Journal of Zhejiang University-SCIENCE C (Computers & Electronics)
ISSN 1869-1951 (Print); ISSN 1869-196X (Online)
www.zju.edu.cn/jzus; www.springerlink.com
E-mail: jzus@zju.edu.cn

Designing a location update strategy for free-moving and network-constrained objects with varying velocity[*#]

Yuan-Ko HUANG, Lien-Fa LIN

(*Department of Information Communication, Kao-Yuan University, Taiwan 821, Kaohsiung County*)

E-mail: huangyk@cc.kyu.edu.tw; lienfa@cc.kyu.edu.tw

Received Nov. 24, 2013; Revision accepted Mar. 14, 2014; Crosschecked July 16, 2014

Abstract: Spatio-temporal databases aim at appropriately managing moving objects so as to support various types of queries. While much research has been conducted on developing query processing techniques, less effort has been made to address the issue of when and how to update location information of moving objects. Previous work shifts the workload of processing updates to each object which usually has limited CPU and battery capacities. This results in a tremendous processing overhead for each moving object. In this paper, we focus on designing efficient update strategies for two important types of moving objects, free-moving objects (FMOs) and network-constrained objects (NCOs), which are classified based on object movement models. For FMOs, we develop a novel update strategy, namely the FMO update strategy (FMOUS), to explicitly indicate a time point at which the object needs to update location information. As each object knows in advance when to update (meaning that it does not have to continuously check), the processing overhead can be greatly reduced. In addition, the FMO update procedure (FMOUP) is designed to efficiently process the updates issued from moving objects. Similarly, for NCOs, we propose the NCO update strategy (NCOUS) and the NCO update procedure (NCOUP) to inform each object when and how to update location information. Extensive experiments are conducted to demonstrate the effectiveness and efficiency of the proposed update strategies.

Key words: Spatio-temporal databases, Moving objects, Free-moving objects, Network-constrained objects
doi:10.1631/jzus.C1300337   **Document code:** A   **CLC number:** TN929

1 Introduction

With the fast advance of mobile and ubiquitous computing, spatio-temporal databases (Forlizzi *et al.*, 2000; Güting *et al.*, 2000; Tao and Papadias, 2002; Huang and Lee, 2010) aiming at efficiently managing a large number of moving objects have become more realistic and attractive. Many applications, such as mobile communication systems, traffic monitoring systems, flight control systems, and geographical information systems, can benefit from the advent of spatio-temporal databases. We focus on the issue of managing two important types of moving objects in spatio-temporal databases. They are free-moving objects (FMOs) and network-constrained objects (NCOs), classified based on object movement models. Each FMO can move freely in any direction throughout the Euclidean space (e.g., a ship moving on the sea). In contrast, each NCO is constrained to move on a transportation network (e.g., a car or pedestrian moving on the roads). Due to the movement of objects (including FMOs and NCOs), the object information stored in spatio-temporal databases may be updated frequently, especially in a highly dynamic environment. As a result, how to effectively reduce the number of location updates from moving objects while appropriately maintaining their location information becomes a major challenge for spatio-temporal databases.

Previous research on reducing the cost of updating the location of moving objects can be classified

[*] Project supported by the National Science Council of Taiwan (Nos. NSC-102-2119-M-244-001 and MOST-103-2119-M-244-001)
[#] A preliminary version was presented at AINA, March 25–28, 2013, Barcelona, Spain

into three categories. Research in the first category (Song and Roussopoulos, 2001; Xiong *et al.*, 2005; 2006; Yu *et al.*, 2005) tracks the object locations only. As an example, consider the case in Fig. 1a, in which objects move in the Euclidean space (i.e., objects correspond to FMOs) and the curves represent the trajectories of moving objects. One update strategy is that each FMO reports its location to the database server every *t* timestamps (e.g., FMO_1), and the other strategy is to update the FMO location every *d* moving distance units (e.g., FMO_2). If objects move in the transportation network (Fig. 1b), where the movement of each object is constrained to the network and represented as a line segment, objects NCO_1 and NCO_2 correspond to the first and second update strategies, respectively. These two strategies are inefficient because the velocity, including speed and direction, of moving objects is not taken into consideration. For instance, an object moving with high speed (e.g., a car driving along a highway) may incur a large number of location updates as the time required to move *d* distance units is short.

In order to alleviate the problem of frequent updates, motion models are adopted in research of the second category (Sistla *et al.*, 1997; Wolfson *et al.*, 1999; Wolfson and Yin, 2003; Tao *et al.*, 2004) to predict the location of a moving object. As each object is aware of its real location obtained from a Global Positioning System (GPS) device, a location update is issued to the database server only when the deviation between the real and predicted locations exceeds a given threshold *T*. Consider the object FMO_3 in Fig. 1a. As long as the object's real location is within a circle C_T centered at the predicted location with radius *T*, its location information stored in the server does not need to be updated. For example, the solid line and dashed line refer to the real and predicted locations of object *o*, respectively. At time t_1, the location information does not need to be updated because *o* is still inside circle C_T. However, an update would occur at time t_2 as the deviation begins to exceed threshold *T*. Similarly, in Fig. 1b, object NCO_3 does not update the location information if it is on the segment S_T centered at the predicted location with length $2T$. We define circle C_T and segment S_T with a fixed size as the possible region in which the object lies.

To further reduce the update frequency, research in the third category (Cheng *et al.*, 2004; Chung *et al.*, 2009; Huang *et al.*, 2009; Chen *et al.*, 2010) takes advantage of the object's location, speed, and direction to construct a variable-size possible region, named the 'spatio-temporal possible region', whose location and size change as time progresses. As shown in Figs. 1a and 1b, objects FMO_4 and NCO_4 show the different shapes of spatio-temporal possible regions, called the 'sector-like region' and 'segment-bounded region', for the Euclidean space and transportation network, respectively. The sector-like region of moving object FMO_4 in the Euclidean space is constructed by considering object FMO_4's start location $l(t_0)$, and speed and direction varying within $[s_o, S_o]$ and $[\theta_o, \Theta_o]$, respectively. With $l(t_0)$, $[s_o, S_o]$, and $[\theta_o, \Theta_o]$, the sector-like region $R(t_i)$ of FMO_4 at time t_i can be computed and enclosed by four endpoints, two segments, and two arcs. The segment-bounded region $R(t_i)$ of object NCO_4 at time t_i is computed by considering only NCO_4's start location $l(t_0)$ and speed range $[s_o, S_o]$, because NCO_4's moving direction is constrained by the underlying network

