where $\theta = cos^{-1}(x)$, for $x \in [-1, 1]$, and satisfies the recurrence*

$$\begin{aligned} T_0(x) &= 1,\ T_1(x) = x, \\ T_k(x) &= 2xT_{k-1}(x) - T_{k-2}(x),\ k > 1. \end{aligned}$$

We can use Chebyshev Polynomials as the basis functions to approximate a function $f(x, y),\ x \in [-1, 1], y \in [-1, 1]$, as a degree-$k$ polynomial $\hat{f}(x, y)$ as follows

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

where $a_{i,j}$ are the Chebyshev coefficients. The coefficients $a_{i,j}$ are computed using the following Theorem [10].

Theorem 1.

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

where

$$c = \begin{cases} 4, & \textit{if } i \neq 0,\ j \neq 0, \\ 2, & \textit{if } i = 0,\ j \neq 0, \textit{ or if } i \neq 0,\ j = 0 \\ 1, & \textit{if } i = 0,\ j = 0 \end{cases}$$

6.2 Approximating Density On-line

Without loss of generality, we normalize the XY-plane into a square with low-left corner $(-1,-1)$ and top-right corner $(1,1)$. We maintain the approximated density function $\hat{d}_t(x,y)$ for each timestamp $t \in [t_{now}, t_{now}+H]$. All coefficients for the density functions $\hat{d}_t(x,y)$ are initialized to zero. We update the coefficients upon object insertion or deletion, to reflect the density changes.

We denote a rectangle with low-left corner $(x^\perp, y^\perp)$, and top-right corner $(x^\top, y^\top)$ by $[x^\perp, x^\top] \times [y^\perp, y^\top]$.

6.2.1 Object Insertion

Given an object insertion (t_{now}, v_1, x_1, y_1), we first compute the predicted trajectory (x_t, y_t) at $t \in [t_{now}, t_{now}+H]$, where $x_t = x_1 + (t - t_{now})v_1^x$, and $y_t = y_1 + (t - t_{now})v_1^y$.

Let $d_t(x,y)$ and $d'_t(x,y)$ be the density functions *before* and *after* the object insertion, and let

$$\begin{aligned} \hat{d}_t(x,y) &= \sum_{i=0,j=0}^{0\le i+j\le k} a_{i,j}^{[t]} T_i(x)T_j(y), \\ \hat{d}'_t(x,y) &= \sum_{i=0,j=0}^{0\le i+j\le k} a_{i,j}^{\prime[t]} T_i(x)T_j(y). \end{aligned}$$

To simplify notation, we will omit t from the coefficient superscripts, and simply write $a_{i,j}$ and $a'_{i,j}$.

For each point (x_t, y_t) on an object's trajectory, let $S_l(x_t,y_t) = [x_t^\perp, x_t^\top] \times [y_t^\perp, y_t^\top]$ be the square with edge length l centered at (x_t, y_t). Clearly, this object belongs to the l-square neighborhood for any point in $S_l(x_t,y_t)$, and increases the density for points in $S_l(x_t,y_t)$ by $1/l^2$.

Let $d_t(x,y) = d'_t(x,y) + \delta_t(x,y)$, where

$$\delta_t(x,y) = \begin{cases} 1/l^2, & \text{if } (x,y) \in S_l(x_t,y_t), \\ 0. & \text{otherwise.} \end{cases} \tag{2}$$

Let the approximated polynomial for $\delta_t(x,y)$ be

$$\hat{\delta}_t(x,y) = \sum_{i=0,j=0}^{0\le i+j\le k} a_{i,j}^{[\delta]} T_i(x)T_j(y).$$

Lemma 3. *Let $f(x,y)$ and $g(x,y)$ have Chebyshev approximations $\hat{f}(x,y)$ and $\hat{g}(x,y)$ with coefficients $a_{i,j}^{[f]}$ and $a_{i,j}^{[g]}$, respectively. Let $h(x,y) = f(x,y) + g(x,y)$ have Chebyshev approximation $\hat{h}(x,y)$ with coefficients $a_{i,j}^{[h]}$. Then,*

$$a_{i,j}^{[h]} = a_{i,j}^{[f]} + a_{i,j}^{[g]}.$$

Proof. The result follows from Equation 1. □

We can use Lemma 3 to update coefficients upon object insertion, by computing the $a_{i,j}^{[\delta]}$ using the following result.

