Continual Neighborhood Tracking for Moving Objects Using Adaptive Distances

Yoshiharu Ishikawa† Hiroyuki Kitagawa† Tooru Kawashima‡*
†Institute of Information Sciences and Electronics
‡Master's Program in Science and Engineering
University of Tsukuba
{ishikawa,kitagawa}@is.tsukuba.ac.jp

Abstract

Based on the recent progress of digital cartography, global positioning systems (GPSs), and hand-held devices, there are growing needs of technology that provides neighborhood information to moving objects according to their locations and trajectories. In this paper, we propose spatial query generation models that take account of the current position and the past/future trajectories of a moving object to provide appropriate neighborhood information to it. For this purpose, we introduce an influence model of trajectory points *and derive neighborhood query generation models using adaptive* ellipsoid distances*. We describe query processing strategies for these query generation models and show* incremental query update *procedures to support* continual query *facilities with low processing cost. Finally, we present experimental results to show the effectiveness of our approach.*

1. Introduction

Mobile computing technology has gained much interest recently because of the advances of wireless communication, electronics, and positioning systems. The growth of mobile computing has brought a new field of database research area—*moving object* support based on database technology [1, 7, 12, 16, 21, 22, 28, 33, 34]. In this paper, we focus on the retrieval and presentation of neighborhood information for moving objects such as a vehicle equipped with a GPS and a navigation system, and people with hand-held mobile devices. By cooperating with forthcoming technology of intelligent traffic systems (ITSs), such *neighborhood tracking* functionalities would realize new types of applications that have "location- and situation-awareness".

We briefly explain the proposed idea using Fig. 1. Suppose that a vehicle, which came from the left side of the map, is now approaching the position $\boldsymbol{x}$. Its future trajectory is represented by the dotted arrow. Then consider the problem: "for the vehicle at the point $\boldsymbol{x}$, what is an appropriate *spatial query* to retrieve neighborhood information from a spatial database?" A simple approach would be to use the Euclidean distance from $\boldsymbol{x}$ to obtain objects in neighborhood as shown by the circle in Fig. 1; the query can be formulated as a *range query* or a *k-nearest neighbor query* (*k-nn query* for short). Although this approach is simple and clear, it often loses useful information since it does not consider the *trajectory* of the moving object, namely, the past locations and the predicted future ones. In contrast to this Euclidean approach, we use an ellipsoid region to retrieve neighborhood information. The ellipsoid region shown in Fig. 1 is computed based on the past and future trajectories of the object and slightly biased toward the "future"; namely, the center of the ellipsoid region is located at a predicted future position of the moving object and the shape of the region reflects the future trajectory than the past one.

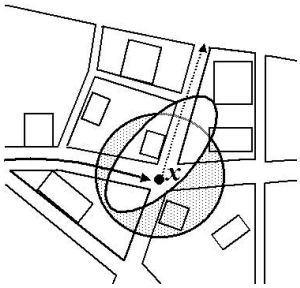

Figure 1. Retrieval of Neighborhood Information

Neighborhood queries in our framework are formalized as spatial queries based on the *ellipsoid distances* [2, 3, 19, 30] that have elliptic isosurfaces. We call such queries *ellipsoid queries*. Two benefits of using ellipsoid queries are summarized as follows:

- **Adaptiveness**: As shown in the following sections, ellipsoid queries can have arbitrary "shapes" that reflect the trajectories and the current positions of moving objects. By tuning query centers and ellipsoid distances adaptively for each trajectory point, our query generation method can generate neighborhood queries that retrieve appropriate neighborhood information along the trajectory of a moving object. Figure 2 illustrates the idea. Suppose that the moving object is located at the point $\boldsymbol{x}_{\tau-1}$ at the time $t = \tau-1$. After one unit time later (i.e., when $t = \tau$), the object moves to $\boldsymbol{x}_{\tau}$, then it goes to $\boldsymbol{x}_{\tau+1}$ at $t = \tau + 1$. Each ellipsoid shown in the figure is the isosurface of the spatial query issued at each trajectory point. As shown in the figure, three ellipsoids have different shapes in terms of rotation and thickness, and focus on the "neighborhood" of the moving object. Our proposed approach shown below provides such an adaptive neighborhood query generation facility for continuously moving objects. Additionally, our method offers some user-specifiable parameters to reflect users' preferences to the query generation.
- **Efficiency and simplicity**: For the efficient retrieval of neighborhood information from a huge spatial database, effective use of spatial indexes [13] is indispensable. Fortunately, there exist some query evaluation techniques proposed for ellipsoid distances that can effectively use spatial indexes [2, 30]. In this paper, we extend these techniques and apply them to our context. The retrieval algorithm shown below is implementable using conventional spatial indexes such as R-trees [17].

*Current affiliation: Hitachi Corporation

Proceedings of the International Database Engineering and Applications Symposium (IDEAS'02)
1098-8068/02 $17.00 © 2002 IEEE

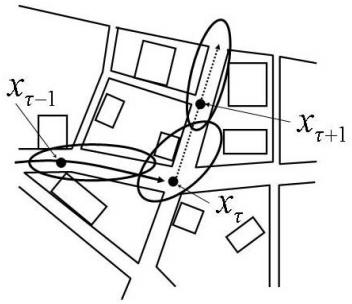

Figure 2. Adaptive Query Generation

As an alternative approach of neighborhood information retrieval, we could utilize spatial networks, consisting of spatial points, line-segments, and polygons, to process connectivity-based retrieval rather than proximity-based retrieval [31]. Although this approach may be able to perform a more detailed computation using connectivity information, it has to manage huge spatial network information and requires specialized data structures.

Our method has an additional feature: the support of *continual queries* [24]. The task to generate and process queries for a moving object along its trajectory is considered as a continual query task for the object. To generate and process queries continuously, we preserve the internal state used in the previous query generation and update the state incrementally. Since this proposed method has low update and storage costs, we can process continuous queries in an efficient manner. This feature would enhance real-time "tracking" capabilities often used in the analyses of spatio-temporal data [4].

The rest of the paper is organized as follows. In Section 2, we introduce some basic notions. The *influence model of trajectory points*, an important concept in this paper, is defined there. In Section 3, we derive some neighborhood query generation models taking account of the influence model. Section 4 introduces the notion of "continual query" task and presents query processing procedures to use spatial indexes efficiently, and Section 5 presents incremental query update methods. Section 6 shows experimental results and Section 7 describes the related work. Finally, Section 8 concludes the paper.

2. Basic Notions

In this section, we introduce basic notions used in the following discussion. Symbols and their meanings are shown in Table 1. Although our approach mainly targets geographical applications in low-dimensional space (two- or three-dimension), we derive formulas for arbitrary d dimension for the generality.

2.1. Representation of Trajectories

In our framework, neighborhood queries are generated based on the trajectory of a moving object, consisting of the past trajectory, the current position, and the future trajectory. Let the start time of a moving object be $t = 1$ and the current time be $t = \tau$. We denote each position of the object at time t ($t = 1, \ldots, \tau$) by a vector $\boldsymbol{x}_i = [x_{i1}, \ldots, x_{id}]^T$ in a d-dimensional space, where 'T' denotes vector transposition. Next, let the predicted time to arrive at the destination be $t = \tau + \tau'$; namely, we will need τ' unit times to arrive at the destination from the current position. The predicted positions of the object at $t = \tau + 1, \ldots, \tau + \tau'$ are also represented by d-dimensional vectors $\boldsymbol{x}_{\tau+1}, \ldots, \boldsymbol{x}_{\tau+\tau'}$. Figure 3 illustrates the definition.