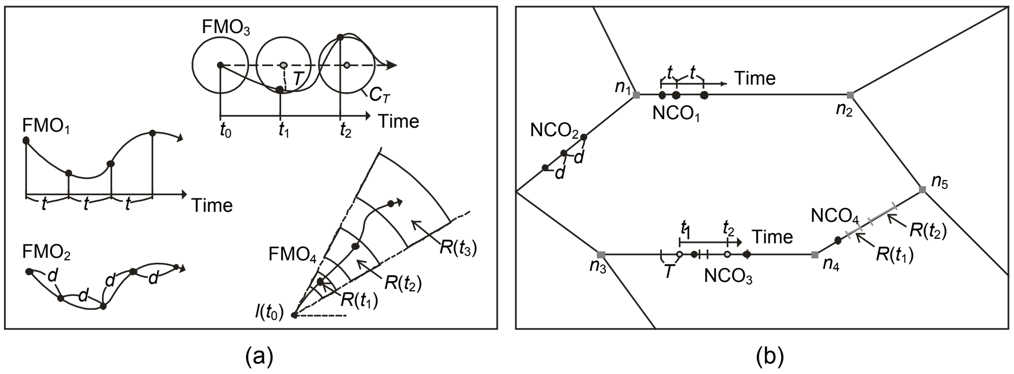


Fig. 1 Update strategies for moving objects: (a) free-moving objects (FMOs); (b) network-constrained objects (NCOs)

(i.e., fixed direction). In this paper, we adopt the sector-like region and segment-bounded region as both moving speed and direction are considered in these two types of regions.

Allowing each object to move inside the spatio-temporal possible region can significantly reduce the cost of communication between each object and the database server. However, it raises other crucial issues affecting the system performance even more. These issues include (1) when to update the spatio-temporal possible region of each object and (2) how to update the spatio-temporal possible region when object update occurs. To address these issues, previous studies (Wolfson and Yin, 2003; Cheng *et al.*, 2004; Huang *et al.*, 2009; Chen *et al.*, 2010) shifted the workload of updating the spatio-temporal possible region to each object. That is, each object needs to continuously check whether its real location is inside the spatio-temporal possible region. This would result in a tremendous processing overhead, especially for moving objects which usually have limited CPU and battery capacities. As such, we design two novel update strategies, namely the FMO update strategy (FMOUS) and the NCO update strategy (NCOUS), for each FMO in the Euclidean space and each NCO in the transportation network, respectively. The two update strategies can be used to explicitly indicate a time point t_u for FMO and NCO, at which it has to update its spatio-temporal possible region. The time point t_u is determined based on (1) whether the object's real location is inside the spatio-temporal possible region and (2) the sizes of the spatio-temporal possible region including the area of the sector-like region and the length of the segment-bounded region. In addition, we develop the FMO update procedure (FMOUP) and the NCO update procedure (NCOUP) to efficiently handle the updates of FMOs and NCOs, respectively. By exploiting these update strategies and update procedures, the computation cost at the client side can be effectively reduced.

This paper is an enhanced version of Huang *et al.* (2013), which focuses on FMOs only. The method is extended to be able to manage not only FMOs in the Euclidean space but also NCOs in the transportation network. In addition to the FMOUS and FMOUP in Huang *et al.* (2013), we design the NCOUS to give each NCO a time point at which its segment-bounded region needs to be updated, and develop the NCOUP to process the location updates of NCOs. A new and comprehensive empirical performance study of the proposed methods is reported.

2 Spatio-temporal possible region

In this section, we first discuss how to construct the spatio-temporal possible region of each FMO in the Euclidean space, in which the spatio-temporal possible region is represented as the sector-like region. Then, we describe the data structures used for representing the transportation network and how to derive the segment-bounded region of each NCO based on such data structures.

2.1 Sector-like region in the Euclidean space

For each FMO in the Euclidean space, its moving speed varies between a minimum and a maximum speed. Also, its moving direction lies in between a minimum and a maximum angle. Note that in our model, each angle is represented as a polar angle within [0, 2π]. For example, Fig. 2 shows that the speed and direction of an object o varies within $[s_o, S_o]$ and $[\theta_o, \Theta_o]$, respectively. When object o moves with the minimum speed s_o and minimum angle θ_o, its location at time t is computed as $l_\alpha(t)=l(t_0)+v_\alpha(t-t_0)$, where $l(t_0)=(x_o, y_o)$ is the start location, $v_\alpha=(s_o\cos\theta_o, s_o\sin\theta_o)$ is the velocity, and t_0 is the start time. When object o moves with the minimum speed s_o and maximum angle Θ_o, the location of object o at time t is $l_\gamma(t)=l(t_0)+v_\gamma(t-t_0)$, where $v_\gamma=(s_o\cos\Theta_o, s_o\sin\Theta_o)$. If the direction of o varies within $[\theta_o, \Theta_o]$, all of its possible locations at time t would form an arc, denoted as $\widehat{l_\alpha l_\gamma}(t)$, in which $l_\alpha(t)$ and $l_\gamma(t)$ are the two endpoints. That is, at time t object o is located at some

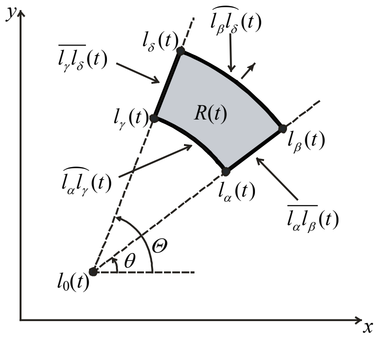

y
$\widehat{l_\beta l_\delta}(t)$
$l_\delta(t)$
$\overline{l_\gamma l_\delta}(t)$
$l_\gamma(t)$
$R(t)$
$l_\beta(t)$
$\widehat{l_\alpha l_\gamma}(t)$
$l_\alpha(t)$
$\overline{l_\alpha l_\beta}(t)$
Θ
θ
$l_0(t)$
x

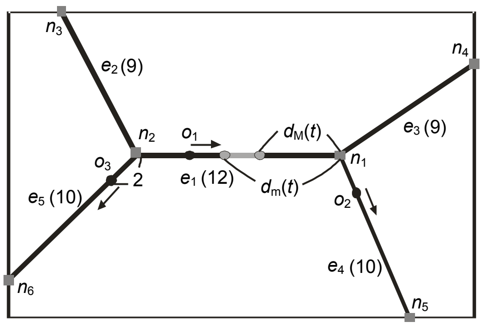
Fig. 2 Sector-like region

point on the arc $\widehat{l_\alpha l_\gamma}(t)$. Similarly, as the speed of object o is S_o and the direction is within $[\theta_o, \Theta_o]$, the possible locations of o can be represented by another arc $\widehat{l_\beta l_\delta}(t)$ whose two endpoints are $l_\beta(t)$ and $l_\delta(t)$. Hence, when the speed and direction are within $[s_o, S_o]$ and $[\theta_o, \Theta_o]$, respectively, all the possible locations of o form a sector-like region $R(t)$ (Fig. 2), which is enclosed by four endpoints $l_\alpha(t)$, $l_\beta(t)$, $l_\gamma(t)$, and $l_\delta(t)$, two segments $\overline{l_\alpha l_\beta}(t)$ and $\overline{l_\gamma l_\delta}(t)$, and two arcs $\widehat{l_\alpha l_\gamma}(t)$ (i.e., the nearest possible locations) and $\widehat{l_\beta l_\delta}(t)$ (i.e., the farthest possible locations).

