Closest Pair Queries in Spatio-Temporal Databases

Chin-Wan Chung Sunghee Choi Yong-Jin Choi

Division of Computer Science
Department of Electrical Engineering and Computer Science
Korea Advanced Institute of Science and Technology
373-1, Kusong-dong, Yusong-gu, Taejon 305-701, Republic of Korea
{chungcw,sunghee}@cs.kaist.ac.kr, omni@islab.kaist.ac.kr

Abstract

In recent years, spatio-temporal databases have been studied intensively. This paper proposes how to process k closest pair queries in spatio-temporal databases for the first time. A spatio-temporal k closest pair query continuously searches the k closest pairs between a set of spatial objects and a set of moving objects for a specified time interval of the query. To maintain the order of the k closest pairs, we use a time function that can represent the change in distance between a spatial object and a moving object as time passes.

For efficient processing of k closest pair queries, we present an event-based structure, instead of a simple split list structure to avoid unnecessary computations, along with a distance bound used to prune unnecessary node accesses. Our event-based method is 9 to 43 times faster, compared to a method using a simple split list structure. Also, our event-based structure can be applied to process spatio-temporal k nearest neighbor queries. In various experiments, our event-based approach is 11 to 46 times faster than an existing approach for processing spatio-temporal k nearest neighbor queries.

Key words: Spatio-Temporal Databases, Moving Object, Closest Pair Query

1 Introduction

In recent years, with the development of technologies such as wireless communication systems and global positioning systems (GPS), spatio-temporal databases have been studied intensively [13, 21, 11]. Research for moving objects can be classified into two groups according to the positions of the objects: the past positions and the future locations. Our paper is concerned with the future positions of moving objects. An example of a query on future locations is as follows: "which cars will be inside the query window 20 minutes from now?" Cars correspond to moving objects that move as time passes. To represent moving objects, we use a data model proposed by Sistla et al [14]. This model can manage the future positions of an object with the latest update information: the spatial position (x_1, x_2), the velocity (v_1, v_2), and the last update time (t_u) where subscripts 1 and 2 indicate each dimension in the 2-dimensional space. The future locations of a moving object can be represented as a function of time t: $(x_1 + (t - t_u) \times v_1, x_2 + (t - t_u) \times v_2)$. Various recent works [1, 9, 12, 13, 18, 20] have been developed on this model as well.

We apply this model to solve k closest pair queries (K-CPQs) and k nearest neighbor queries (K-NNQs) in spatio-temporal databases. A continuous nearest neighbor query [19] retrieves the nearest neighbor of every point on an arrow as shown in Figure 1 (e.g., find the nearest restaurant on my route from the position at time t^l to the position at time t^h). The result of the continuous nearest neighbor query is $\left\{f(t^l, t_1, o_1), (t_1, t_2, o_3), (t_2, t^h, o_5)\right\}$, meaning that the nearest neighbor for the time interval $\left[t^l, t_1\right]$ is o_1, the nearest neighbor for $[t_1, t_2]$ is o_3, and the nearest neighbor for $\left[t_2, t^h\right]$ is o_5. To process the continuous k nearest neighbor queries (K-NNQs), Tao et al. [19] presented a split list $SL = \{(t_1, t_2, \mathbb{K}_1), (t_2, t_3, \mathbb{K}_2), \ldots, (t_n, t_{n+1}, \mathbb{K}_n)\}$ where $\mathbb{K}_i$ denotes K-NN objects for $[t_i, t_{i+1}]$. They use a method that accesses all $\mathbb{K}_i$ for $1 \leq i \leq n$ in order to update SL where a spatio-temporal K-NNQ is processed. This processing method incurs a very long query processing time. The method is applied in situations that involve spatial target objects and a moving query object; however, it cannot be applied to cases involving only moving objects among which one is a query object or a spatial query object and moving target objects. To solve this problem, we use a time function (called *Curve*) that can represent the change in distance not only between a spatial object and a moving object but also between two moving objects.

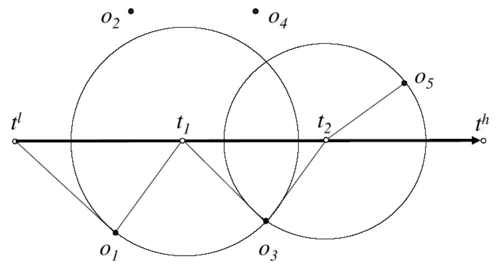


Fig. 1. continuous nearest neighbors

Spatiotemporal K-CPQs involve the problem of finding the k closest pairs (K-CPs) between spatial objects and moving objects as shown in Figure 2 for a given time interval. Let us consider a K-CPQ between cars and intersections of roads. Using the result of the K-CPQ, we can predict the amount of traffic congestion by checking what the density of cars near intersections will be.

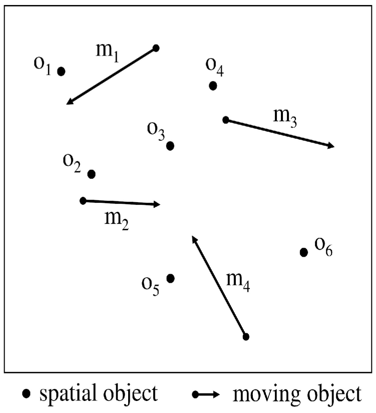


Fig. 2. spatial objects and moving objects

In this paper, we propose an event-based structure along with the concept of Curve. This event-based structure allows us to avoid unnecessary computations, and is paired with MinDistBound which is used for pruning unnecessary node visits, to efficiently process the k closest pair queries (K-CPQs).