Table 1. Symbols and Their Definitions

Name	Definition
d	number of dimensions
$\boldsymbol{x}_i$	location vector ($t = 1, \ldots, \tau, \ldots \tau + \tau'$)
τ	current time
τ'	predicted time to arrive at the destination from the current position
σ	"look ahead" parameter
μ	decay factor for past trajectory points
ν	decay factor for future trajectory points
$\alpha(t)$	influence value of the location information at time t
$\boldsymbol{q}$	query center
$\bar{\boldsymbol{x}}$	weighted average of location information
$\mathbf{A}$, $\mathbf{M}$	distance matrix and derived distance matrix
$\mathbf{C}$, $\tilde{\mathbf{C}}$	covariance matrices
$\mathcal{D}_{\mathrm{Euclid}}(\cdot, \cdot)$	the Euclidean distance function
$\mathcal{D}_{\mathbf{A}}(\cdot, \cdot)$	ellipsoid distance function
$\boldsymbol{s}^-$, $\boldsymbol{s}^+$	(past and future) state vectors
$\mathbf{C}^-$, $\mathbf{C}^+$	(past and future) covariance matrices

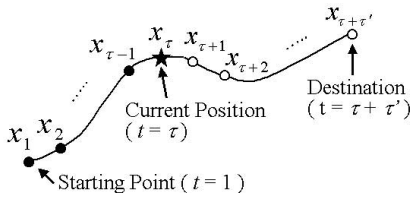

Figure 3. Representation of Trajectory

2.2. Influence Model of Trajectory Points

In the following, we utilize the past (and current) trajectory points $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_\tau$ and the predicted future trajectory points $\boldsymbol{x}_{\tau+1}, \ldots, \boldsymbol{x}_{\tau+\tau'}$ for the generation of the neighborhood query for the moving object located at $\boldsymbol{x}_\tau$. For this purpose, we introduce the *influence model of trajectory points*. This model is based on a simple idea that the influence of a trajectory point is the highest for the "current" position and decays gradually towards past and future. To realize this idea, we incorporate the following three parameters σ, μ, and ν:

- *"look ahead" parameter* σ: This parameter σ ($\sigma \geq 0$) is used to reflect user's preference such that the predicted position $\boldsymbol{x}_{\tau+\sigma}$, at where the moving object will arrive after σ unit times, is the most influential position to generate a neighborhood query. Consider an example: suppose that a car driver driving a vehicle has high interest to the neighborhood of the location at where he will arrive one minute later. In this case, he can set $\sigma = 1$ to reflect his preference, then $\boldsymbol{x}_{\tau+1}$ will have the highest influence.
- *decay factor for past trajectory points* μ: The parameter μ ($0 \leq \mu \leq 1$) is used to set exponential influence values for the past trajectory points. The influence values exponentially decay from $t = \tau + \sigma$ to $t = 1$.
- *decay factor for future trajectory points* ν: The parameter ν ($0 \leq \nu \leq 1$) is used to set exponential influence values for the future trajectory points. The influence values exponentially decay from $t = \tau + \sigma$ to $t = \tau + \tau'$.

By using the three parameters, the influence model of trajectory points is formally defined as follows:

Definition 2.1 The *influence value* for each trajectory point $\boldsymbol{x}_t$ $(t = 1, \ldots, \tau + \tau')$ is given by

$$\alpha(t) = \begin{cases} \mu^{\tau+\sigma-t} & (t = 1, \ldots, \tau + \sigma) \\ \nu^{t-\tau-\sigma} & (t = \tau + \sigma, \ldots, \tau + \tau'). \end{cases} \quad (1)$$

Figure 4 illustrates the notion of influence values. By tuning σ, μ, and ν appropriately, users can reflect their preferences in various situations. The query generation models shown in the next section are based on this model.

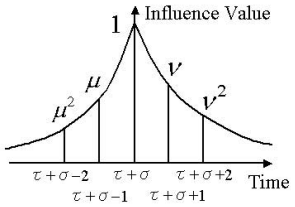

Figure 4. Influence Values

3 Query Generation Models

In the following, we derive neighborhood query generation models for a moving object at position $\boldsymbol{x}_\tau$. The neighborhood queries are constructed as spatial queries that retrieve data points from a spatial database. In general, queries in spatial databases can be specified in terms of the following three factors: 1) *query center* $\boldsymbol{q}$, 2) *distance function* $\mathcal{D}$, and 3) *query task* (e.g., range query and k-nn query). In this section, we focus on how to derive an appropriate query center $\boldsymbol{q}$ and a distance function $\mathcal{D}$ from the given trajectory information. Query tasks are discussed in Section 4.

3.1. Query Center Derivation Models

We introduce two derivation models of a query center. We intend that a user will select an appropriate one based on his or her requirement.

Model cur The model **cur** is a simple approach and does not use past/future trajectories except for $\boldsymbol{x}_{\tau+\sigma}$, the highest influence position. The query center is given by

$$\boldsymbol{q} = \boldsymbol{x}_{\tau+\sigma}. \quad (2)$$

Model avg The model **avg** fully uses the trajectory information and derives the query position as the weighted average of the trajectory points:

$$\boldsymbol{q} = \bar{\boldsymbol{x}} = \frac{\sum_{t=1}^{\tau+\tau'} \alpha(t)\boldsymbol{x}_t}{\sum_{t=1}^{\tau+\tau'} \alpha(t)}. \quad (3)$$

Note that the query center is determined by considering the recent and the near future trajectories, because the influence values ($\alpha(t)$'s) set high weights on the trajectory points around $\boldsymbol{x}_{\tau+\sigma}$.

3.2. Distance Function Derivation Models

We introduce three derivation models of distance functions.

Model EU The model **EU** is based on the ordinary Euclidean distance and introduced only for the comparison purpose. The Euclidean distance has less computational cost and clear semantics, but is not adaptive. In this approach, first a query center $\boldsymbol{q}$ is derived based on either of the query center derivation models, then a distance is computed for each object $\boldsymbol{x}$ by using the Euclidean distance $\mathcal{D}_{\mathrm{Euclid}}(\boldsymbol{x}, \boldsymbol{q})$. In the following, we denote a combination of a query center derivation model and a distance function derivation model such as **EU**(**cur**) and **EU**(**avg**).

Model OV In this model **OV** ('OV' stands for "oval"), we use ellipsoid distances as the distance functions. By setting parameters appropriately, we can tune the "shape" of a distance function based on the application need. The *ellipsoid distance* (also called elliptic or ellipsoidal distance) is often used in various application areas such as image retrieval [9], pattern recognition and classification [8], and statistics [20]. It is defined as follows:

Definition 3.1 An *ellipsoid distance* has the following *quadratic form*

$$\mathcal{D}^2_{\mathbf{A}}(\boldsymbol{x}_i, \boldsymbol{q}) = (\boldsymbol{x}_i - \boldsymbol{q})^T \mathbf{A} (\boldsymbol{x}_i - \boldsymbol{q}), \quad (4)$$

where $\mathbf{A}$ is a *symmetric positive definite matrix* ($\mathbf{A}^T = \mathbf{A}$ and $\boldsymbol{x}^T\mathbf{A}\boldsymbol{x} > 0$ for any $\boldsymbol{x} \neq \mathbf{0}$). We call $\mathbf{A}$ the *distance matrix* for $\mathcal{D}_{\mathbf{A}}(\cdot, \cdot)$.

An ellipsoid distance $\mathcal{D}_{\mathbf{A}}(\cdot, \cdot)$ defined by the above formula has arbitrary-rotated ellipsoidal isosurfaces. In a special case, if $\mathbf{A}$ is a unit matrix $\mathbf{I}$, the induced distance $\mathcal{D}_{\mathbf{I}}(\cdot, \cdot)$ agrees with the Euclidean distance (i.e., model **EU**).