Algorithm 4 ObjectionInsert((t_{now}, v_1, x_1, y_1))

```
for $t \in [t_{now}, t_{now}+H]$ do
    Let the function $\hat{d}_t(x,y)$ have coefficients $a_{i,j}$.
    $x_t = x_1 + (t - t_{now})v_1^x$, $y_t = y_1 + (t - t_{now})v_1^y$.
    Compute coefficients $a_{i,j}^{[\delta]}$ using Equation 3.
    $a_{i,j} = a_{i,j} + a_{i,j}^{[\delta]}$
end for
```

Lemma 4.

$$a_{i,j}^{[\delta]} = \frac{1}{\pi^2 l^2} a_i^{[x]} a_j^{[y]} \tag{3}$$

where

$$a_i^{[x]} = \begin{cases} \arccos(x_t^\perp) - \arccos(x_t^\top) & \textit{if } i = 0, \\ \frac{2}{i}(\sin(i \arccos(x_t^\perp)) - \sin(i \arccos(x_t^\top))) & \textit{if } i > 0. \end{cases}$$

and

$$a_j^{[y]} = \begin{cases} \arccos(y_t^\perp) - \arccos(y_t^\top) & \textit{if } j = 0, \\ \frac{2}{j}(\sin(j \arccos(y_t^\perp)) - \sin(j \arccos(y_t^\top))) & \textit{if } j > 0. \end{cases}$$

Proof. We can compute $a_{i,j}^{[\delta]}$ by replacing $f(x,y)$ in Equation 1 with the value $\delta_t(x,y)$ from Equation 2. This yields

$$\begin{aligned} a_{i,j}^{[\delta]} &= \frac{c}{\pi^2} \int_{x_t^\perp}^{x_t^\top} \int_{y_t^\perp}^{y_t^\top} \frac{\frac{1}{l^2} T_i(x)T_j(y)}{\sqrt{1-x^2}\sqrt{1-y^2}} dx\, dy \\ &= \frac{c}{\pi^2 l^2} \int_{x_t^\perp}^{x_t^\top} \frac{T_i(x)}{\sqrt{1-x^2}} dx \int_{y_t^\perp}^{y_t^\top} \frac{T_j(y)}{\sqrt{1-y^2}} dy \end{aligned}$$

Equation 3 follows from the indefinite integral

$$\int \frac{T_i(x)}{\sqrt{1-x^2}} = \begin{cases} -\frac{\sin(i \arccos(x))}{i} & \text{if } i > 0, \\ -\arccos(x) & \text{if } i = 0 \end{cases}$$

□

Algorithm 4 shows how inserts are handled.

6.2.2 Object Deletion

The object deletion $(t_2, t_{now}, v_2, x_2, y_2)$ is handled similarly. We compute the trajectory (x_t, y_t) of the object at timestamp $t \in [t_{now}, t_2+H]$, where $x_t = x_2 + (t - t_2)v_2^x$ and $y_t = y_2 + (t - t_2)v_2^y$. Then we compute $a_{i,j}^{[\delta]}$ using Equation 3. Finally, we subtract $a_{i,j}^{[\delta]}$ from the original coefficients $a_{i,j}$. Algorithm 5 shows how deletes are handled.

6.3 Evaluating PDR Queries

A trivial approach would be to discretize the XY-plane into $m_d \times m_d$ grid, classifying a cell as dense if its center

Algorithm 5 ObjectionDelete($t_2, t_{now}, v_2, x_2, y_2$)

```
for $t \in [t_{now}, t_2 + H]$ do
    Let the function $\hat{d}_t(x, y)$ have coefficients $a_{i,j}$.
    $x_t = x_2 + (t - t_2)v_2^x$, $y_t = y_2 + (t - t_2)v_2^y$.
    Compute coefficients $a_{i,j}^{[\delta]}$ using Equation 3.
    $a_{i,j} = a_{i,j} - a_{i,j}^{[\delta]}$
end for
```

point is dense, using $\hat{d}_t(x, y)$. This method, however, incurs high computation overhead if we use a finer grid for improved accuracy.