In experiments on spatio-temporal K-CPQs, our event-based method is up to 43 times faster compared to a method using a simple split list structure. We performed spatio-temporal K-CPQs with various values of k. The results show that higher values of k lead to longer query processing time as in general experimental results [5]. We also applied our event-based structure to process k nearest neighbor searches on spatio-temporal databases proposed in [19]. In several such experiments of running spatio-temporal k nearest neighbor searches, our event-based method performed 11 to 46 times faster than the method of [19].

The contributions of this paper are as follows:

- We propose an event-based structure to avoid unnecessary computations in order to efficiently process K-NNQs or K-CPQs for various data sets in spatio-temporal databases.
- We develop an efficient algorithm for spatio-temporal K-CPQs, and especially derive a heap management algorithm that uses MinDistBound to minimize the total number of node accesses.

The rest of the paper is organized as follows. Section 2 discusses the related work on K-NNQs in spatio-temporal databases, K-CPQs in spatial databases, and the TPR-tree. In Section 3, we explain how to maintain the intermediate results of K-CPQs using an event-based structure. Section 4 describes a spatiotemporal K-CPQ algorithm that uses MinDistBound to prune unnecessary node visits. Section 5 presents experimental results and discusses them in detail. Finally, conclusions are made in Section 6.

2 Related Work

First, we briefly describe past work on K-NNQs in spatio-temporal databases. Then, we explain K-CPQs in spatial databases. And, we describe the TPR-tree for moving objects that correspond to a data set for processing spatio-temporal K-CPQs. Finally, we explain the k-th level problem.

Zheng and Lee [22] proposed a method for nearest neighbor queries for a moving object(or query) on spatial data. They pre-compute the Voronoi diagram of spatial objects and store it in the R-tree. The Voronoi diagram allows efficient processing of nearest neighbor queries. However, the method only deals with one nearest neighbor. Song and Roussopoulos [15] discussed nearest neighbor queries in R-trees that employ sampling. Their method has no accuracy guarantee since even a high sampling rate may miss some results.

Tao et al. [19] proposed a nearest neighbor query that retrieves the nearest neighbor of every point on a line segment. The method can retrieve all nearest neighbors for the query time interval and can return an accurate result. However, it is limited to performing the k nearest neighbor queries only between a moving query object and a set of spatial objects.

Hjaltason and Samet [8] proposed distance-join algorithms for closest pair queries in spatial databases. These algorithms are based on a priority queue that requires a large amount of main memory because they store not only the node pairs but also the object pairs. To solve this problem, Corral et al. [5] proposed a closest pair algorithm based on a heap structure that maintains only the internal node pairs. In [6], considering distance functions between two MBRs, they deduce lower and upper bounds for the k closest pairs of objects within two MBRs and use them for a pruning heuristic and updating strategies for their branch-and-bound algorithms to improve performance. They also applied their methods for extensions of K-CPQ such as K-Self-CPQ, Semi-CPQ, and K-FPQ(the k farthest pairs query). Our spatio-temporal K-CPQ algorithm is based on their work.

Saltenis et al. [13] proposed the TPR-tree that is based on the R-tree and supports spatio-temporal queries for the future locations of moving objects. Figure 3 shows a time-parameterized bounding interval(TPBI) of the TPR-tree and three one-dimensional moving objects($m_1 \sim m_3$) bounded by the bounding interval at t_0. TPBI is the most important concept of the TPR-tree and consists of a spatial interval $[y^l, y^h]$ and a velocity interval $[w^l, w^h]$ as shown in Figure 3. The spatial interval of TPBI is represented at index creation time t_0. The incline of an arrow indicates the moving objects velocity. As shown in Figure 3, the objects move within the range of bold lines as time passes.

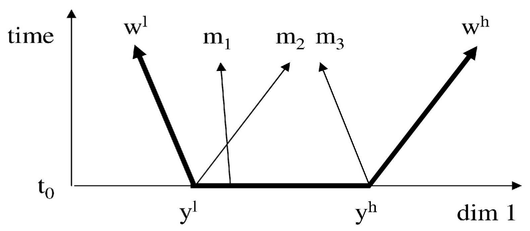


Fig. 3. Time-parameterized bounding interval of the TPR-tree at t_0

The k-th level problem in an arrangement of n curves is introduced in computational geometry. Tamaki and Tokuyama [17] proposed a kinetic data structure for the k-th level problem considering a small number of objects.We propose an event-based structure that can be maintained incrementally. Our method can be practically adapted in situations of a large number of objects, compared to the kinetic data structure. In addition, the kinetic data structure method was not implemented, and consequently no experimental result is available from the method. In this paper, we deal with curves of order from 1^{st} to k^{th}, whereas the k-th level problem focuses on the k^{th} order.

3 Event Maintenance Technique

In this section, we first define the problem of spatio-temporal K-CPQs formally and describe a method to calculate the change in distance between a spatial object and a moving object. The problem for determining the order of K-CPQs is conceptually the same as the k-th level problem [17]. Next, we introduce an event-based structure that can efficiently keep the ordered results with respect to the distance in spatio-temporal databases. Finally, we describe how to update the event-based structure.