An ellipsoid distance can be tuned by setting $\mathbf{A}$ appropriately. In this model **OV**, we derive an appropriate distance matrix based on the trajectory of a moving object using the influence model of trajectory points. For the derivation, we consider the following *penalty* formula:

$$P(\mathbf{A}, \boldsymbol{q}) = \sum_{t=1}^{\tau+\tau'} \alpha(t) \mathcal{D}^2_{\mathbf{A}}(\boldsymbol{x}_t, \boldsymbol{q}) \quad (5)$$

Since the influence factor $\alpha(t)$ is incorporated in the summation, the position $\boldsymbol{x}_{\tau+\sigma}$, the most important position, exert the highest influence on the penalty, and the effects of other positions decay gradually towards past and future.

To derive the optimal distance matrix $\mathbf{M}$ according to the penalty (Eq. (5)), we apply the optimization technique used in [19]; $\mathbf{M}$ is derived as the matrix that minimizes the penalty:

$$\mathbf{M} = \operatorname*{argmin}_{\mathbf{A}} P(\mathbf{A}, \boldsymbol{q}). \quad (6)$$

Since $\mathbf{M} = \mathbf{O}$ ($\mathbf{O}$ represents the null matrix) is obtained when we do not restrict $\mathbf{M}$, we set a constraint on $\mathbf{M}$ such that

$$\det(\mathbf{M}) = 1, \quad (7)$$

where $\det(\mathbf{M})$ is the determinant of $\mathbf{M}$. From this formulation, $\mathbf{M}$ can be derived as follows (the proof is shown in [18]).

Theorem 3.1 The matrix $\mathbf{M}$ that minimizes Eq. (5) under the constraint Eq. (7) can be derived as

$$\mathbf{M} = \det(\mathbf{C})^{\frac{1}{d}} \mathbf{C}^{-1}, \text{ where} \quad (8)$$

$$\mathbf{C} = \sum_{t=1}^{\tau+\tau'} \alpha(t)(\boldsymbol{x}_t - \boldsymbol{q})(\boldsymbol{x}_t - \boldsymbol{q}). \quad (9)$$

$\mathbf{C} = [c_{jk}]$ is called the *(weighted sample) covariance matrix* and $\det(\mathbf{C})$ is the determinant of $\mathbf{C}$. ■

The distance derived as above is a variation of *statistical distance* (also called *Mahalanobis distance*) often used in pattern analysis and multivariate statistics [8, 20]. In [19], we used a similar technique to derive distance functions based on user-specified examples for feedback-based interactive information retrieval. The statistical distance takes the spatial correlation of sample points into consideration and puts appropriate bias on each dimension so that it gains elliptic isosurfaces.

Now we mention the relationship between the **EU** model and the **avg** model. In Eq. (6), we have treated $\boldsymbol{q}$ as a constant, but we can treat $\boldsymbol{q}$ as an another optimization variable such as $(\mathbf{M},\, \boldsymbol{q}_{\mathrm{opt}}) = \mathrm{argmin}_{\mathbf{A},\, \boldsymbol{q}}\, P(\mathbf{A},\, \boldsymbol{q})$. In this case, we obtain $\boldsymbol{q}_{\mathrm{opt}} = \bar{\boldsymbol{x}}$ and the result coincides with **OV**(**avg**). This would be a good property to give a theoretical foundation to the **avg** model.

Model HB The model **OV** derived above has a benefit that it can utilize trajectory information to derive distance functions, but it lacks of robustness compared to the Euclidean distance. For example, if an object continuously moves along a straight line, the covariance matrix **C** becomes an ill-conditioned matrix, then the derived distance $\mathcal{D}_{\mathbf{M}}(\cdot,\, \cdot)$ tends to have too narrow isosurfaces. Moreover, in an extreme case, **C** approaches to a singular matrix and we cannot derive **M** using Eq. (8). This is because the model **OV** uses the spatial correlation of trajectory points; it requires d-dimensional spatial spreading of sample points.

To alleviate this problem, we introduce a hybrid model **HB** that integrates two models **EU** and **OV**; it inherits the robustness feature from **EU** and the adaptivity feature from **OV**. The idea is based on the heuristics to *regularize* ill-conditioned matrices to obtain non-singular covariance estimates [26]. In this model, we use the following matrix $\tilde{\mathbf{C}}$, instead of **C**, as the covariance matrix:

$$\tilde{\mathbf{C}} = \lambda \frac{\mathbf{C}}{\|\mathbf{C}\|} + (1 - \lambda) \frac{\mathbf{I}}{\|\mathbf{I}\|}, \tag{10}$$

where $\|\cdot\|$ is the Frobenius matrix norm [15] and works as weight normalization factors. The parameter λ ($0 \leq \lambda \leq 1$) specifies how to set weights to **EU** and **OV**. In this model, as in the case of **OV**, the distance matrix **M** is derived as follows:

$$\mathbf{M} = \det(\tilde{\mathbf{C}})^{\frac{1}{d}} \tilde{\mathbf{C}}^{-1}. \tag{11}$$

Note that when $\lambda = 0$ and 1, the **HB** model reduces to **EU** and **OV**, respectively. Therefore, we can say that **HB** is a generalized version of **EU** and **OV**.

4. Query Processing

In this section, we describe the query processing strategies for the query derivation models shown in Section 3.

4.1. Query Task

First, we introduce the notion of a query task. A *query task* specifies what should be retrieved when a query center $\boldsymbol{q}$ and a distance function $\mathcal{D}$ are given. As described in Section 3, a spatial query is fixed by specifying a query center, a distance function, and a query task. In our framework, query centers and distance functions are variables and change according to the movement of an object, while a query task is usually fixed throughout the movement. Therefore, we can say that a query task specifies a *continual query* for the moving object.

In this paper, we consider the following two query tasks:

- **range query task (equi-volume query task)**: On each movement, data objects within distance ε from the current query center $\boldsymbol{q}$ are retrieved. When a query center $\boldsymbol{q}$, a distance matrix **M**, and a distance value ε is given, we denote the ellipsoid region, centered at $\boldsymbol{q}$ and enclosed by the isosurface of distance ε, by

$$\mathrm{ellip}(\mathbf{M},\, \boldsymbol{q},\, \varepsilon) = \{\boldsymbol{p} \mid \boldsymbol{p} \in \mathcal{R}^d,\, \mathcal{D}^2_{\mathbf{M}}(\boldsymbol{p},\, \boldsymbol{q}) \leq \varepsilon^2\}. \tag{12}$$

- **k-nn query task**: For each movement, k nearest objects from $\boldsymbol{q}$ are retrieved.

Here we mention an important property of the range query task. The volume of an ellipsoid region $\mathrm{ellip}(\mathbf{M},\, \boldsymbol{q},\, \varepsilon)$ is given by the following formula [6]:

$$vol(\mathrm{ellip}(\mathbf{M},\, \boldsymbol{q},\, \varepsilon)) = \frac{\pi^{\frac{d}{2}}}{\Gamma(\frac{d}{2} + 1)} \frac{\varepsilon^d}{\det(\mathbf{M})^{\frac{1}{2}}}, \tag{13}$$

where $\Gamma(\cdot)$ is the gamma function. Since our ellipsoid distance derivation models **OV** and **HB** assure that $\det(\mathbf{M}) = 1$ as shown in Eq. (7), if ε is constant, the volume of each ellipsoid calculated on each object movement also becomes constant. Namely, if ε is constant, we can assure that the volume of the ellipsoid region for each range query is also constant throughout the movement. Therefore, we can say that the range query task is also an *equi-volume continual query* task.

4.2. Use of Bounding Regions

It is indispensable to use *spatial indexes* [13] effectively for the efficient query processing in spatial databases. Although we have to process ellipsoid queries for the **OV** and **HB** models, spatial indexes such as R-tree only support Euclidean distance-based queries and rectangle-based range queries in general. Therefore, we incorporate the approach proposed in [2] into our framework. The idea of the approach is to calculate the bounding region (bounding box or bounding sphere) that tightly bounds the given ellipsoid region then retrieve candidate objects using spatial indexes.