Instead, we present the following method to efficiently identify dense regions, exploiting the ease of computing lower- and upper-bounds for Chebyshev polynomials. We first compute the lower-bound $d^\perp$ and upper-bound $d^\top$ of $\hat{d}_t(x, y)$ for the entire XY-plane, which we have normalized to $[-1 : 1] \times [-1 : 1]$. If $d^\perp \geq \rho$, the entire plane is dense. If $d^\top < \rho$, the plane is nowhere dense. Otherwise, we divide plane into 4 subregions, and we repeat the procedure for each subregion. We continue recursively until the subregion is smaller than $1/m_d$ on edge, at which stage we compute the density d at its center. If $d \geq \rho$, we take this subregion as dense. We obtain bounds as follows.

First, $T_i(x) = \cos(i \arccos(x))$ is a cosine function, so its lower and upper bound for $z \in [z_1, z_2]$ are

$$T_i^\perp[z] = \begin{cases} -1 & \text{if } \cos\left(\frac{k\pi}{i}\right) \in [z_1, z_2] \text{ for some odd } k, \\ \min\{T_i(z_1), T_i(z_2)\}, & \text{otherwise, and} \end{cases}$$

$$T_i^\top[z] = \begin{cases} 1 & \text{if } \cos\left(\frac{k\pi}{i}\right) \in [z_1, z_2] \text{ for some even } k, \\ \max\{T_i(z_1), T_i(z_2)\}, & \text{otherwise.} \end{cases}$$

We now bound $\hat{d}_t(x, y) = \sum_{i=0,j=0}^{i+j \leq k} a_{i,j} T_i(x) T_j(y)$, for $x \in [x^\perp : x^\top] \subseteq [-1 : 1]$ and $y \in [y^\perp : y^\top] \subseteq [-1 : 1]$. The lower bound for $(a_{i,j} T_i(x) T_j(y))$ is simply the lowest of $a_{i,j} T_i^\perp[x] T_j^\perp[y]$, $a_{i,j} T_i^\perp[x] T_j^\top[y]$, $a_{i,j} T_i^\top[x] T_j^\perp[y]$, and $a_{i,j} T_i^\top[x] T_j^\top[y]$. Finally, the lower bound of $\hat{d}_t(x, y)$ is the sum of the lower bounds for the terms $a_{i,j} T_i(x) T_j(y)$. An upper bound for $\hat{d}_t(x, y)$ is computed similarly.

6.4 Multiple Polynomials for Accuracy

Using a single global polynomial $\hat{d}_t(x, y)$ to approximate density over the entire XY-plane may not be accurate enough when the density distribution is very skewed. We can instead divide the space into $g \times g$ grid cells, using a different polynomial to approximate density in each cell.

It is straightforward to modify Algorithms 4 and 5 to accommodate multiple polynomials. If an object is at location (x_t, y_t) on its trajectory and $S_l(x_t, y_t)$ is the square with edge length l centered at (x_t, y_t), we know that only the points within $S_l(x_t, y_t)$ will have their density increased (or decreased) by $1/l^2$. First, we determine the cells that overlap with square $S_l(x_t, y_t)$. The polynomial coefficients for these overlapping cells must be updated. Next, we compute the overlap rectangle between these cells and $S_l(x_t, y_t)$. The lower and upper bounds of the rectangle in the X- and Y-dimensions are used as $x_t^\perp$, $x_t^\top$, $y_t^\perp$, $y_t^\top$, when we compute the Chebyshev $a_{i,j}^{[\delta]}$ using Equation 3.

Given $g \times g$ degree-k polynomials are used, the storage overhead for the coefficients is $Hg^2(k + 1)(k + 2)/2$.

Parameter	Value
Page size	4K
Buffer size	10% of dataset size
Random disk access time	10 ms
Maximum update interval(U)	20
Predication window length (W)	10
Edge length of l-square(l)	**40**, 60
Number of objects	10K, **100K**, 500K
Relative density threshold ϱ	1, 2, **3**, 4, 5
Num. of polynomials ($g \times g$)	**400**, 1600
Degree of polynomial (k)	3, 4, **5**
Num. of cells in Density Histogram ($m \times m$)	**10000**, 40000, 62500
Grid for polynomial evaluation ($m_d \times m_d$)	500×500

Table 1. Experimental setup

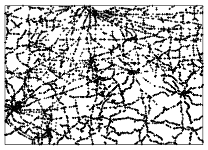
7 Experimental Evaluation

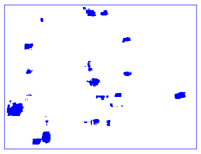
Our experiments were run on a 2.8Ghz Pentium IV CPU with 1Gb of main memory. We used the method of [4] to generate synthetic data sets, but also simulated real world applications more closely by using the Chicago metropolitan road network in our experiments.