3.1 Definition of Problem and Curve

Let $\mathbb{O} = \{o_1, o_2, \ldots, o_N\}$ be a set of spatial objects and $\mathbb{M} = \{m_1, m_2, \ldots, m_L\}$ be a set of moving objects. Let $[t^l, t^h]$ be the time interval of a query. The output of a K-CPQ from $\mathbb{O}$ and $\mathbb{M}$ for $[t^l, t^h]$ is $\{(t_1, t_2, \mathbb{K}_1), (t_2, t_3, \mathbb{K}_2), \ldots, (t_n, t_{n+1}, \mathbb{K}_n)\}$ where $t^l = t_1$, $t^h = t_{n+1}$, and for $1 \le i \le n$, each $[t_i, t_{i+1}]$ is the maximum interval in which the corresponding K-CPs $\mathbb{K}_i$ can be maintained. For $(t_g, t_{g+1}, \mathbb{K}_g)$, $\mathbb{K}_g = \{(o_{\alpha_1}, m_{\beta_1}), \ldots, (o_{\alpha_k}, m_{\beta_k})\}$ satisfies the following condition:

$$\begin{array}{l} dist(o_{\alpha_1}, m_{\beta_1}, t) \le \ldots \le dist(o_{\alpha_k}, m_{\beta_k}, t) \le dist(o_i, m_j, t), \\ o_{\alpha_1}, \ldots, o_{\alpha_k} \in \mathbb{O} \bigwedge m_{\beta_1}, \ldots, m_{\beta_k} \bigwedge \\ \forall (o_i, m_j) \in (\mathbb{O} \times \mathbb{M} - \{(o_{\alpha_1}, m_{\beta_1}), \ldots, (o_{\alpha_k}, m_{\beta_k})\}) \bigwedge \forall t \in [t_g, t_{g+1}]. \end{array}$$

Now, let us observe how to maintain the order of pairs in $\mathbb{K}_g$ for $[t_g, t_{g+1}]$. The distance d between a spatial object and a moving object can be represented as a function of time t, and d changes as time passes. Let (s_1, s_2) be the location of a spatial object and $(x_1 + (t - t_u) \times v1, x_2 + (t - t_u) \times v2)$ be the future locations of a moving object as mentioned in Section 1. Then, $d^2 = (x_1 + (t - t_u) \times v_1 - s_1)^2 + (x_2 + (t - t_u) \times v_2 - s_2)^2$. We can use this formula (called Curve) from which we can easily determine the distance ordering of closest pairs of spatial objects and moving objects. Figure 4 shows 6 curves for 6 pairs of spatial objects and moving objects. As seen in Figure 4, we can clearly determine the ascending order of curves in the time intervals. An intersection of two curves indicates the distances of two pairs are equal at the corresponding point in time. Figure 5 shows the result of a K-CPQ from 6 curves for $[t^l, t^h]$ where $k = 5$. To determine the order of two pairs, we use d^2 because it involves fewer and less costly computations than d.

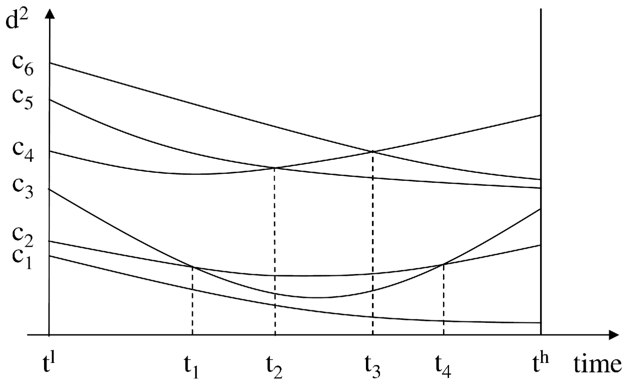


Fig. 4. 6 curves

time	order				
	1st	2nd	3rd	4th	5th
$t^l \sim t_1$	c_1	c_2	c_3	c_4	c_5
$t_1 \sim t_2$	c_1	c_3	c_2	c_4	c_5
$t_2 \sim t_3$	c_1	c_3	c_2	c_5	c_4
$t_3 \sim t_4$	c_1	c_3	c_2	c_5	c_6
$t_4 \sim t^h$	c_1	c_2	c_3	c_5	c_6

Fig. 5. Result of K-CPQ ($k = 5$)

3.2 Data Structure

We can acquire the exact result of a K-CPQ by checking all the pairs between two object sets. In order to find a precise final result, we can consider comparing and updating the intermediate result, which corresponds to a structure that enumerates all K-CPs, such as that of Figure 5, until all the pairs between the two sets have been checked. However, this method is very inefficient since it requires a large memory space and a great deal of CPU time. To solve this problem, we propose an efficient method that can find the K-CPs without wasting resources. Let us observe two K-CPs from two continuous time intervals in Figure 5. In general, the order of $k-2$ curves is the same while the order of the other 2 curves is different. For example, let us consider the K-CPs= $\{c_1, c_3, c_2, c_4, c_5\}$ for $[t_1, t_2]$ and the K-CPs= $\{c_1, c_3, c_2, c_5, c_4\}$ for $[t_2, t_3]$. The order of the 3 curves (c_1, c_2, c_3) is the same for $[t_1, t_3]$, however the order of the other 2 curves (c_4, c_5) for $[t_1, t_2]$ and $[t_2, t_3]$ are different. We call the change in the order of curves an *event* and use the event to efficiently process K-CPQs.

Table 1
Symbol description

Symbol	description
c	curve : $\alpha t^2 + \beta t + \gamma$
t^l, t^h	start time and end time of a query time interval
e	event $(t_{e1}, t_{e2}, c_d, c_u)$
$\mathbb{E}$	ordered set of events $\{e_1, e_2, \ldots, e_n\}$
b	boundary (t_{b1}, t_{b2}, c)
$\mathbb{B}$	ordered set of boundaries $\{b_1, b_2, \ldots, b_m\}$
I_c	curve index
$\mathbb{R}$	event-based structure $(I_c, t_1^l, t_1^h, \mathbb{E}, \mathbb{B}^k)$

Table 1 presents the symbols used throughout the paper. An event e consists of $(t_{e1}, t_{e2}, c_d, c_u)$, meaning that $d2(c_d, t_{e1} - \epsilon) > d2(c_u, t_{e1} - \epsilon)$, $d2(c_d, t_{e1}) = d2(c_u, t_{e1})$, $d2(c_d, t) < d2(c_u, t)$ for all $t \in (t_{e1}, t_{e2})$ where the function $d2(c, t)$ returns the value of c at time t. $\mathbb{E}$ and $\mathbb{B}$ are ordered by t_{e1} and t_{b1}, respectively. An ordered set $\mathbb{B}$ of boundaries is used to avoid unnecessary calculations. $\mathbb{B}^k$ identifies the ordered set of boundaries each of which is the k^{th} curve of the result of a K-CPQ. In order to access curves quickly, we employ a curve index I_c on a complete binary search tree for K-CPs and use curves for the key of the tree.