The *MBB* (*minimum bounding box*) *region* $\mathrm{MBB}(\mathbf{M},\, \boldsymbol{q},\, \varepsilon)$ that tightly bounds the ellipsoid region $\mathrm{ellip}(\mathbf{M},\, \boldsymbol{q},\, \varepsilon)$ (Eq. (12)) is given as follows [2]:

$$\mathrm{MBB}(\mathbf{M},\, \boldsymbol{q},\, \varepsilon)_i = \left[q_i - \varepsilon \sqrt{\mathbf{M}^{-1}_{ii}},\; q_i + \varepsilon \sqrt{\mathbf{M}^{-1}_{ii}} \right], \tag{14}$$

where $\mathrm{MBB}(\mathbf{M},\, \boldsymbol{q},\, \varepsilon)_i$ ($i = 1, \ldots, d$) denotes the range that $\mathrm{MBB}(\mathbf{M},\, \boldsymbol{q},\, \varepsilon)$ takes in the i-th dimension. $\mathbf{M}^{-1}_{ii}$ means the $(i,\, i)$ entry of the matrix $\mathbf{M}^{-1}$. This is illustrated in Fig. 5. The *MBS* (*minimum bounding sphere*) *region* $\mathrm{MBS}(\mathbf{M},\, \boldsymbol{q},\, \varepsilon)$ that tightly bounds the ellipsoid region $\mathrm{ellip}(\mathbf{M},\, \boldsymbol{q},\, \varepsilon)$ is derived as follows [2]:

$$\mathrm{MBS}(\mathbf{M},\, \boldsymbol{q},\, \varepsilon) = \left\{ \boldsymbol{p} \mid \boldsymbol{p} \in \mathcal{R}^d,\; \mathcal{D}^2_{\mathrm{Euclid}}(\boldsymbol{p},\, \boldsymbol{q}) \leq \frac{\varepsilon^2}{\lambda_{\min}} \right\}, \tag{15}$$

where $\lambda_{\min}$ is the smallest eigenvalue of **M**. Therefore, the radius of the MBS is given by $\varepsilon / \sqrt{\lambda_{\min}}$. This is illustrated in Fig. 6. The selection of the bounding region approximation method depends on the available spatial indexes; for example, if an R-tree index is available, we should use the MBB approximation.

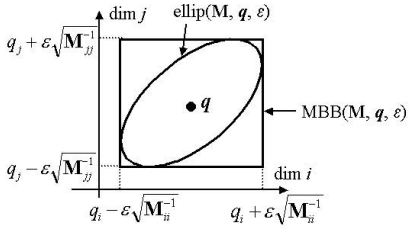

Figure 5. MBB-based Approximation

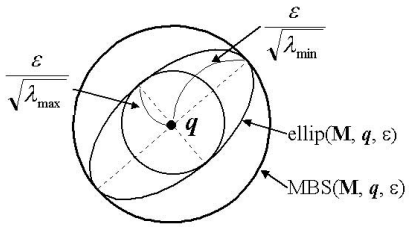

Figure 6. MBS-based Approximation

4.3. Query Processing Algorithms

In this subsection, we show query processing algorithms to process neighborhood queries using spatial indexes. We assume that the underlying spatial index module provides the following functions:

- `rect_search`(r): retrieves all the objects within a d-dimensional rectangle region r.
- `dist_search`($\boldsymbol{q}$, ε): retrieves all the objects $\boldsymbol{p}$ that satisfies $\mathcal{D}_{\mathrm{Euclid}}(\boldsymbol{p}, \boldsymbol{q}) \leq \varepsilon$.
- `knn_search`($\boldsymbol{q}$, k): retrieves the nearest k objects from $\boldsymbol{q}$ based on the Euclidean distance.

Since these search functions can be easily supported by traditional spatial indexes, the query processing procedures shown below are considered to be general ones.

Processing range queries The processing of a range query task is basically based on the approach in [2]. If we use the MBB-based approximation, we first issue a range query `rect_search`($\mathrm{MBB}(\mathbf{M}, \boldsymbol{q}, \varepsilon)$) for the filtering; note that according to the nature of the bounding region, the retrieved objects may contain *false alarms* but there are no *false dismissals* [11]. Therefore, we check whether each retrieved object $\boldsymbol{p}$ satisfies the original condition $\mathcal{D}_{\mathbf{M}}(\boldsymbol{p}, \boldsymbol{q}) \leq \varepsilon$ and discard false alarms to obtain the final result. If we use the MBS-based approximation, the query `dist_search`($\boldsymbol{q}, \varepsilon/\sqrt{\lambda_{\min}}$) is issued as the filtering query considering the spherical bounding relationship shown in Fig. 6.

Processing k-nn queries For k-nn queries based on ellipsoid distances, processing algorithms that effectively use conventional spatial indexes are shown in [2, 30]. However, these algorithms directly use the internal structures of spatial indexes. Instead of them, we use the following k-nn query processing procedure that is implementable using only the three basic search functions described above.

1. Issue `knn_search`($\boldsymbol{q}$, k), then get k-nn objects in terms of the Euclidean distance $\mathcal{D}_{\mathrm{Euclid}}(\cdot, \cdot)$. Let the Euclidean distance from $\boldsymbol{q}$ to the k-th object be δ.
2. Apply the range query procedure shown above to retrieve objects within the ellipsoid region $\mathrm{ellip}(\mathbf{M}, \boldsymbol{q}, \delta\sqrt{\lambda_{\max}})$.
3. Select k nearest objects in terms of the ellipsoid distance $\mathcal{D}_{\mathbf{M}}(\cdot, \cdot)$ from the objects retrieved in step 2.

The ellipsoid region $\mathrm{ellip}(\mathbf{M}, \boldsymbol{q}, \delta\sqrt{\lambda_{\max}})$ in Step 2 is an ellipsoid centered at $\boldsymbol{q}$ and tightly bounds a sphere centered at $\boldsymbol{q}$ with radius δ, as shown in Fig. 7. According to Step 1, we can find at least k objects within the ellipsoid $\mathrm{ellip}(\mathbf{M}, \boldsymbol{q}, \delta\sqrt{\lambda_{\max}})$. Therefore, it is assured that all the result objects of the original k-nn query are contained in this ellipsoid region. Therefore, we issue a range query to retrieve all the objects within $\mathrm{ellip}(\mathbf{M}, \boldsymbol{q}, \delta\sqrt{\lambda_{\max}})$ using the above procedure, then rank them in terms of the ellipsoid distance $\mathcal{D}_{\mathbf{M}}(\cdot, \cdot)$ to obtain the k-nn objects.

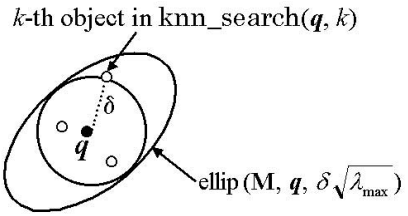

Figure 7. Processing k-NN Query

A similar k-nn query processing strategy is used in [23] to retrieve medical images with similar shapes based on the morphology concept. They used a cost-effective lower-bounding distance function instead of a costly morphological shape similarity function to determine the upper bound distance for the efficient k-nn query processing.

5. Incremental Query Update

5.1. Our Approach

When an object moves to the next position, we have to recalculate a new query center and a new distance function to generate the next neighborhood query. However, it is costly to recalculate all the required information from scratch and to maintain a long sequence of trajectory points permanently. In this section, we show incremental query update strategies which are efficient in terms of storage and computation costs.

Note that we do not reuse the previous query result for the next query processing. We simply process a new query without considering the previous query result. The reasons as follows:

1. If the spatial indexes are well-organized, page caching will reduce the I/O cost because continual object movement usually has page access locality.
2. It is difficult to reuse previous query result with low cost in our context since query centers and distance functions change every retrieval time.
3. Although ellipsoid distances are criticized by their computation costs in high dimensional spaces [2, 27, 30], our main target areas are low dimensional spaces such as $d = 2$ and 3 so that the calculation cost of ellipsoid distances in query processing is not the bottleneck of query processing.