The XY-plane was a square, 1,000 miles on edge. We generated three datasets CH10K, CH100K, and CH500K, with 10K, 100K and 500K moving objects, respectively. At least 1% of the objects issued updates at each timestamp. Object velocities were between 15 and 100 miles per hour, and were drawn from a skewed distribution. The maximum update interval U was set to 20 timestamps, and the prediction window length W to 10, giving a time horizon $H = 30$.

For each configuration, we ran a query workload and reported the average performance per query. Each query has three parameters: the density threshold ρ, the edge length l of the square neighborhood, and the query time q_t. We actually used a *relative density threshold* ϱ. Given N objects in the region of area 10^6 square miles, we choose $\rho = \frac{N\varrho}{10^6}$. The parameter ϱ is varied from 1 to 5, leading to ρ varying between 0.5 to 2.5 for dataset CH500k. Query time q_t is randomly distributed in $[t_{now}, t_{now} + H]$ when the query is issued at t_{now}. The edge length l is set to be 40 or 60.

We used a main memory buffer to store all density histograms, and the polynomial coefficients for $t \in [t_{now}, t_{now} + H]$. The buffer size depended on the time horizon H, and the granularity of density histograms for the filtering step, or the number and the degree of polynomials for the approximation method. However, this buffer


(a) Objects.


(b) Dense regions (FR).

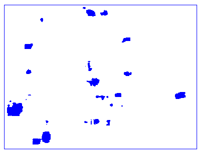
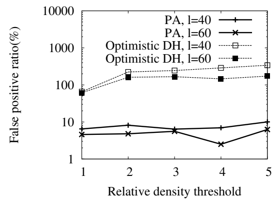
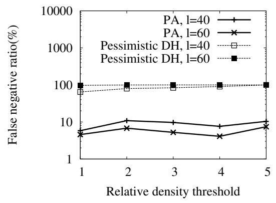

(c) Dense regions (PA).

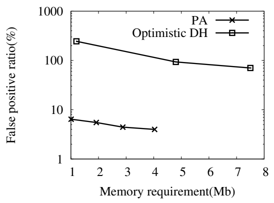
Fig. 7. An example (CH10k).



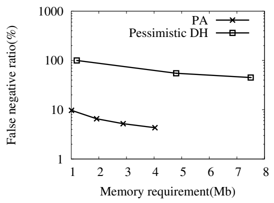
(a) r_{fp} vs. l and ϱ.



(b) r_{fn} vs. l and ϱ.



(c) r_{fp} ($l = 40$, $\varrho = 3$).



(d) r_{fn} ($l = 40$, $\varrho = 3$).

Fig. 8. Accuracy (dataset CH100K).

size is independent of dataset size. The number of counters in a density histogram was set to 10000, 40000 and 62500, with 10000 being the default. The number of polynomials was set to 400 and 1600, with 400 being the default value. The polynomial degree was set to 3,4 or 5, with 5 being the default value. In the default setting, the density histograms require a buffer of size 1.2Mb, while the polynomial approximation method requires 1.0Mb. This memory overhead remains the same as we increase the dataset size, so our method scales well. The TPR-tree index for the refinement step in filtering-refinement processing was assigned a buffer 10% of dataset size. Table 1 shows the experimental setup, with values in bold denoting the default values.

We compared three methods by accuracy and overhead. FR is the filter-refinement method (Section 5) to find exact answers to dense region queries. PA is the polynomial approximation method (Section 6), which returns approximate answers. We also compared PA with DH, the density histogram method used in the filtering step.

7.1 An Example

Figure 7(a) depicts a snapshot of the dataset CH10K. Figures 7(b) and 7(c) show the dense regions identified using the FR algorithm and the PA method, respectively. The dense regions found using both methods clearly have arbitrary shape and size, demonstrating their flexibility. Earlier methods only identified fixed-size square dense regions [7]. Further, the dense regions found by the PA method matches those found by the FR method very well.

7.2 Accuracy

We measure the accuracy of an approximated method using two metrics, the ratio of false positives r_{fp} and false negatives r_{fn}. Let D_1 be the union of all the regions dense under the query criteria, and let D_2 be the union of the dense regions actually identified by the method. Now, $r_{fp} = \frac{area(D_2 - D_1)}{area(D_1)}$, and $r_{fn} = \frac{area(D_1 - D_2)}{area(D_1)}$. We note that r_{fp} may exceed 100%, while r_{fn} never does.