2.2 Segment-bounded region in the transportation network

Each object NCO moves with a varying speed within $[s_o, S_o]$ and a fixed direction in a transportation network, which is represented as an undirected weighted graph consisting of a set of nodes and edges. To efficiently represent the transportation network, we use three tables to store the information of edges, nodes, and NCOs of the network. The first is the edge table T_e storing for each edge e_i of the transportation network: (1) its start node n_s and end node n_e (where $n_\mathrm{s}<n_\mathrm{e}$), (2) its length len (i.e., the distance between n_s and n_e), and (3) a set S_obj of moving objects currently on e_i. The second one is the node table T_n, which maintains for each node n_i the set S_adj of edges connecting n_i. The last one is the object table T_o maintaining the information of each NCO. T_o stores for each NCO o_i: (1) the edge e_j containing it, (2) the reference time t_r, (3) the distance dist between o_i and the start node n_s of edge e_j (i.e., $e_j.n_\mathrm{s}$) at t_r, (4) its minimum moving speed s_o, and (5) its maximum speed S_o. The three tables are updated only when objects reach the network nodes. Take a transportation network consisting of nodes n_1 to n_6 and edges e_1 to e_5 (Fig. 3a) to illustrate the information stored in the three tables T_e, T_n, and T_o, where an edge label is followed by its len enclosed in parentheses. In this transportation network, each of objects o_1 to o_5 moves with a minimum speed of 1 m/s and a maximum speed of 2 m/s, and its moving direction is indicated by the corresponding arrow. Fig. 3b shows the detailed information of tables T_e, T_n, and T_o. Note that a positive speed value of an object in T_o (e.g., object o_2) indicates that this object moves from the start node to the end node, and that a negative value (e.g., object o_1) indicates that it moves in the opposite direction.

Using these three tables, the segment-bounded region $R(t)$ of each NCO in the transportation network can be computed. Consider an NCO o that moves with a speed within $[s_o, S_o]$. Having accessed tables T_o and T_e, we know that object o is located on the edge e_j whose start node and end node are n_s and n_e, respectively. If object o moves with the minimum speed s_o, then its distance to node n_s at time t can be computed as $d_\mathrm{m}(t)$=dist+$s_o(t-t_\mathrm{r})$. Conversely, if o moves with the maximum speed S_o, then its distance to node n_s at time t is represented as $d_\mathrm{M}(t)$=dist+$S_o(t-t_\mathrm{r})$. When the speed of object o varies within $[s_o, S_o]$, all the possible locations of o would be bounded by the line segment $\overline{d_\mathrm{m}d_\mathrm{M}}(t)$ whose length is equal to $\|d_\mathrm{M}(t)-d_\mathrm{m}(t)\|$. Taking object o_1 in Fig. 3a as an example, its minimum speed s_o and maximum speed S_o are equal to −1 and −2, respectively, because it moves toward the start node n_1. The distance $d_\mathrm{m}(t)$ of object o_1 to node n_1 at time 2 is computed as 8−1×2=6. Also, the distance $d_\mathrm{M}(t)$ of o_1 to n_1 is equal to 8−2×2=4. Finally,



(a)

T_e

e_i	n_s	n_e	len	S_obj
e_1	n_1	n_2	12	{o_1}
e_2	n_2	n_3	9	null
e_3	n_1	n_4	9	null
e_4	n_1	n_5	10	{o_2}
e_5	n_2	n_6	10	{o_3}

T_n

n_i	S_adj
n_1	{e_1, e_3, e_4}
n_2	{e_1, e_2, e_5}
n_3	{e_2}
n_4	{e_3}
n_5	{e_4}
n_6	{e_5}

T_o

o_i	e_j	t_r	dist	s_o	S_o
o_1	e_1	0	8	-1	-2
o_2	e_4	0	2	1	2
o_3	e_5	0	2	1	2

(b)

Fig. 3 Segment-bounded region: (a) a transportation network; (b) three tables

the segment-bounded region $\overline{d_m d_M}(t)$ of object o_1 (depicted as the shaded line) has a length of |4−6|=2.

3 FMOUS and FMOUP in the Euclidean space

In this section, we describe how FMOUS indicates a time point at which each FMO needs to update location information and how to process such an update using FMOUP.

3.1 FMOUS

Due to varying traffic conditions such as jams and accidents on the road, an object FMO may move out of its sector-like region at some time point t. As a result, the new location, speed, and direction of the FMO have to be updated at time t. Furthermore, because the scope of the sector-like region $R(t)$ of an FMO grows as time progresses, the FMO has to update its precise location information to decrease the number of possible locations. Thus, FMOUS is designed to determine the time point t_u at which the FMO needs to update its location information, according to (1) time point t_α at which the FMO moves out of its sector-like region $R(t_\alpha)$ and (2) time point t_β at which the size of its sector-like region $R(t_\beta)$ exceeds a system parameter $A_{\max}$. In the following, we describe how to obtain t_α and t_β.

Assume that at time t_c an object o whose location is (x_o^c, y_o^c) changes its moving speed to x_o^c and/or moving angle to θ_o^c. As (x_o^c, y_o^c) is within the corresponding sector-like region $S(t_c)$, o will be out of $R(t_\alpha)$ at a future time t_α when (1) o moves out from arc $\widehat{l_\beta l_\delta}(t_\alpha)$, (2) o moves out from arc $\widehat{l_\alpha l_\gamma}(t_\alpha)$, (3) o moves out from segment $\overline{l_\alpha l_\beta}(t_\alpha)$, or (4) o moves out from segment $\overline{l_\gamma l_\delta}(t_\alpha)$.