5.2. Incremental Update Procedures

In this subsection, we assume that a moving object, which is located at $\boldsymbol{x}_\tau$ at $t = \tau$, actually arrives at the predicted next point $\boldsymbol{x}_{\tau+1}$ in time. Namely, the predicted arrival point $\boldsymbol{x}_{\tau+1}$,

Table 2. Changes of Influence Values

	$\boldsymbol{x}_1$	$\cdots$	$\boldsymbol{x}_\tau$	$\boldsymbol{x}_{\tau+1}$	$\cdots$	$\boldsymbol{x}_{\tau+\sigma}$	$\boldsymbol{x}_{\tau+\sigma+1}$	$\cdots$	$\boldsymbol{x}_{\tau+\tau'}$
$t=\tau$	$\mu^{\tau+\sigma-1}$	$\cdots$	μ^{σ}	$\mu^{\sigma-1}$	$\cdots$	1	ν	$\cdots$	$\nu^{\tau'-\sigma}$
$t=\tau+1$	$\mu^{\tau+\sigma}$	$\cdots$	$\mu^{\sigma+1}$	μ^{σ}	$\cdots$	μ	1	$\cdots$	$\nu^{\tau'-\sigma-1}$

the point where the object was supposed to reach at $t=\tau+1$, is approximately equal to the point $\tilde{\boldsymbol{x}}_{\tau+1}$ where the object is actually located at $t=\tau+1$ ($\boldsymbol{x}_{\tau+1} \approx \tilde{\boldsymbol{x}}_{\tau+1}$).

Changes of influence values When a moving object arrives at $\boldsymbol{x}_{\tau+1}$, we have to update the influence values on the trajectory points for the next query generation. When the current time changes from $t=\tau$ to $t=\tau+1$, the influence decay factors change as shown in Table 2. After the change, $\boldsymbol{x}_{\tau+\sigma+1}$ has the highest influence value for the query generation.

Update procedures for query centers and distance functions shown below have to reflect these changes of influence values.

Updating query centers For the model **cur**, we can simply set $\boldsymbol{x}_{\tau+\sigma+1}$ to the new query center $\boldsymbol{q}$. For the model **avg**, we can obtain an incremental update algorithm as follows. First, we translate Eq. (3) into

$$\bar{\boldsymbol{x}}|_\tau = \frac{\boldsymbol{s}^-|_\tau + \boldsymbol{s}^+|_\tau}{w|_\tau}, \tag{16}$$

where vectors $\boldsymbol{s}^-|_\tau$ and $\boldsymbol{s}^+|_\tau$ are defined as

$$\boldsymbol{s}^-|_\tau = \sum_{t=1}^{\tau+\sigma} \mu^{\tau+\sigma-t}\boldsymbol{x}_t \tag{17}$$

$$\boldsymbol{s}^+|_\tau = \sum_{t=\tau+\sigma+1}^{\tau+\tau'} \nu^{t-\tau-\sigma}\boldsymbol{x}_t. \tag{18}$$

The notation "$|_\tau$" represents that the variable is as of $t=\tau$. Scalar values $w|_\tau$ is defined by

$$w|_\tau = w^-|_\tau + w^+|_\tau \tag{19}$$

$$w^-|_\tau = \sum_{t=1}^{\tau+\sigma} \mu^{\tau+\sigma-t} = \frac{1-\mu^{\tau+\sigma}}{1-\mu} \tag{20}$$

$$w^+|_\tau = \sum_{t=\tau+\sigma+1}^{\tau+\tau'} \nu^{t-\tau-\sigma} = \frac{\nu(1-\nu^{\tau'-\sigma})}{1-\nu}. \tag{21}$$

When the object moves to $\boldsymbol{x}_{\tau+1}$ at $t=\tau+1$, we can update the state vectors by

$$\boldsymbol{s}^-|_{\tau+1} = \mu\boldsymbol{s}^-|_\tau + \boldsymbol{x}_{\tau+\sigma+1} \tag{22}$$

$$\boldsymbol{s}^+|_{\tau+1} = \frac{1}{\nu}\boldsymbol{s}^+|_\tau - \boldsymbol{x}_{\tau+\sigma+1}. \tag{23}$$

Then we can derive the update formula of the query center for the model **avg** as follows:

$$\bar{\boldsymbol{x}}|_{\tau+1} = \frac{\boldsymbol{s}^-|_{\tau+1} + \boldsymbol{s}^+|_{\tau+1}}{w|_{\tau+1}}. \tag{24}$$

For this algorithm, we have to store *only* two d-dimensional vectors $\boldsymbol{s}^-$ (Eq. (17)) and $\boldsymbol{s}^+$ (Eq. (18)) as internal states. When the object arrives at $\boldsymbol{x}_{\tau+1}$, we can update the query center using Eqs. (22), (23), and (24). The algorithm has a benefit that we do not have to maintain a long sequence of trajectory points; we can update $\boldsymbol{q}$ in *constant time* for fixed dimensionality d.

Updating distance functions For the models **OV** and **HB**, we have to update the distance matrix $\mathbf{M}$ according to the movement of an object. As shown in Eq. (9) and Eq. (10), an update of $\mathbf{M}$ can be reduced to an update of the covariance matrix $\mathbf{C}$. In the following, we show an incremental update procedure for the covariance matrix $\mathbf{C}$ when query centers are calculated by the **avg** model. Based on the similar approach, we can also derive the update procedure for the **cur** model [18].

Now we show only the derived update procedure for the **avg** model. Details of the derivation are described in [18]. First, we separate $\mathbf{C}|_\tau$ into two "past" and "future" covariance matrices $\mathbf{C}^-|_\tau = [c^-_{jk}|_\tau]$ and $\mathbf{C}^+|_\tau = [c^+_{jk}|_\tau]$:

$$\mathbf{C}|_\tau = \mathbf{C}^-|_\tau + \mathbf{C}^+|_\tau, \tag{25}$$

which are defined as

$$c^-_{jk}|_\tau = \sum_{t=1}^{\tau+\sigma} \mu^{\tau+\sigma-t}(x_{tj} - \bar{x}_j|_\tau)(x_{tk} - \bar{x}_k|_\tau) \tag{26}$$

$$c^+_{jk}|_\tau = \sum_{t=\tau+\sigma+1}^{\tau+\tau'} \nu^{t-\tau-\sigma}(x_{tj} - \bar{x}_j|_\tau)(x_{tk} - \bar{x}_k|_\tau) \tag{27}$$

Updates are processed as follows. First d-dimensional vectors $\boldsymbol{m}$ and $\boldsymbol{r}$ are calculated:

$$\boldsymbol{m} = \frac{-w^+|_\tau \boldsymbol{s}^-|_\tau + w^-|_\tau \boldsymbol{s}^+|_\tau}{w|_\tau} \tag{28}$$

$$\boldsymbol{r} = \frac{(\mu w|_\tau - w|_{\tau+1})\boldsymbol{s}^-|_\tau + (\frac{1}{\nu}w|_\tau - w|_{\tau+1})\boldsymbol{s}^+|_\tau}{w|_\tau w|_{\tau+1}} \tag{29}$$

Then a differential vector $\Delta\boldsymbol{x} = [\Delta x_1, \ldots, \Delta x_d]^T$ is calculated by

$$\Delta\boldsymbol{x} = \boldsymbol{x}_{\tau+\sigma+1} - \bar{\boldsymbol{x}}|_\tau, \tag{30}$$

and the state covariance matrices are updated:

$$\mathbf{C}^-|_{\tau+1} = \mu\left[\mathbf{C}^-|_\tau + \boldsymbol{r}\boldsymbol{m}^T + \boldsymbol{m}\boldsymbol{r}^T + w^-|_\tau \boldsymbol{r}\boldsymbol{r}^T\right] + (\Delta\boldsymbol{x} - \boldsymbol{r})(\Delta\boldsymbol{x} - \boldsymbol{r})^T \tag{31}$$

$$\mathbf{C}^+|_{\tau+1} = \frac{1}{\nu}[\mathbf{C}^+|_\tau - \boldsymbol{r}\boldsymbol{m}^T - \boldsymbol{m}\boldsymbol{r}^T + w^+|_\tau \boldsymbol{r}\boldsymbol{r}^T] - (\Delta\boldsymbol{x} - \boldsymbol{r})(\Delta\boldsymbol{x} - \boldsymbol{r})^T \tag{32}$$

For this distance function update algorithms, we have to maintain only two $d \times d$ matrices $\mathbf{C}^-$ (Eq. (26)) and $\mathbf{C}^+$ (Eq. (27)) as the state matrices and the update costs are *constant* for a fixed d. When the object arrive at $\boldsymbol{x}_{\tau+1}$, we can update the covariance matrix $\mathbf{C}|_\tau$ to $\mathbf{C}|_{\tau+1}$ using the above formulas. Similar to the case of the query centers, this algorithm has a benefit that we do not have to maintain a long sequence of trajectory points.