Let $t_1^l = t^l$ and $t_1^h = t_{e1}$ of the first event in $\mathbb{E}$. We propose an event-based structure $\mathbb{R} = (I_c, t_1^l, t_1^h, \mathbb{E}, \mathbb{B}^k)$ to efficiently update the intermediate results of a K-CPQ until we acquire the final result of the K-CPQ. The first component I_c of $\mathbb{R}$ is for K-CPs initially for $[t_1^l, t_1^h]$. We maintain $\mathbb{E}$ such that each event e in $\mathbb{E}$ includes at least one curve from K-CPs for $[e.t_{e1}, e.t_{e2}]$ where $e.\alpha$ denotes the α component of e. If the intersection of all curves of a certain event e and all curves of K-CPs for $[e.t_{e1}, e.t_{e2}]$ is empty, it means that the event is not related to the K-CPs and the event must be discarded. From Figure 4, we can acquire the following $\mathbb{R} = (I_c, t_1^l, t_1^h, \mathbb{E}, \mathbb{B}^5)$ when $k = 5$:

- I_c for 5 curves ordered as c_1, c_2, c_3, c_4, and c_5
- $t_1^l = t^l$, $t_1^h = t_1$
- $\mathbb{E} = \{(t_1, t_2, c_3, c_2), (t_2, t_3, c_5, c_4), (t_3, t_4, c_6, c_4), (t_4, t^h, c_2, c_3)\}$
- $\mathbb{B}^5 = \{(t^l, t_2, c_5), (t_2, t_3, c_4), (t_3, t^h, c_6)\}$

3.3 Maintaining Event-Based Structure

This section describes an efficient method for maintaining the event-based structure $\mathbb{R}$. First, we introduce a definition with respect to $\mathbb{B}$, which is required to avoid unnecessary computations. Let $\mathbb{B}_1$ and $\mathbb{B}_2$ be ordered boundary sets. We define a predicate $\rhd$ between $\mathbb{B}_1$ and $\mathbb{B}_2$ when $\mathbb{B}_1.b_1.t_{b1} = \mathbb{B}_2.b_1.t_{b1}$ and $\mathbb{B}_1.b_{|B_1|}.t_{b2} = \mathbb{B}_2.b_{|B_2|}.t_{b2}$. $\mathbb{B}_1 \rhd \mathbb{B}_2$ returns **TRUE** if all the curves of $\mathbb{B}_1$ are above all of the curves of $\mathbb{B}_2$, as shown in Figure 6. Otherwise, $\mathbb{B}_1 \rhd \mathbb{B}_2$ returns **FALSE**.

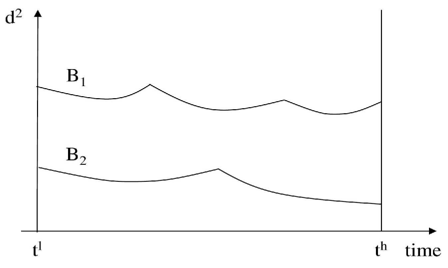


Fig. 6. Two ordered sets of boundaries

We briefly explain how to generate new events between curves of I_c and a newly considered curve. Consider a simple example as seen in Figure 7. Figure 7 shows 5 curves for $[t_i, t_{i+1})$ and the corresponding complete binary search tree I_c for the 5 curves ordered as c_2, c_1, c_3, c_4, and c_5. Then, consider a new curve c_6 that leads to the change of the order of the 5 curves as shown in Figure 8. Three events are generated from I_c and c_6. The shaded circles indicate the nodes that are visited during the processing for generating new events. Three events on I_c and c_6 for $[t_i, t_{i+1})$ is generated as the following : $\{(t_1, t_2, c_6, c_3), (t_2, t_3, c_6, c_1), (t_3, t_{i+1}, c_1, c_6)\}$.

We introduce how to update $\mathbb{R}$. The method is divided into the following two steps: the event generation step and the rearrangement step. The event generation step checks whether a new curve can be pruned by $\mathbb{B}^k$ of $\mathbb{R}$ and creates two curve indexes I'_c and I''_c from the previous I_c of $\mathbb{R}$ and the new curve if the curve is not pruned. The states of I'_c and I''_c validate for the time interval of the first event. We use I'_c for the rearrangement step and I''_c as I_c of the changed state of $\mathbb{R}$. The event generation step also adds new events generated by the new curve to $\mathbb{E}$ of $\mathbb{R}$. The rearrangement step makes new $\mathbb{B}^k$