The first condition holds only for the case in which the moving speed s_o^c is greater than the maximum speed S_o. As shown in Fig. 4a, object o moves towards the direction indicated by the arrow and its s_o^c is larger than S_o. In this case, object o would move out from arc $\widehat{l_\beta l_\delta}(t_\alpha)$ and t_α can be estimated as follows:

$$t_\alpha = t_c + \frac{S_o(t_c - t_0) - \sqrt{C}}{\sqrt{A+B} - \sqrt{C} - S_o}, \tag{1}$$

where

$$\begin{cases} A = (x_o^c + s_o^c \cos\theta_o^c - x_o)^2, \\ B = (y_o^c + s_o^c \sin\theta_o^c - y_o)^2, \\ C = (x_o^c - x_o)^2 + (y_o^c - y_o)^2. \end{cases}$$

The second condition is $s_o^c < s_o$. In this case, o will move out from arc $\widehat{l_\alpha l_\gamma}(t_\alpha)$ at time t_α. Substituting s_o for S_o in Eq. (1) leads to the t_α in this condition.

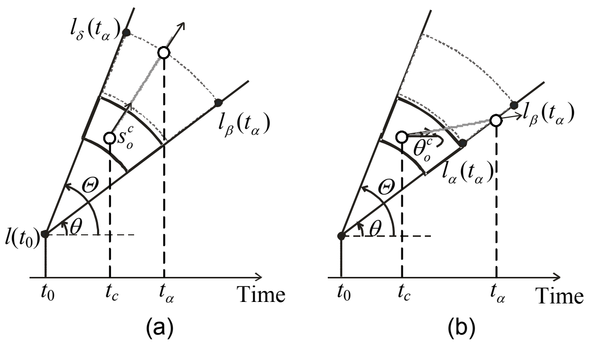


Fig. 4 Determining the time point t_α at which the object moves out from the arc (a) or the segment (b)

The third condition is that object o changes its moving angle to θ_o^c at time t_c which is less than the minimum angle θ_o. Therefore, o will move out from segment $\overline{l_\alpha l_\beta}(t_\alpha)$ at time t_α (Fig. 4b). Let (x_r, y_r) be the point on segment $\overline{l_\alpha l_\beta}(t_\alpha)$ where object o moves out of region $R(t_\alpha)$, computed as follows:

$$\begin{cases} x_r = \dfrac{(y_o^c - x_o^c \tan\theta_o^c) - (y_o - x_o \tan\theta_o)}{\tan\theta_o - \tan\theta_o^c}, \\ y_r = x_r \tan\theta_o + y_o - x_o \tan\theta_o. \end{cases} \tag{2}$$

Then t_α can be estimated as

$$t_\alpha = t_c + \frac{\sqrt{(x_o^c - x_r)^2 + (y_o^c - y_r)^2}}{s_o^c}. \tag{3}$$

The last condition holds only if θ_o^c is greater than the maximum angle Θ_o. This means that at time t_α object o moves out from a point (x_l, y_l) on segment $\overline{l_\gamma l_\delta}(t_\alpha)$, where

$$\begin{cases} x_1 = \dfrac{(y_o^c - x_o^c \tan\theta_o^c) - (y_o - x_o \tan\Theta_o)}{\tan\Theta_o - \tan\theta_o^c}, \\ y_1 = x_r \tan\Theta_o + y_o - x_o \tan\Theta_o. \end{cases} \tag{4}$$

Also, t_α can be determined by Eq. (3) after replacing x_r and y_r by x_1 and y_1, respectively.

To maintain the location information of moving objects, a system parameter $A_{\max}$ can be used to limit the number of possible locations inside the sector-like region of each FMO. For each object o, if the size of its sector-like region $R(t_\beta)$ at time t_β exceeds $A_{\max}$, then o needs to update the location, speed, and direction to the server. The time point t_β is computed as follows:

$$t_\beta = t_0 + \sqrt{\frac{360A_{\max}}{\pi(\Theta_o - \theta_o)(S_o^2 - s_o^2)}}. \tag{5}$$

Thus, t_α and t_β are both determined. The time point t_u at which object o needs to update its location information is then determined by $t_u=\min(t_\alpha, t_\beta)$. Benefiting from FMOUS, each FMO knows in advance when to update its location information. This means that the object does not have to continuously check its location within time interval $[t_0, t_u]$, and thus the processing overhead can be greatly reduced.

3.2 FMOUP

Once an update of FMO occurs, FMOUP is adopted to rapidly renew the location information of the object in the database server and determine a new sector-like region for this FMO. Let us use the example in Fig. 5 to illustrate how FMOUP works:

Step 1: At time t_0, FMO o sends its real location (x_o, y_o), moving speed s, and moving angle θ to the location-based database server.

Step 2: At the server side, the sector-like region $R(t)$ for FMO o is determined depending on factors such as the historical object information (e.g., location, speed, and angle), the traffic condition, and the location update frequency.

Step 3: The determined $R(t)$ and the system parameter $A_{\max}$ are sent to FMO o.

Step 4: Once FMO o changes its moving speed or direction at time t_c, the time point t_α is computed. Also, the time point t_β can be estimated. Then, the update time t_u is set to $\min(t_\alpha, t_\beta)$.

Step 5: At time t_u at which FMO o needs to issue an update, the location (x_u, y_u), speed s_u, and angle θ_u are sent to the location server.

Step 6: According to (x_u, y_u), s_u, and θ_u, the server determines a new sector-like region $R(t)$ for FMO o.

Step 7: The new $R(t)$ is sent to FMO o.

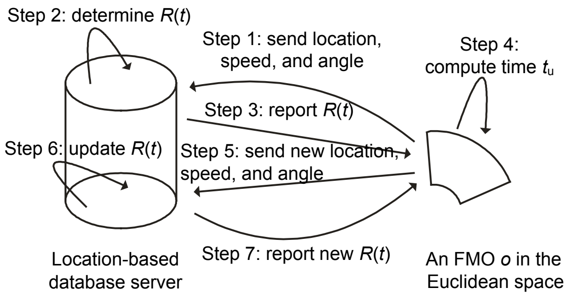

Step 2: determine R(t)
Step 1: send location, speed, and angle
Step 4: compute time t_u
Step 3: report R(t)
Step 6: update R(t)
Step 5: send new location, speed, and angle
Location-based database server
Step 7: report new R(t)
An FMO o in the Euclidean space

Fig. 5 Schematic of the free-moving object update procedure (FMOUP)

4 NCOUS and NCOUP in the transportation network

4.1 NCOUS

Compared to FMOUS for the Euclidean space, NCOUS for the transportation network is more complex since the time point at which NCO reaches a node in the network would affect the update time t_u. As a result, NCOUS is proposed to determine t_u by considering (1) the time point t_α at which NCO moves out of its segment-bounded region $R(t_\alpha)$, (2) the time point t_β at which the length of the segment-bounded region $R(t_\beta)$ is greater than a system parameter $L_{\max}$, and (3) the time point t_δ at which the endpoint $d_M(t_\delta)$ of the segment-bounded region $R(t_\delta)$ reaches a node in the network. The determination of t_α, t_β, and t_δ is detailed in the following.