After the updates of the query center and the distance function, we invoke the neighborhood object retrieval procedure shown in Section 4 to perform the specified query task. Finally, we increment the current time as $\tau \leftarrow \tau+1$. Then we can use the same update procedure for the next point.

5.3. Practical Update Procedures

Unfortunately, the incremental update procedures shown above have two problems:

1. If $\boldsymbol{x}_{\tau+1}$, the predicted point for the time $t = \tau + 1$, and $\tilde{\boldsymbol{x}}_{\tau+1}$, the actually arrived point at $t = \tau + 1$, are quite different, the calculated query center and the distance function will drift from the "true" ones. This situation is caused by two reasons: 1) the moving object came to the point $\boldsymbol{x}_{\tau+1}$ earlier or later than expected, or 2) the moving object has changed its route.
2. The query center derivation model **avg** and the distance function derivation models **OV** and **HB** have a noise problem: even if $\boldsymbol{x}_{\tau+1} \approx \tilde{\boldsymbol{x}}_{\tau+1}$ is satisfied, the use of Eq. (18) and Eq. (32) to update $\boldsymbol{s}^+$ and $\mathbf{C}^+$ causes the amplification of small noises because $\nu < 1$. Therefore, repeated use of Eq. (18) and Eq. (32) for a number of incremental updates will result in incorrect query generation.

The practical solution for this problem is as follows:

- If the actual point $\tilde{\boldsymbol{x}}_{\tau+1}$ at $t = \tau + 1$ is mostly equal to the prediction ($\tilde{\boldsymbol{x}}_{\tau+1} \approx \boldsymbol{x}_{\tau+1}$), or the user allow small errors, apply the ordinal update procedure in Subsection 5.2.
- Otherwise, get a new prediction of future trajectory points $\boldsymbol{x}_{\tau+2}, \boldsymbol{x}_{\tau+3}, \ldots$ from the route calculation module (e.g., a car navigation system) then recompute statistics values $\boldsymbol{s}^+$ (required only for **avg**) and $\mathbf{C}^+$ (required for **OV** and **HB**).

This update strategy seems to be costly, but remember the role of the future decay factor ν; it sets high weights on the "near future" positions and discards far future ones. Therefore, the route calculation module has to calculate only a small number of trajectory positions that will be reached in the near future. Additionally, the recomputation cost of $\boldsymbol{s}^+$ and $\mathbf{C}^+$ would not be higher than the cost to estimate future trajectories often recalculated by the route calculation module in typical mobile applications.

Finally, we have to remind the benefit of our approach: we can "forget" the past information freely; in contrast to $\boldsymbol{x}^+$ and $\mathbf{C}^+$, we do not incur additional processing cost for $\boldsymbol{x}^-$ and $\mathbf{C}^-$.

6 Experimental Results

6.1. Behaviors of Query Generation Models

In the first experiment, we examine the difference between query generation models and their behaviors under different parameter settings. Figure 8 shows a trajectory of a moving object from A to B, and the object is currently located at $\boldsymbol{x}$. We assume that the object is moving with constant velocity and 35 trajectory points are taken along the trajectory. The figure shows the isosorfaces of queries generated by **EU**(**cur**) and **OV**(**cur**) under the conditions $\sigma = 0$ (no look ahead) and $\mu = \nu = 0.5$ (same decay late for the past and the future) for the range query task. As shown in the figure, the model **OV**, represented by an ellipse, is well conformed along the trajectory as expected.

Figure 9 shows isosurfaces of queries generated by **OV**(**cur**) under the conditions $\sigma = 0$ (no look ahead: represented by the solid ellipse centered at $\boldsymbol{q}_1$) and $\sigma = 5$ (highest weight on five unit times later: represented by the dotted ellipse centered at $\boldsymbol{q}_2$). Other conditions are the same as Fig. 8. The query center $\boldsymbol{q}_1$ for $\sigma = 0$ is approximately equal to the point $\boldsymbol{x}$, the current location of the moving object. We can easily observe that $\boldsymbol{q}_2$ is more biased towards the future trajectory because of the look ahead parameter σ.

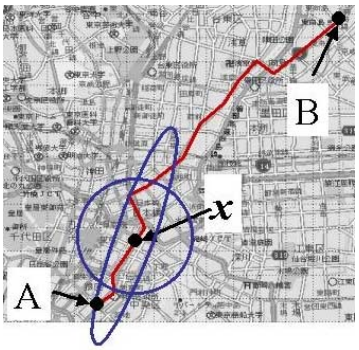

Figure 8. Comparison of EU and OV

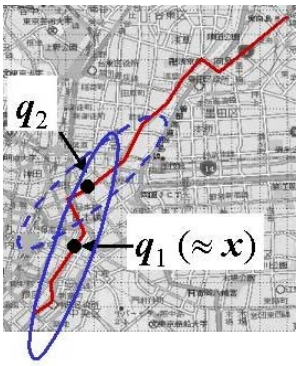

Figure 9. Comparison of Different σ-values ($\sigma = 0, 5$)

Figure 10 shows the behaviors of **OV**(**avg**) under two different ν values $\nu = 0.4$ (middle influence weights on future points: represented by the solid ellipse centered at $\boldsymbol{q}_1$) and $\nu = 0.9$ (high influence weights on future points: represented by the dotted ellipse centered at $\boldsymbol{q}_2$). Other parameters are set as $\mu = 0.4$ and $\sigma = 0$. The query center $\boldsymbol{q}_1$ is approximately the same as $\boldsymbol{x}$, the current location of the moving object. As shown in the figure, query $\boldsymbol{q}_2$ is more biased to the future because of the parameter setting of ν.

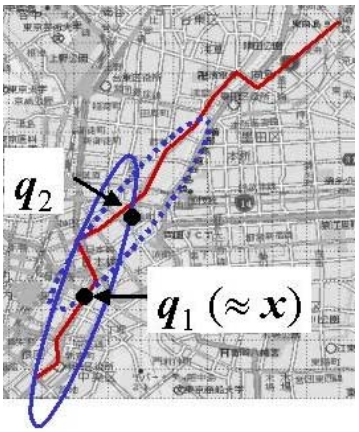

Figure 10. Comparison of Different ν-values ($\nu = 0.4, 0.9$)

Figure 11 shows the behaviors under different λ values for **HB**(**cur**): $\lambda = 1.0$ (the solid ellipse) and 0.7 (the dotted ellipse). The former case corresponds to **OV**(**cur**) since $\lambda = 1$. The small circle is an isosurface of **EU**(**cur**). As shown in the figure, **OV**(**cur**) has a narrow isosurface for this case because the trajectory nearby x is almost on the straight line. It may be an excessive behavior for a user who has interests only for the neighborhood information. In contrast to this, **HB**(**cur**) with $\lambda = 0.7$ has a more mild behavior; we can observe that the **HB** model has a mixed behavior of **OV** and **EU**.