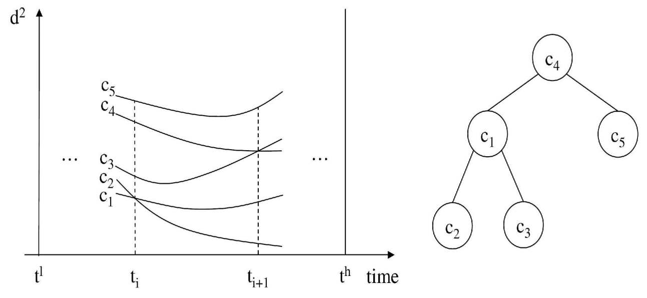


Fig. 7. curves and I_c

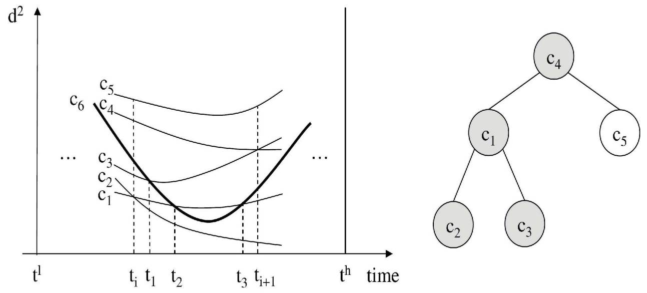


Fig. 8. Three new events generated from I_c and c_6 for $[t_i, t_{i+1})$

using I'_c and $\mathbb{E}$ obtained from the event generation step and removes invalid events from $\mathbb{E}$.

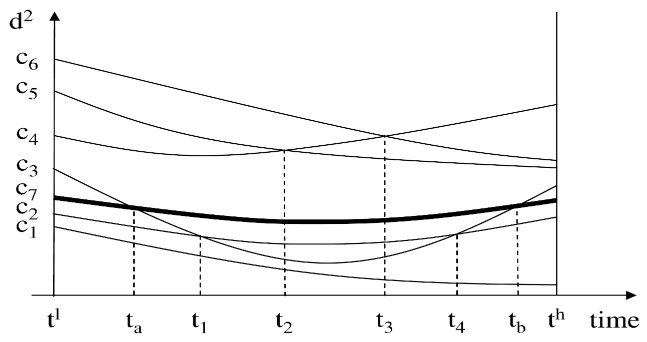
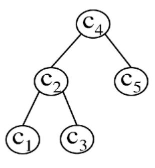
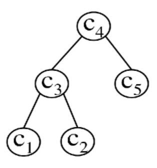
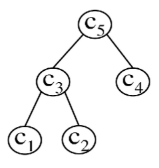
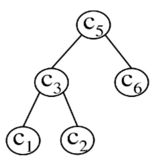
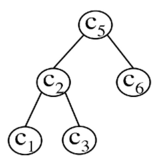


Fig. 9. Change of the order of K-CPs by c_7

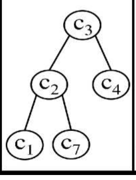
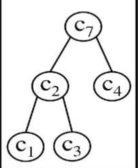
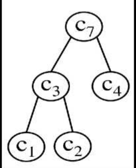
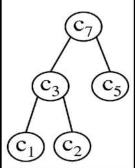
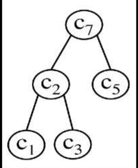
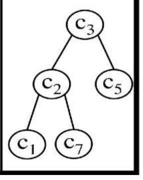
Next, we briefly present how to update $\mathbb{R}$ due to the effects of a new curve over a query time interval $[t^l, t^h]$. We assume that the current state r_1 of $\mathbb{R}$ is the state of Figure 4 and a new curve c_7 changes the state of $\mathbb{R}$ as shown in Figure 9. Let r_2 be the updated state of $\mathbb{R}$. Figure 10(a) illustrates the event generation step in detail. First, we check whether there exist new events between I_c and c_7 for $[t^l, t_1)$. An event (t_a, t_1, c_3, c_7) is created. Then, the processing is repeated as follows. We update the order of nodes of I_c affected by the first event (t_1, t_2, c_3, c_2) of $\mathbb{E}$. That is, the order of the 5 curves in I_c is changed from $\{c_1, c_2, c_3, c_4, c_5\}$ to $\{c_1, c_3, c_2, c_4, c_5\}$. Then, we check whether there exist any new events between the modified I_c and c_7 for $[t_1, t_2)$. For $[t_1, t_2)$, no event is created in our example. The same process is applied according to the order of the remaining events. As seen in Figure 10(a), a new event (t_b, t^h, c_7, c_3) is created for $[t_4, t^h]$. After the process of generating new events, the contents of $\mathbb{E}$ from its previous state and the two new events are merged to obtain the updated value for $\mathbb{E} = \{(t_a, t_1, c_3, c_7), (t_1, t_2, c_3, c_2), (t_2, t_3, c_5, c_4), (t_3, t_4, c_6, c_4), (t_4, t_b, c_2, c_3), (t_b, t^h, c_7, c_3)\}$.

Figure 10(b) illustrates the rearrangement step in detail. The ascending order of the 5 curves for the initial state of I'_c for $[t^l, t_a)$ is $\{c_1, c_2, c_7, c_3, c_4\}$. The order of the 5 curves of I'_c is modified by the first event (t_a, t_1, c_3, c_7) for $[t_a, t_1)$ so that the order becomes $\{c_1, c_2, c_3, c_7, c_4\}$. In a manner similar to the event generation step, we repeatedly process the rearrangement step. We construct $\mathbb{B}^k$ from the change of the k^{th} node of I'_c and remove invalid events in $\mathbb{E}$ if I'_c is not affected by the corresponding events. As seen in Figure 10(b), since the event (t_3, t_4, c_6, c_4) does not affect I'_c, with the order of curve set $\{c_1, c_3, c_2, c_7, c_5\}$ for $[t_2, t_3)$, the event is removed from $\mathbb{E}$. Consequently, we can acquire the following updated $\mathbb{R} = (I_c, t^l_1, t^h_1, \mathbb{E}, \mathbb{B}^5)$:

- I_c for 5 curves ordered as $\{c_1, c_2, c_7, c_3, c_4\}$
- $t^l_1 = t^l$, $t^h_1 = t_a$
- $\mathbb{E} = \{(t_a, t_1, c_3, c_7), (t_1, t_2, c_3, c_2), (t_2, t_4, c_5, c_4), (t_4, t_b, c_2, c_3), (t_b, t^h, c_7, c_3)\}$
- $\mathbb{B}^5 = \{(t^l, t_2, c_4), (t_2, t^h, c_5)\}$

time	$[t^l,t_1)$	$[t_1,t_2)$	$[t_2,t_3)$	$[t_3,t_4)$	$[t_4,t^h]$
event	-	(t_1,t_2,c_3,c_2)	(t_2,t_3,c_5,c_4)	(t_3,t_4,c_6,c_4)	(t_4,t^h,c_2,c_3)
state of I_c					
new event	(t_a,t_1,c_3,c_7)	-	-	-	(t_b,t^h,c_7,c_3)

(a) Event Generation Step

time	$[t^l,t_a)$	$[t_a,t_1)$	$[t_1,t_2)$	$[t_2,t_3)$	$[t_3,t_4)$	$[t_4,t_b)$	$[t_b,t^h]$
event	-	(t_a,t_1,c_3,c_7)	(t_1,t_2,c_3,c_2)	(t_2,t_3,c_5,c_4)	(t_3,t_4,c_6,c_4)	(t_4,t_b,c_2,c_3)	(t_b,t^h,c_7,c_3)
state of I_c'					-		

(b) Rearrangement Step

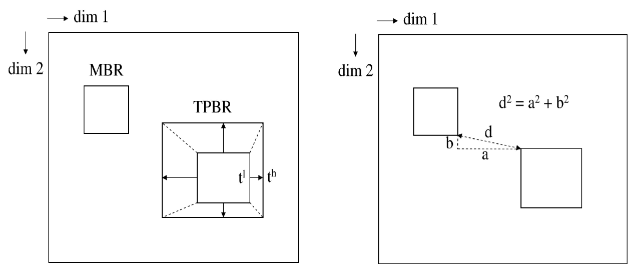
Fig. 10. Updating $\mathbb{R}$

4 K-CPQ Algorithm

In this section, we present an efficient algorithm for K-CPQs between a set of spatial objects and a set of moving objects. We assume that spatio-temporal K-CPQs are processed from the R*-tree for spatial objects and the TPR-tree for moving objects since the R*-tree is one of the most popular indexes in spatial databases [7, 3] and the TPR-tree is a popular index for spatiotemporal databases [13, 12]. We first discuss the minimum distance boundary called MinDistBound needed for pruning visits of unnecessary nodes. Then, we show the Heap algorithm using MinDistBound and the event-based structure mentioned in the previous section.

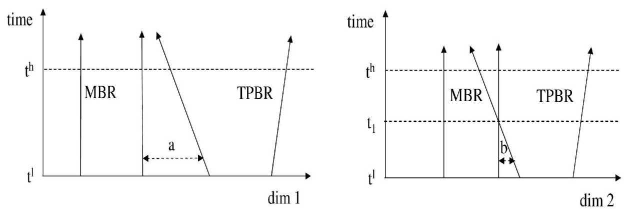
4.1 Minimum Distance Boundary

We use MinDistBound to decide which node pair to visit first in order to minimize the total number of accessed nodes. MinDistBound is an ordered set $\mathbb{B}$ of boundaries that indicates minimum distances between an MBR of an R-tree and a TPBR (called an MBR in a general tree structure) of a TPR-tree as time passes. Figure 11(a) illustrates an MBR of the R-tree and a TPBR of the TPR-tree. The inner rectangle of the TPBR is the MBR of the TPBR at time t^l. Similarly, the outer rectangle of the TPBR is the MBR of the TPBR at time t^h. As shown in Figure 11(b), let us consider the MBR of the TPBR at a certain time between t^l and t^h. Since the square of the minimum distance between two MBRs is $d^2 = a^2 + b^2$, we can consider the distances in two individual 1-dimensional spaces instead of a distance in 2-dimensional space for the MinDistBound between two MBRs. Figure 12 depicts the change in distance for each dimension as time passes. In the case of Figure 12(b), the distance is 0 after time t_1 since the MBR and the TPBR intersect with each other. Figure 13 illustrates a method that creates a MinDistBound between an MBR and a TPBR in the 2-dimensional space. As seen in Figure 13, the MinDistBound for a 2-dimensional space is the sum of the MinDistBound for each dimension.



(a) (b)

Fig. 11. MBR and TPBR



(a) (b)

Fig. 12. Distance for each dimension

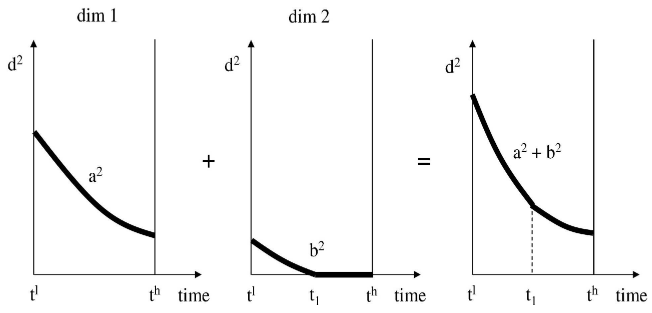


Fig. 13. Distance between an MBR and a TPBR

We use four functions for obtaining MinDistBound in the 2-dimensional space. Let MinDistBound1D be a function that creates a MinDistBound between an MBR and a TPBR in the corresponding 1-dimensional space. In the MinDistBound1D function, we use the method to find the intersecting time interval of two TPBRs to maintain the TPR-tree proposed by [13]. We can easily calculate MinDistBound for each dimension from the method proposed by [13]. The function CommonTimeInterval returns a common time interval from the time intervals of two boundaries. The FirstBoundary($\mathbb{B}$) returns the first boundary of $\mathbb{B}$. The NextBoundary($\mathbb{B}, b$) returns the next boundary of b in $\mathbb{B}$. For obtaining MinDistBound, we use the plane-sweep technique that is widely used in the area of computational geometry [4]. Figure 14 shows the MDB algorithm for generating MinDistBound between an MBR and a TPBR. As seen in Figure 14, this algorithm can be generalized into a D-dimensional space.