Let object o be the NCO that changes its moving speed to s_o^c at time t_c and dist^c be the distance of object o to the start node n_s at time t_c. The time point t_α is then determined based on whether s_o^c is (1) less than the minimum speed s_o, or (2) greater than the maximum speed S_o. If $s_o^c<s_o$, at time t_α object o will be out of segment-bounded region $R(t_\alpha)$ through endpoint $d_m(t_\alpha)$. For example, in Fig. 6a, the speed of object o (depicted as the white point) is changed to s_o^c at time t_c and segment-bounded region $R(t_c)$ is represented as

the shaded line segment. As $s_o^c < s_o$, object o will move out from $d_m(t_\alpha)$ at time t_α. In this case, t_α can be estimated as

$$t_\alpha = t_c + \frac{(\text{dist}^c - \text{dist}) - s_o(t_c - t_r)}{s_o - s_o^c}, \tag{6}$$

where t_r is the reference time point and dist is the distance of o to node n_s at t_r.

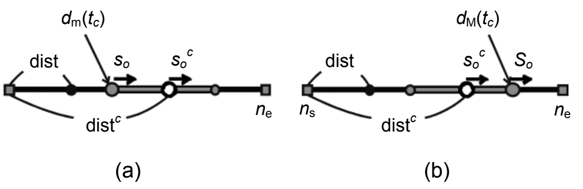


Fig. 6 Determining the time point t_α at which the object moves out from $d_m(t)$ (a) or $d_M(t)$ (b)

When $s_o^c > S_o$, at time t_α object o will be out of segment-bounded region $R(t_\alpha)$ through endpoint $d_M(t_\alpha)$ (Fig. 6b). Hence,

$$t_\alpha = t_c + \frac{(\text{dist} - \text{dist}^c) - S_o(t_c - t_r)}{s_o^c - S_o}. \tag{7}$$

Given a system parameter $L_{\max}$, time point t_β is computed based on whether the length of segment-bounded region $R(t)$ of NCO is greater than $L_{\max}$. That is, at t_β, $L_{\max}=|d_m(t_\beta)-d_M(t_\beta)|$. As $d_m(t_\beta)=\text{dist}+s_o(t_\beta-t_r)$ and $d_M(t_\beta)=\text{dist}+S_o(t_\beta-t_r)$, we have

$$t_\beta = t_r + \frac{L_{\max}}{|s_o - S_o|}. \tag{8}$$

For an application in which the location information of moving objects needs to be as precise as possible, $L_{\max}$ will be set to a lower value. Otherwise, a higher value of $L_{\max}$ is preferable.

For each NCO o, if at time t_δ the endpoint $d_M(t)$ of its segment-bounded region reaches a node in the transportation network, then o needs to update its information stored in tables T_e and T_o. For example, in Fig. 3, if o moves from the previous edge e_p to the next edge e_n at time t_δ, it will be removed from the set S_{obj} of e_p and then added into that of e_n. As for table T_o, the values of e_j, t_r, and dist are updated to e_n, t_δ, and 0 (or e_n.len), respectively, where dist=0 (or dist=e_n.len) as $d_M(t_\delta)$ reaches the start node n_s (or end node n_e) of edge e_n.

The determination of the time point t_δ depends on whether object o moves towards the start node n_s or the end node n_e. If o moves towards n_s, then the time point t_δ at which $d_M(t_\delta)$ reaches n_s is computed as

$$t_\delta = t_r + \frac{-\text{dist}}{S_o}. \tag{9}$$

Otherwise, the time point t_δ at which $d_M(t_\delta)$ reaches n_e is computed as

$$t_\delta = t_r + \frac{e_n.\text{len} - \text{dist}}{S_o}. \tag{10}$$

Having determined t_α, t_β, and t_δ, the time t_u is then determined by $t_u=\min(t_\alpha, t_\beta, t_\delta)$.

4.2 NCOUP

The main goal of NCOUP is to quickly evaluate the location update issued from each NCO and reassign a new segment-bounded region for NCO. Similar to FMOUP (Section 3.2), NCOUP works as follows (Fig. 7):

Step 1: At time t_0, NCO o sends an update in the form of (e, dist, s, t_0) to the location-based database server, which indicates that o moves on edge e at time t_0 and its moving speed and distance to the start node of e are s and dist, respectively.

Step 2: At the server side, the three tables T_e, T_n, and T_o are modified according to the tuple (e, dist, s, t_0) of NCO o. In addition, the minimum speed s_o and the maximum speed S_o in table T_o are determined, depending on factors such as the historical object information, the traffic condition, and the location update frequency, to construct the segment-bounded region $R(t)$ for NCO o.

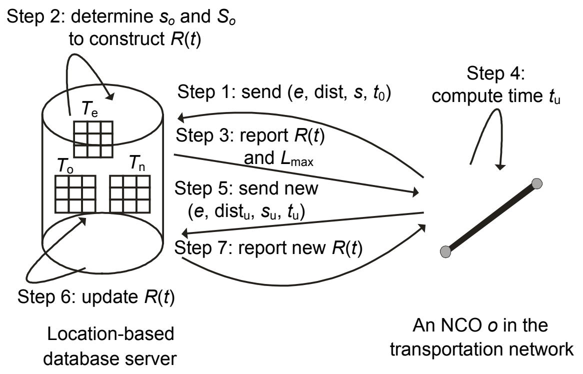
Step 3: The determined $R(t)$ and the system parameter $L_{\max}$ are sent to NCO o.

Step 4: When NCO o changes its moving speed to s_o^c at time t_c, the time points t_α, t_β, and t_δ are calculated. The update time t_u is then set to $\min(t_\alpha, t_\beta, t_\delta)$.

Step 5: At t_u, the distance dist_u to the start node and the updated speed s_u of NCO o are sent to the location server.

Step 6: According to dist_u and s_u, the server determines a new segment-bounded region $R(t)$ for NCO o.