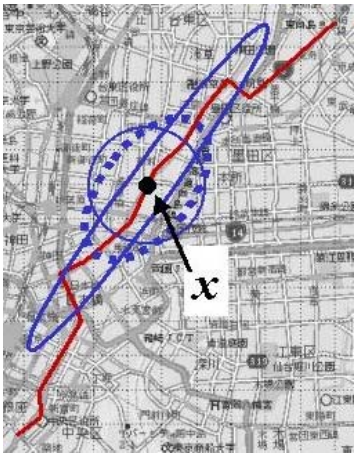

Figure 11. Comparison of Different λ-values ($\lambda = 1.0,\ 0.7$)

6.2. Continual Neighborhood Tracking

In this experiment, we perform a trace-based simulation of continual neighborhood tracking. Figure 12 shows a driving route from the point A to C via the intermediate point B. We assume that a vehicle make a short stop at B. From A, it takes about thirteen minutes to reach C. For this route, we collected real driving trace data, represented by the positions of a car, obtained for every five seconds of the drive. We put small circles on the trajectory for every thirty seconds based on the trace data. As shown in the figure, the driving speed is slow nearby B because the road is congested around there, and the vehicle has to turn down a side road to stop by the intermediate point B.

Figure 12. Driving Trace Data

In Fig. 13, the isosurfaces of neighborhood queries for the driving trace data is shown. To make the presentation understandable, isosurfaces are presented for every one minute of the driving. The neighborhood tracking is based on the range query task and the **OV**(**cur**) model with the parameters $\sigma = 1$, $\mu = 0.8$, and $\nu = 0.8$. As shown in the figure, each neighborhood query captures the local trajectory around it. Note that, for the straight roads driven with high speed, the isosurfaces take narrower shapes, and for the curved and crowded roads (especially around the point B) driven with slow speed, the isosurfaces take more rounded shapes. Based on this experiment, we can say that the proposed method can adaptively modify spatial queries according to the situation (the direction and the speed) of the moving object.

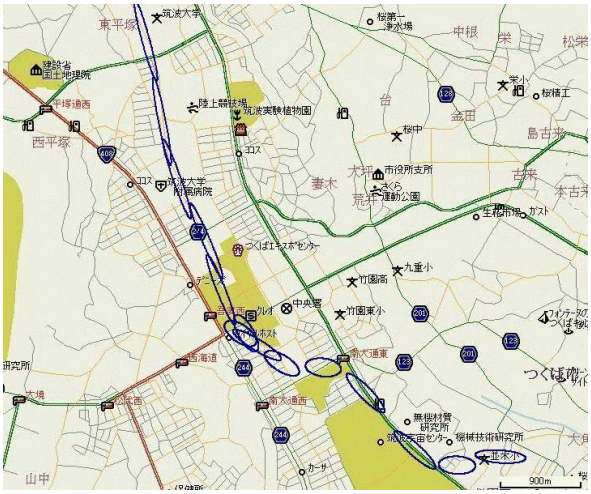

Figure 13. Neighborhood Tracking Result

6.3. Retrieval Cost

Finally we report the analysis of the retrieval cost. We use the Montgomery County dataset [10] with mid-points of road segments from the Montgomery County of Maryland that consists of 27,282 points (Fig. 14). We compare three approaches: sequential scan, the Euclidean distance-based approach, and the ellipsoid distance-based approach (**OV**(**cur**) is used). As the spatial index facility, we use the R*-tree extension of GiST [14]. We assume that an object moves along the road as shown in the figure; the road runs from south (Washington D.C.) to north and there are 62 mid-points of road segments on it. We also assume that the moving object specifies the k-nn query task ($k = 1,\ 10,\ 50,\ 100,\ 150$) and a k-nn query is issued on each of the 62 mid-points. We examine the number of page accesses to process k-nn queries.

Figure 15 shows the average number of page accesses to process k-nn queries on each query point. The x-axis represents the parameter k ($k = 1,\ 10,\ 50,\ 100,\ 150$) and the y-axis shows the number of page accesses in log-scale. In this experiment, we do not assume the existence of page caches. As shown in the figure, the page access cost of ellipsoid queries is slightly higher than that of the Euclidean distance-based queries (this property always holds because of the k-nn query procedure shown in Section 4), but quite lower than that of sequential scan. As shown in this experiment, the use of a spatial index based on the algorithms shown in Section 4 reduces the processing cost, and is indispensable in our context. Although this experiment does not assume the existence of a page cache, if a spatial index is well-organized and

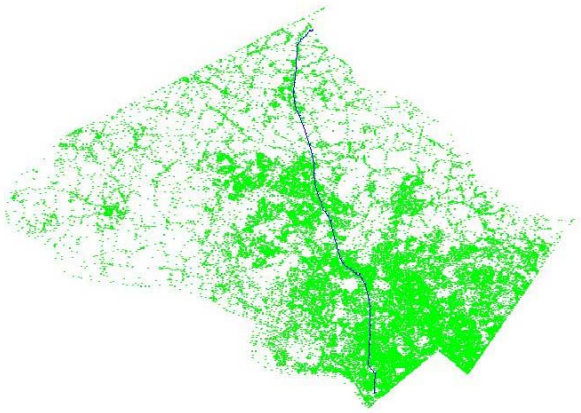

Figure 14. Montgomery County Dataset

the spatial locality is well preserved in each leaf page of the index, page caching will further reduce redundant page I/Os since the trajectories of moving objects have spatial contiguity. Based on this experiment we can say that our use of a spatial index facility will support the neighborhood tracking task with low cost.

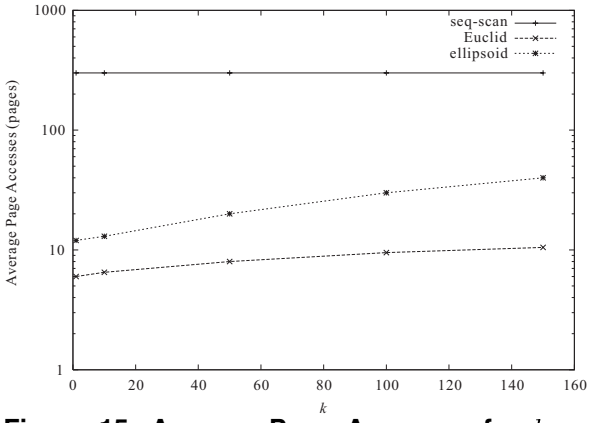

Figure 15. Average Page Accesses for k-nn Query

7. Related Work

Recent years, database technologies according to moving objects have been extensively developed: the main topics are data modeling issues [12, 16, 34] and efficient indexing and retrieval methods. In [1, 7, 21, 22, 28, 33], indexing methods are proposed to query moving objects stored in a database. In a typical setting, the trajectory of a moving object is represented by a function $\boldsymbol{x}(t)$ parameterized by time t. In many cases, a linear function is used as $\boldsymbol{x}(t)$ for its simplicity. In contrast to these indexing approaches aiming for efficient retrieval of moving objects stored in a database, we intend to provide neighborhood information for a moving object along its trajectory. Moreover, most of the indexing approaches for moving objects only consider static cases such that trajectories of objects are fixed beforehand. In contrast to this, we have considered a more dynamic situation and proposed adaptive query generation and processing strategies. And note that our approach is applicable to the parameterized representation of trajectories by transforming $\boldsymbol{x}(t)$ into a sequence of points using sampling.

The influence model of trajectory positions proposed in this paper is a model to set the highest importance to the current position and to discount the importance of past and future trajectory points exponentially. The approach setting exponential weights to past samples then focusing mainly on recent data samples is used in various areas that process time-varying data in an online manner, for example, in signal processing and control [5, 25], time-series analysis [35], and machine learning [32]. The approach is often called *exponential forgetting* or *exponential discounting* and the past decay parameter μ is called a *forgetting factor* or a *discount-rate parameter*. Our approach shares the same idea with these approaches but we also set exponential weights to future predicted positions to focus on the "near future". Sample weighting is also used in statistics, for example, to estimate covariance matrices in a robust manner by excluding the effects of outliers [29].