4.2 Heap Algorithm

We use a heuristic that aims at further improving our algorithm after accessing two internal nodes. For the child nodes of the two nodes, the heuristic is to sort the pairs of an MBR and a TPBR in ascending order of MinMinDist which indicates the minimum value of MinDistBound for $[t^l, t^h]$. This order of processing is expected to improve the pruning of pairs. A heap is used to store pairs of nodes according to their MinMinDist. The pair with the smallest value resides at the top of the heap. This pair is the next candidate for visiting. Figure 15 shows the overall algorithm. The Create function initializes a heap H and the Empty function checks whether H is empty. The first element of H

```
Algorithm MDB(MBR $R1$, TPBR $R2$, time $t^l$, time $t^h$)

1.  $\mathbb{B} \leftarrow MinDistBound1D(R1, R2, t^l, t^h, 1)$

2.  for $i \leftarrow 2$ to $D$ do
3.      $\mathbb{B}' \leftarrow MinDistBound1D(R1, R2, t^l, t^h, i)$
4.      $b \leftarrow FirstBoundary(\mathbb{B})$
5.      $b' \leftarrow FirstBoundary(\mathbb{B}')$
6.      $\mathbb{B}'' \leftarrow \{\}$

7.      while $b \neq null \bigwedge b' \neq null$ do
8.          $[t_1, t_2] \leftarrow CommonTimeInterval(b, b')$
9.          $c \leftarrow b.c + b'.c$
10.         $\mathbb{B}'' \leftarrow \mathbb{B}'' \bigcup \{(t_1, t_2, c)\}$
11.         if $b.t_{b2} < b'.t_{b2}$
12.             $b \leftarrow NextBoundary(\mathbb{B}, b)$
13.         else if $b.t_{b2} > b'.t_{b2}$
14.             $b' \leftarrow NextBoundary(\mathbb{B}', b')$
15.         else if $b.t_{b2} = b'.t_{b2}$
16.             $b \leftarrow NextBoundary(\mathbb{B}, b)$
17.             $b' \leftarrow NextBoundary(\mathbb{B}', b')$

18.     remove $\mathbb{B}$ and $\mathbb{B}'$
19.     $\mathbb{B} \leftarrow \mathbb{B}''$

20. return $\mathbb{B}$
```

Fig. 14. MDB algorithm

is returned by the Delete function and an entry pair is inserted by the Insert function. For pruning of unnecessary node visits, we use MaxMinDist that indicates the maximum value of MinDistBound for $[t^l, t^h]$.

```
Algorithm Heap(R-tree $R_1$, TPR-tree $R_2$, number $k$, time $t^l$, time $t^h$)

1.  $H \leftarrow Create()$
2.  $\mathbb{B}^k \leftarrow \{(t^l, t^h, \infty)\}$
3.  $Insert(H, R_1.RootNode, R_2.RootNode)$

4.  while not $Empty(H)$ do
5.      $(N_1, N_2) \leftarrow Delete(H)$
6.      $\mathbb{B} \leftarrow MDB(N_1.MBR, N_2.TPBR, t^l, t^h)$
7.      if MinMinDist of $\mathbb{B}$ > MaxMinDist of $\mathbb{B}^k$, then stop
8.      if $\mathbb{B} \rhd \mathbb{B}^k$ is FALSE
9.          if $N_1$ and $N_2$ are internal nodes
10.             calculate $\mathbb{B}$ for each possible entry pair
11.             for each pair $(E_1, E_2)$ where $\mathbb{B} \rhd \mathbb{B}^k$ is FALSE
12.                 $Insert(H, E_1.ChildNode, E_2.ChildNode)$
13.         if $N_1$ and $N_2$ are leaf nodes
14.             for each curve $c$ of all entry pairs
15.                 if $\{(t^l, t^h, c)\} \rhd \mathbb{B}^k$ is FALSE, update $\mathbb{R} = (I_c, t_1^l, t_1^h, \mathbb{E}, \mathbb{B}^k)$
```

Fig. 15. Heap Algorithm

5 Experimental Evaluation

In this section, we present the experimental environment and the experimental results of our proposed method. In order to create a realistic environment for our experiments, moving objects are synthetically generated using real-life spatial data. While the disk access is a popular performance metric, the CPU time can be significant for some operations. The total processing time includes both the disk access time and the CPU time. Therefore, we evaluate the total processing time need to process K-CPQs. Then, we show the experimental results for K-NNQs.

5.1 Experimental Environment

For a practical situation, let us consider K-CPQs between cars and intersections of roads within a part of a city. We generate moving objects in a way which makes an experimental environment more realistic. We use Sequoia data [16], which is real-life spatial data popularly used in spatial database research, to generate moving objects. We set the initial spatial positions of moving objects using Sequoia data. There are approximately 11000 moving objects. Objects are set to move randomly with the maximum speed 1 for one unit of time in the 10000 × 100002D space. To generate spatial objects that correspond to the other data of spatio-temporal K-CPQs, we use another set of real-life spatial data, Tiger/lines [10]. The number of spatial objects is about 3200. Experiments are conducted on a Pentium IV 2.4 GHz PC with 1 GBytes main memory. The size of the nodes for the R-tree and the TPR-tree is set to 4 KBytes, which corresponds to a page size. The value of k varies from 20 to 250. The length of the query time interval varies from 1 to 10 units.