We first compare the accuracies of PA and DH. The DH filtering step method divides cells into three categories: rejects, accepts, and candidate cells. We can either add all the candidate cells into the answer set, which we call *optimistic DH*, or reject all the candidate cells, which we call *pessimistic DH*. Optimistic DH has $r_{fn} = 0$ and $r_{fp} \geq 0$, while pessimistic DH has $r_{fp} = 0$ and $r_{fn} \geq 0$.

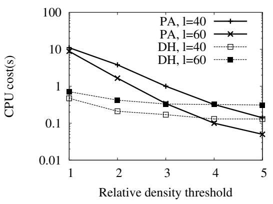
Figures 8(a) and 8(b) compare the error ratios with respect to l and the relative density threshold ϱ, using 1.0 Mb memory for PA and 1.2Mb memory for DH, in the default setting. The PA method has error ratio less than 10%, while the DH method may have false positive ratios r_{fp} up to 200% and false negative ratio r_{fn} up to 100%.

An interesting feature in Figure 8(a) and 8(b) is that increasing the relative density threshold reduces the union D_1 of dense regions, leading to higher error ratios.

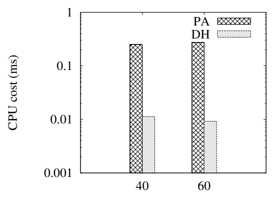
We examine the memory to error ratio trade-offs for PA and DH. The memory overhead is varied by varying the number and/or degree of polynomials for PA, or by varying the number of cells for DH. Figure 8(c) compares the false positive ratio of PA and optimistic DH, with $l = 40$ and $\varrho = 3$. Figure 8(d) compares the false negative ratio of PA and pessimistic DH. As expected, the error ratios for both two methods drop with increased memory overhead, because the approximation quality is improved. we see again that PA has a smaller error ratio than DH, even when DH is given 2–7 more memory than PA.

It seems clear that DH is not suitable for use by itself to evaluate PDR queries, and must be combined with plane-sweep in the refinement step. In contrast, the PA method provides very accurate answers to PDR queries, with a relatively small memory overhead.

Figure 9(a) shows the CPU costs for query evaluation for PA and DH. The CPU cost for DH remains almost unchanged as ϱ varies, since we must check the candidacy for each cell, regardless of the threshold. In contrast, the CPU cost for PA drops as ϱ increases from 1 to 5. As ϱ increases, the branch-and-bound technique becomes more effective in

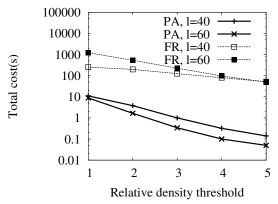


(a) Query CPU.

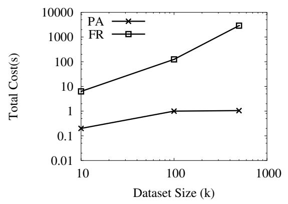


(b) Build CPU.

Fig. 9. CPU costs (dataset CH100K).



(a) Query cost(CH100K).



(b) Query cost vs. dataset.

Fig. 10. Cost.

pruning out non-dense regions (See Section 6.3). PA incurs lower CPU cost than DH when $\varrho > 4$.

Figure 9(b) shows the CPU costs to maintain the density histogram and the polynomials per location update. As expected, PA incurs an order of magnitude higher CPU cost than DH does, since PA requires computing the $\arccos()$ and $\sin()$ functions. However, DH has very poor accuracy, and must always be used with the far more expensive refinement step, which adds greatly to its true cost.

7.3 Query Costs

We now compare FR and PA in terms of the CPU and I/O costs of query evaluation. PA incurs no I/O at all, since we can hold all the polynomial coefficients in memory. FR incurs I/O costs, since it must evaluate spatio-temporal range queries over the TPR-tree during the refinement step. We measure the total query costs for FR as the sum of CPU and I/O cost, with each random I/O being charged 10ms.

Figure 10(a) compares the total query cost for PA and FR, with respect to ϱ, for the dataset CH100K. As in the default setting, PA used 400 degree-5 polynomials, while FR used 10,000 counters per density histogram. Clearly, PA has an order of magnitude lower query cost than FR, which must issue a large number of spatio-temporal range queries during refinement. Further, the plane-sweep algorithm during its refinement step is also more expensive than polynomial evaluation.