8. Conclusions and Future Work

In this paper, we proposed a new approach to retrieve and provide neighborhood information to moving objects. Our approach use the past and future trajectories of a moving object to generate an appropriate query that mainly focuses on the current (or near future) neighborhood of the object. For this purpose, we introduced the influence model of trajectory points to discount past and future positions gradually. In our context, queries are formulated as ellipsoid queries that can be tuned by considering the trajectories of moving objects and users' preferences. Since ellipsoid queries can be supported efficiently using conventional spatial indexes, our approach can be implemented with low overhead. Also, we presented the incremental query update procedures to generate the next query from the previous query states in an incremental manner. Therefore, our approach would be highly adaptive in practical situations.

In future work, we would like to develop a semi-automatic parameter tuning scheme that incorporates sensory inputs from the external devices to tune the parameters to realize "situation-aware" neighborhood information presentation systems.

Acknowledgments

This research is supported in part by the Grant-in-Aid for Scientific Research from the Japan Society for the Promotion of Science and the Ministry of Education, Culture, Sports, Science and Technology (#12480067 and #12780183). Also, financial support from Secom Science and Technology Foundation is gratefully acknowledged.

References

[1] P. K. Agarwal, L. Arge, and J. Erickson. Indexing Moving Points. In *Proc. ACM PODS*, pages 175–186, 2000.

[2] M. Ankerst, B. Braunmüller, H.-P. Kriegel, and T. Seidl. Improving Adaptable Similarity Query Processing Using Approximations. In *Proc. of VLDB*, pages 206–217, 1998.

[3] M. Ankerst, H.-P. Kriegel, and T. Seidl. A Multistep Approach for Shape Similarity Search in Image Databases. *IEEE TKDE*, 6:996–1004, 1998.

[4] ArcView Tracking Analyst Extension. http://www.esri.com/software/arcview/extensions/trackingext.html.

[5] K. J. Åström and B. Wittenmark. *Adaptive Control*. Addison-Wesley, 2nd edition, 1995.

[6] H. Cramér. *Mathematical Methods for Statistics* (10th printing). Princeton University Press, 1963. (first published in Sweden, Uppsala 1945, by Almqvist & Wiksells).

[7] O. Devillers, M. Golin, K. Kedem, and S. Schirra. Queries on Voronoi Diagrams of Moving Points. *Computational Geometry*, 6:315–327, 1996.

[8] R. O. Duda, P. H. Hart, and D. G. Stork. *Pattern Classification*. Wiley, 2nd edition, 2000.

[9] C. Faloutsos, R. Barber, M. Flickner, J. Hafner, W. Niblack, D. Petkovic, and W. Equitz. Efficient and Effective Querying by Image Content. *Journal of Intelligent Information Systems*, 3(3/4):231–262, July 1994.

[10] C. Faloutsos and I. Kamel. Beyond Uniformity and Independence: Analysis of R-trees Using the Concept of Fractal Dimension. In *Proc. ACM PODS*, pages 4–13, 1994.

[11] C. Faloutsos. *Searching Multimedia Databases by Content*. Kluwer Academic Publishers, 1996.

[12] L. Forlizzi, R. H. Güting, E. Nardelli, and M. Schneider. A Data Model and Data Structures for Moving Objects Databases. In *Proc. ACM SIGMOD*, pages 319–330, 2000.

[13] V. Gaede and O. Günther. Multidimensional Access Methods. *ACM Computing Surveys*, 30(2):170–231, 1998.

[14] The GiST Indexing Project. http://gist.cs.berkeley.edu:8000/gist/.

[15] G. H. Golub and C. F. Van Loan. *Matrix Computations*. Johns Hopkins University Press, Baltimore, MD, 3rd edition, 1996.

[16] R. H. Güting, M. H. Böhlen, M. Erwig, C. S. Jensen, N. A. Lorentzos, M. Schneider, and M. Vazirgiannis. A Foundation for Representing and Querying Moving Objects. *ACM TODS*, 25(1):1–42, 2000.

[17] A. Guttman. R-Trees: A Dynamic Index Structure for Spatial Searching. In *Proc. ACM SIGMOD*, pages 47–57, 1984.

[18] Y. Ishikawa, H. Kitagawa, and T. Kawashima. Continual Neighborhood Tracking for Moving Objects Using Adaptive Distances. Technical Report, ISE-TR-02-188, Institute of Information Science and Electronics, University of Tsukuba, Apr. 2002. (available from http://www.kde.is.tsukuba.ac.jp/)

[19] Y. Ishikawa, R. Subramanya, and C. Faloutsos. MindReader: Querying Databases through Multiple Examples. In *Proc. of VLDB*, pages 218–227, 1998.

[20] R. A. Johnson and D. W. Wichern. *Applied Multivariate Statistical Analysis*. Prentice Hall, 3rd edition, 1992.

[21] G. Kollios, D. Gunopulos, and V. J. Tsotras. Nearest Neighbor Queries in a Mobile Environment. In *Proc. of Intl. Workshop Spatio-Temporal Database Management (STDBM'99)*, volume 1678, pages 119–134, LNCS.

[22] G. Kollios, D. Gunopulos, and V. T. Tsotras. On Indexing Mobile Objects. In *Proc. ACM PODS*, pages 261–272, 1999.

[23] P. Korn, N. Sidiropoulos, C. Faloutsos, E. Siegel, and Z. Protopapas. Fast Nearest Neighbor Search in Medical Image Databases. *IEEE TKDE*, 10(6):889–904, 1998.

[24] L. Liu, C. Pu, and W. Tang. Continual Queries for Internet Scale Event-Driven Information Delivery. *IEEE Trans. on Knowledge and Data Engineering*, 11(4):610–628, 1999.

[25] L. Ljung and T. Söderström. *Theory and Practice of Recursive Identification*. MIT Press, 1983.

[26] T. Musha and Y. Okamoto. *Inverse Problems and Their Solutions*. Ohmsha, Ltd., Tokyo, Japan, 1992. (in Japanese).

[27] R. T. Ng and D. Tam. Multilevel Filtering for High-Dimensional Image Data: Why and How. *IEEE TKDE*, 11(6):916–928, 1999.

[28] D. Pfoser, C. S. Jensen, and Yannis Theodoridis. Novel Approaches in Query Processing for Moving Objects. In *Proc. VLDB*, pages 395–406, 2000.

[29] P. J. Rousseeuw and A. M. Leroy. *Robust Regression and Outlier Detection*. Wiley, 1987.

[30] T. Seidl and H.-P. Kriegel. Efficient User-Adaptable Similarity Search in Large Multimedia Databases. In *Proc. of VLDB*, pages 506–515, 1997.

[31] S. Shekhar and D.-R. Liu. CCAM: A Connectivity-Clustered Access Method for Networks and Network Computations. *IEEE TKDE*, 9(1):102–119, 1997.

[32] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 1998.

[33] S. Šaltenis, C. S. Jensen, S. T. Leutenegger, and M. A. Lopez. Indexing the Positions of Continuously Moving Objects. In *Proc. ACM SIGMOD*, pages 331–342, 2000.

[34] O. Wolfson, B. Xu, S. Chamberlain, and L. Jiang. Moving Objects Databases: Issues and Solutions. In *Proc. 10th SSDBM*, pages 111–122, Capri, Italy, 1998.

[35] B.-K. Yi, N. D. Sidiropoulos, T. Johnson, H. V. Jagadish, C. Faloutsos, and A. Biliris. Online Data Mining for Co-Evolving Time Sequences. In *Proc. ICDE*, pages 13–22, 2000.