Step 7: The new $R(t)$ is sent to NCO o.



Step 2: determine s_o and S_o to construct $R(t)$
T_e
T_o
T_n
Step 1: send (e, dist, s, t_0)
Step 3: report $R(t)$ and L_{max}
Step 5: send new (e, dist_u, s_u, t_u)
Step 7: report new $R(t)$
Step 4: compute time t_u
Step 6: update $R(t)$
Location-based database server
An NCO o in the transportation network

Fig. 7 Schematic of the network-constrained object update procedure (NCOUP)

5 Performance evaluation

In this section, we conducted two sets of experiments for the proposed update strategies, i.e., FMOUS for the Euclidean space and NCOUS for the transportation network. In the first set of experiments, the efficiency of FMOUS was demonstrated based on comparison with other strategies in terms of (1) the number of updates issued from FMOs and (2) the processing overhead for the update strategies. The second set of experiments was aimed to investigate the performances of NCOUS and its competitors by measuring the number of object updates and the processing cost.

5.1 Experimental settings

All experiments were performed on a PC with Intel Core Duo 1.8 GHz CPU and 1 GB RAM. The algorithms were implemented in C++. One synthetic dataset and one real dataset were used in the simulation for FMOUS. The synthetic dataset consisted of 100k FMOs whose start locations were uniformly spread over a region of 1 000 km×1 000 km. As for the real dataset, we used the CA file (TIGER/Line, http://www.census.gov/geo/www/tiger) which contains 62k FMOs distributed over a region of 25 000 km×25 000 km. The minimum speed s_o and maximum speed S_o of an FMO o were randomly generated between 0 and 90 m/s. For the moving direction of FMO o, the minimum angle θ_o was randomly generated within [0, 2π], and the maximum angle $\Theta_o=\theta_o+\theta'$, where $\theta'=\pi/6$, $\pi/3$, or $\pi/2$. The initial speed (resp. angle) of an FMO was randomly assigned a value bounded by s_o and S_o (resp. θ_o and Θ_o). To evaluate the performance of NCOUS, we used two road maps and generated 100k NCOs using the generator proposed by Brinkhoff (2002). The first road map was Oldenburg, a city in Germany, consisting of about 6000 nodes and 7000 edges, and the second was the San Joaquin County with about 18 200 nodes and 23 800 edges. The minimum speed s_o and maximum speed S_o of each NCO were uniformly distributed between 0 and 40 m/s. Then, the initial speed of each NCO was set to a value within [s_o, S_o]. When an NCO o reached a node n in the network, the next edge on which o moved was randomly chosen among the edges connecting n. In the experimental space, a percentage P_o of FMOs (resp. NCOs) would change the moving speed and/or direction (resp. moving speed only) every T_o time units, where P_o and T_o were equal to 5% and 10, respectively, unless otherwise specified. The assignment of the new speed and angle was the same as that of the initial speed and angle. In this simulation, the evaluation time interval had a default value of 100 time units, the default $A_{\max}$ for FMO was set to 0.1% of the entire space, and the default $L_{\max}$ for NCO was set to 0.1% of the entire network.

The performance was measured by the number of object updates and the processing overhead for the proposed update strategies FMOUS and NCOUS. We compared FMOUS and NCOUS against the fixed time interval update strategy (FIUS for short) in which the location information of each object (including FMO and NCO) was updated every T_o time units, and the circular region update strategy (CRUS for short) in which an update occurred if the location of FMO was not inside the circle C_T whose size was the same as $A_{\max}$ or if the location of NCO was not inside the segment S_T whose size was the same as $L_{\max}$.

5.2 Efficiency of FMOUS

We first compared the proposed FMOUS with FIUS and CRUS using the synthetic dataset and CA dataset. Four sets of experiments were conducted to investigate the effects of two important factors, P_o and

T_o, on the number of object updates and the processing overhead for these update strategies.

Fig. 8 shows the number of object updates for FMOUS and its competitors under various P_o. Note that hereafter a logarithmic scale is used for the y-axis of all figures. Figs. 8a and 8b illustrate the number of updates for the synthetic dataset and CA dataset, respectively, by varying P_o from 5% to 30%. FMOUS significantly outperformed FIUS, even for a large value of P_o (e.g., 30%), which indicates that many of FMOs have changed their speeds and directions. The reason is that for FIUS each object has to update its location information every 10 time units even if its speed or direction does not change, whereas for FMOUS unnecessary updates can be avoided by allowing each object to move within the sector-like region. Similarly, the performance gap between FMOUS and CRUS increased with the increase of P_o. This is mainly due to the fact that, for CRUS some objects with varying speeds and directions could easily move out of the circle C_T whose size is fixed and thus object update occurs while for FMOUS such objects are still inside their sector-like regions and thus object updates are not needed.

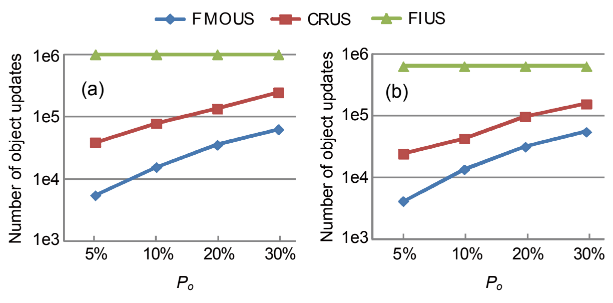


Fig. 8 Number of object updates vs. P_o: (a) synthetic dataset; (b) CA dataset

Fig. 9 shows the impact of T_o on the performances of FMOUS, FIUS, and CRUS. In these experiments, we varied the value of T_o from 5 to 50 time units and measured the number of object updates for FMOUS, FIUS, and CRUS. Figs. 9a and 9b show that for all update strategies, the number of object updates decreased with the increase of T_o. This is because a greater T_o (i.e., a longer time interval within which FMO does not change speed or direction) would allow more FMOs to delay their updates to the location server so that the total number of updates decreases. In both sets of experiments, FMOUS outperformed FIUS and CRUS in the entire range of T_o. The experiments confirmed again that using FMOUS can dramatically reduce the number of object updates.