Figure 10(b) demonstrates the scalability of our methods, by showing how the query cost varies with respect to the number of moving objects, when $l = 40$ and $\varrho = 3$. We observe that the cost of FR is proportional to the dataset size. In contrast, the cost of PA remains almost the same as we increase the dataset size, since polynomial evaluation depends on the number of coefficients, not on the the number of moving objects. PA is likely to be appropriate when we need quick responses, and the dataset size is large.

8 Conclusion

In this paper, we propose a new definition of dense region queries. Under our definition, we are able to answer dense region queries completely and uniquely using this definition. Dense regions in our approach may have arbitrary shape and size, as well as local density guarantees. We present two methods, the first, an exact method, and the second, an approximate method. We demonstrate through extensive experiments that our exact method is efficient and is superior to current approaches. Our approximate method runs orders of magnitude faster than our exact method, at the cost of a tolerable loss of accuracy.

References

[1] Y. Cai and R. Ng. Indexing Spatio-temporal Trajectories With Chebyshev Polynomials. In *SIGMOD*, 2004.

[2] V. P. Chakka, A. Everspaugh, and J. M. Patel. Indexing Large Trajectory Data Sets With SETI. In *CIDR*, 2003.

[3] L. Forlizzi, R. H. Guting, E. Nardelli, and M. Schneider. A data model and data structures for moving objects databases. In *SIGMOD*, 2000.

[4] M. Hadjieleftheriou, G. Kollios, D. Gunopulos, and V. J. Tsotras. On-line discovery of dense areas in spatio-temporal databases. In *SSTD*, 2003.

[5] M. Hadjieleftheriou, G. Kollios, V. J. Tsotras, and D. Gunopulos. Efficient Indexing of Spatiotemporal Objects. In *EDBT*, 2002.

[6] C. S. Jensen, D. Lin, and B. C. Ooi. Query and Update Efficient B+-Tree Based Indexing of Moving Objects. In *VLDB*, 2004.

[7] C. S. Jensen, D. Lin, B. C. Ooi, and R. Zhang. Effective density queries on continuously moving objects. In *ICDE*, 2006.

[8] P. Kalnis, N. Mamoulis, and S. Bakiras. On Discovering Moving Clusters in Spatio-temporal Data. In *SSTD*, 2005.

[9] Y. Li, J. Han, and J. Yang. Clustering moving objects. In *KDD*, 2004.

[10] J. C. Mason and D. Handscomb. *Chebyshev Polynomials*. Chapman and Hall, 2003.

[11] J. Ni and C. V. Ravishankar. PA-Tree: A Parametric Indexing Scheme for Spatio-temporal Trajectories. In *SSTD*, 2005.

[12] D. Papadias, P. Kalnis, J. Zhang, and Y. Tao. Efficient OLAP Operations in Spatial Data Warehouses. In *SSTD*, 2001.

[13] D. Papadias, Y. Tao, P. Kalnis, and J. Zhang. Indexing spatio-temporal data warehouses. In *ICDE*, 2002.

[14] J. M. Patel, Y. Chen, and V. P. Chakka. STRIPES: An Efficient Index for Predicted Trajectories. In *SIGMOD*, 2004.

[15] D. Pfoser, C. S. Jensen, and Y. Theodoridis. Novel Approaches in Query Processing for Moving Object Trajectories. In *VLDB*, 2000.

[16] S. Saltenis, C. S. Jensen, S. T. Leutenegger, and M. A. Lopez. Indexing the Positions of Continuously Moving Objects. In *SIGMOD*, 2000.

[17] Z. Song and N. Roussopoulos. SEB-tree: An Approach to Index Continuously Moving Objects. In *Mobile Data Management*, pages 340–344, 2003.

[18] Y. Tao, D. Papadias, and J. Sun. The TPR*-Tree: An Optimized Spatio-Temporal Access Method for Predictive Queries. In *VLDB*, 2003.

[19] Y. Tao, J. Sun, and D. Papadias. Selectivity estimation for predictive spatio-temporal queries. In *ICDE*, 2003.

[20] M. L. Yiu and N. Mamoulis. Clustering objects on a spatial network. In *SIGMOD*, 2004.