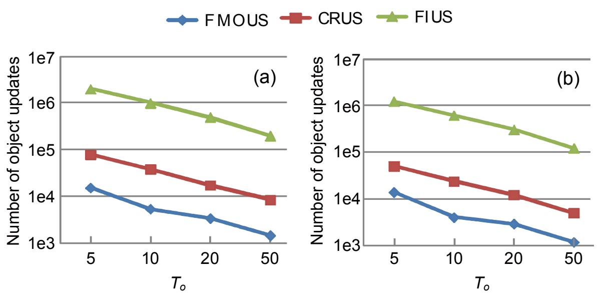


Fig. 9 Number of object updates vs. T_o: (a) synthetic dataset; (b) CA dataset

The following two sets of experiments demonstrated the effectiveness of FMOUS by studying the effects of the number of objects changing their speeds or directions (i.e., P_o) and the time interval within which the object's speed and direction remained unchanged (i.e., T_o) on the processing overhead of FMOUS, FIUS, and CRUS. For FIUS, the processing overhead is the number of time units of the time interval within which the FMO needs to update its location information. For FMOUS and CRUS, the processing overhead is the number of time units of the time interval within which the FMO has to check whether its real location is out of sector-like region $R(t)$ and circle C_T, respectively.

Fig. 10 shows the processing overhead of FMOUS, CRUS, and FIUS as a function of P_o varying from 5% to 30%. CRUS had the worst performance among the three update strategies and its processing overhead remained constant for different values of P_o. The main reason is that in CRUS each FMO does not know when to update its location and thus it has to check its location at each time instant, regardless of whether the moving speed or direction changes. Similarly, FIUS had a constant overhead under various P_o for both the synthetic dataset and CA dataset. This is because for FIUS all objects would process location updates every 10 time units. In both experiments, FMOUS outperformed its competitors significantly as using FMOUS to give each object a time point at which to update its location can effectively reduce the processing overhead.

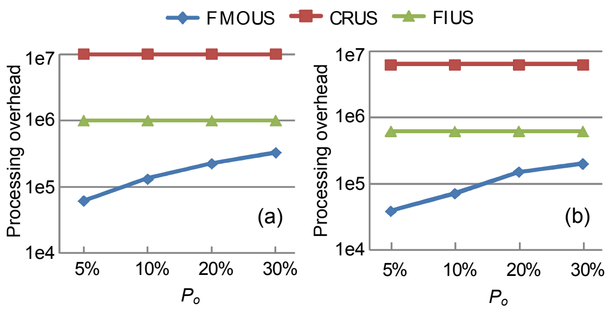


Fig. 10 Processing overhead vs. P_o: (a) synthetic dataset; (b) CA dataset

The last set of experiments shows the processing overhead of FMOUS, CRUS, and FIUS under various T_o (Fig. 11). Similar to the experiments demonstrated by Fig. 9, we varied the value of T_o from 5 to 50 time units to investigate the effect of the time interval within which FMO does not change speed or direction. When each FMO changed speed or direction frequently (e.g., T_o=5), FMOUS outperformed CRUS or FIUS by a factor of about 100 or 20, respectively. When T_o>5, FMOUS reduced the processing overhead by a factor of up to 200 compared to CRUS and a factor of 20 compared to FIUS.

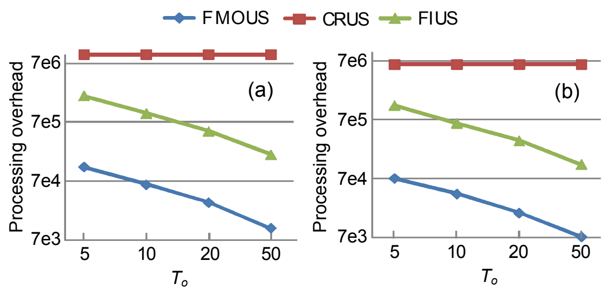


Fig. 11 Processing overhead vs. T_o: (a) synthetic dataset; (b) CA dataset

5.3 Efficiency of NCOUS

To demonstrate the efficiency of NCOUS, we compared it with FIUS and CRUS in terms of the number of object updates and the processing overhead. We studied the impacts of P_o and T_o on the performances of NCOUS, FIUS, and CRUS.

Fig. 12 shows the number of object updates for NCOUS, FIUS, and CRUS under various P_o and T_o. As shown in Fig. 12a, FIUS had the worst performance in all cases because of its simple update strategy in which the object reports the location information periodically. When P_o increased, the number of object updates for NCOUS and CRUS increased, because (1) for NCOUS more NCOs could reach the network nodes or move out of the segment-bounded region $R(t)$ and (2) for CRUS more NCOs are not inside their corresponding segment S_T so that more updates occur. Fig. 12b shows the number of object updates under different values of T_o. The curves for all update strategies showed decreasing trend, following a similar reasoning for the experiment demonstrated by Fig. 9.

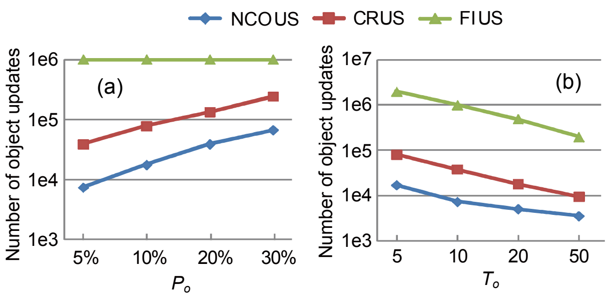


Fig. 12 Number of object updates with varying P_o (a) and varying T_o (b)

Fig. 13 shows how the varying P_o and T_o affect the processing overhead of NCOUS, FIUS, and CRUS. For CRUS, the processing overhead remained 10^7. The reason for this high processing cost is that at each time unit, each NCO inevitably checks whether its real location is inside the segment S_T. As for FIUS, its processing overhead is dominated by T_o and is insensitive to P_o, because the processing cost is required only when NCO updates its location information. The experimental results showed that NCOUS has a significantly better performance in terms of processing overhead compared to CRUS and FIUS.

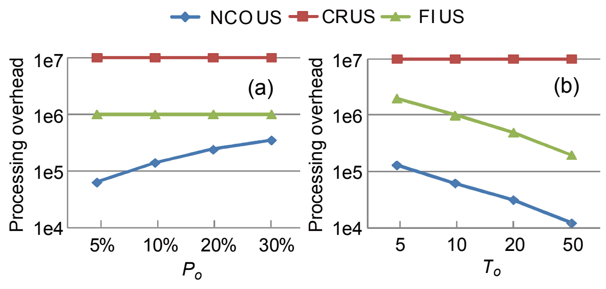


Fig. 13 Processing overhead with varying P_o (a) and varying T_o (b)

The last experiment was aimed to study how well NCOUS, FIUS, and CRUS work for a larger transportation network, San Joaquin County. Figs. 14a and 14b show the number of object updates for the three strategies under various P_o and T_o, respectively. Compared to the experimental results in Fig. 12 for a small transportation network, Oldenburg, the number of object updates for NCOUS slightly increased. The reason for the higher update number is that the road connectivity of the larger network is more complicated so that more time points t_δ at which NCO reaches a network node exist. Nevertheless, NCOUS still significantly outperformed FIUS and CRUS in all cases. Figs. 14c and 14d show that NCOUS had almost the same processing overhead in comparison with Fig. 13. From the experimental results, we know that NCOUS is also suitable for a transportation network with more complicated road connectivity.

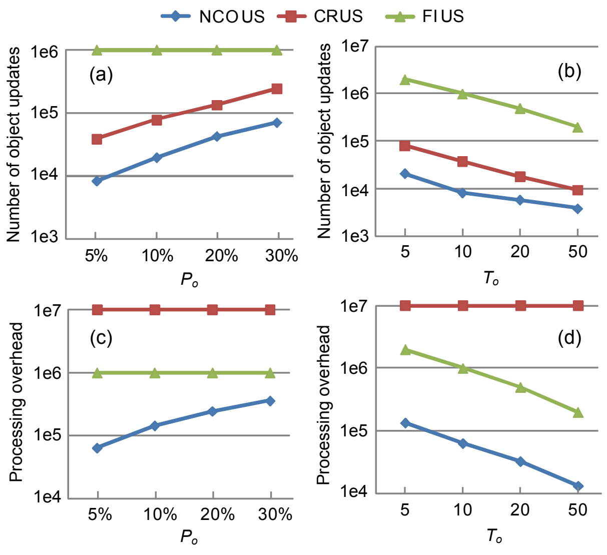


Fig. 14 The performances of the three strategies for a larger network, San Joaquin County
(a) Update number vs. P_o; (b) Update number vs. T_o; (c) Processing overhead vs. P_o; (d) Processing overhead vs. T_o

6 Conclusions

This paper focuses on developing efficient update strategies to process the location updates issued from moving objects (including FMOs and NCOs) with varying speeds and directions. Previous work shifted the workload of processing updates to each object whose CPU and battery capacity are limited, which would result in a tremendous processing overhead. To efficiently process object updates, for FMOs we designed FMOUS to explicitly give each FMO a time point at which to update its location information. In addition, FMOUP was used to process object updates. Similarly, for NCOs, we proposed NCOUS and NCOUP to inform each object when and how to update location information. Experimental results showed that the proposed update strategies can greatly reduce the processing overhead for each object while maintaining the location information.

There are several interesting avenues for future extensions of this work. Our next step is to properly represent the transportation network and conduct more experiments investigating the performances of different update strategies. An important research direction is how to efficiently answer the spatio-temporal queries, such as the range query and K-nearest neighbor query, on FMOs with sector-like regions and NCOs with segment-bounded regions. A further extension is to develop a specialized index structure for managing FMOs and NCOs.

References

Brinkhoff, T., 2002. A framework for generating network-based moving objects. *GeoInformatica*, **6**(2):153-180. [doi:10.1023/A:1015231126594]

Chen, S., Ooi, B.C., Zhang, Z., 2010. An adaptive updating protocol for reducing moving object database workload. Int. Conf. on Very Large Data Bases, p.735-746.

Cheng, R., Kalashnikov, D.V., Prabhakar, S., 2004. Querying imprecise data in moving object environments. *IEEE Trans. Knowl. Data Eng.*, **16**(9):1112-1127. [doi:10.1109/TKDE.2004.46]

Chung, B.S.E., Lee, W.C., Chen, A.L.P., 2009. Processing probabilistic spatio-temporal range queries over moving objects with uncertainty. Int. Conf. on Extending Database Technology, p.60-71.

Forlizzi, L., Güting, R.H., Nardelli, E., *et al.*, 2000. A data model and data structures for moving objects databases. Int. Conf. on ACM Management of Data, p.319-330.

Güting, R.H., Bohlen, M.H., Erwig, M., *et al.*, 2000. A foundation for representing and querying moving objects. *ACM Trans. Database Syst.*, **25**(1):1-42. [doi:10.1145/352958.352963]

Huang, Y.K., Lee, C., 2010. Efficient evaluation of continuous spatio-temporal queries on moving objects with uncertain velocity. *GeoInformatica*, **14**(2):163-200. [doi:10.1007/s10707-009-0081-8]

Huang, Y.K., Liao, S.J., Lee, C., 2009. Evaluating continuous K-nearest neighbor query on moving objects with uncertainty. *Inform. Syst.*, **34**(4-5):415-437. [doi:10.1016/j.is.2009.01.001]

Huang, Y.K., Su, I.F., Lin, L.F., *et al.*, 2013. Efficient processing of updates for moving objects with varying speed and direction. Int. Conf. on Advanced Information Networking and Applications, p.854-861.

Sistla, A.P., Wolfson, O., Chamberlain, S., *et al.*, 1997. Modeling and querying moving objects. Int. Conf. on Data Engineering, p.422-432.

Song, Z., Roussopoulos, N., 2001. K-nearest neighbor search for moving query point. Int. Conf. on Spatial and Temporal Databases, p.79-96.

Tao, Y., Papadias, D., 2002. Time parameterized queries in spatio-temporal databases. Int. Conf. on ACM Management of Data, p.334-345.

Tao, Y., Faloutsos, C., Papadias, D., *et al.*, 2004. Prediction and indexing of moving objects with unknown motion patterns. Int. Conf. on ACM Management of Data, p.611-622.

Wolfson, O., Yin, H., 2003. Accuracy and resource consumption in tracking and location prediction. *LNCS*, **2750**:325-343. [doi:10.1007/978-3-540-45072-6_19]

Wolfson, O., Sistla, A.P., Chamberlain, S., *et al.*, 1999. Updating and querying databases that track mobile units. *Distr. Parall. Databases*, **7**(3):257-387. [doi:10.1023/A:1008782710752]

Xiong, X., Mokbel, M.F., Aref, W.G., 2005. SEA-CNN: scalable processing of continuous K-nearest neighbor queries in spatio-temporal databases. Int. Conf. on Data Engineering, p.643-654.

Xiong, X., Mokbel, M.F., Aref, W.G., 2006. LUGrid: update-tolerant grid-based indexing for moving object. Int. Conf. on Mobile Data Management, p.13-20. [doi:10.1109/MDM.2006.102]

Yu, X., Pu, K.Q., Koudas, N., 2005. Monitoring K-nearest neighbor queries over moving objects. Int. Conf. on Data Engineering, p.631-642.