Since our event-based structure can be applied in the processing of spatiotemporal K-NNQs, we additionally show the effect of the event-based structure on processing spatio-temporal K-NNQs. The length of the query varies among 1%, 2%, 4%, 8%, and 16% of the size of one axis in the 2D space and about 137000 spatial objects are considered. We compare our event-based method with an existing method [19] for processing spatio-temporal K-NNQs using the above experimental environment which is similar to that used in [19].

5.2 Experimental Results

For processing spatio-temporal K-CPQs, we can use a split list structure enumerating all K-CPs such as those of Figure 5 as well as our event-based structure. Figure 16 illustrates the performance of two methods: one method uses our event-based structure (EBS) and the other method uses the split list structure (SLS) proposed by [19]. The total cost is the number of seconds taken to process K-CPQs. The length of the query time interval is set to 1 unit. In the experiments for spatio-temporal K-CPQs where $20 \leq k \leq 80$, the method using SLS has the total cost ranging from 327.1 to 2221.4 seconds, while the method using EBS has the total cost ranging from 33.5 to 51.6 seconds. Since the method using SLS enumerating all K-CPs requires a large number of comparisons from the split list structure in order to maintain the order of the k closest pairs, it is very inefficient. However, our method using EBS can remarkably reduce the number of comparisons by using I_c and $\mathbb{B}^k$. Figure 17 shows the experimental results of our EBS for larger k's. The results using SLS for larger k's have been omitted as it requires a considerably large amount of time compared to our event-based structure.

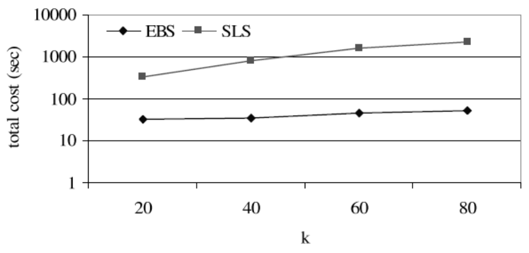


Fig. 16. Total cost of K-CPQ for $20 \leq k \leq 80$

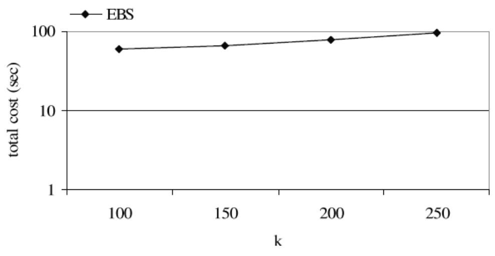


Fig. 17. Total cost of K-CPQ for $100 \leq k \leq 250$

We assess the total cost for various lengths of the query time interval in spatiotemporal K-CPQs. Figure 18 illustrates the experimental results with respect to the length of the query time interval from 1 to 5 units where k is set to 20. Our method using EBS is 9 to 17 times faster than the method using SLS. Figure 19 shows the total cost for the larger length of the query time interval.

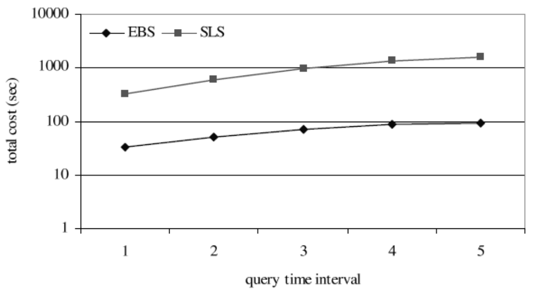


Fig. 18. Total cost of K-CPQ for query time interval from 1 to 5

Our event-based structure can be applied in the processing of spatio-temporal

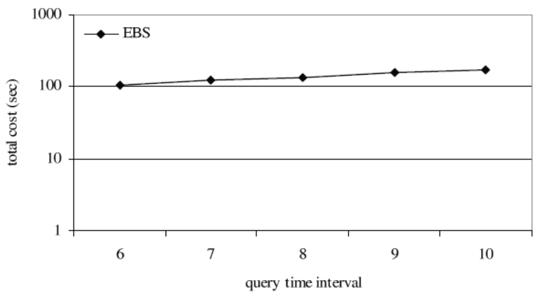


Fig. 19. Total cost of K-CPQ for query time interval from 6 to 10

K-NNQs. We compare our event-based method with the existing split list method proposed by Tao et al. [19] for processing spatio-temporal k nearest neighbor queries. Figure 20 illustrates the experimental results between our event-based method (EBM) and the existing split list method (SLM). The length of the query is set to 1% of the size of one axis. The results of 20 queries are averaged. In various experiments on K-NNQs, our event-based approach is 11 to 46 times faster than the existing approach.

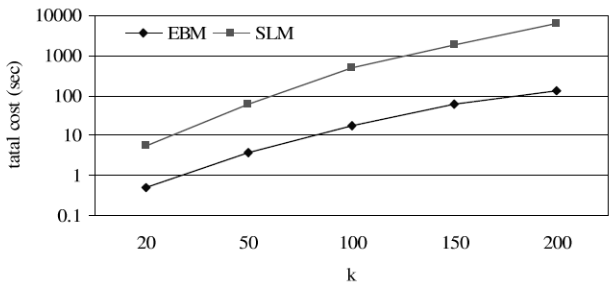
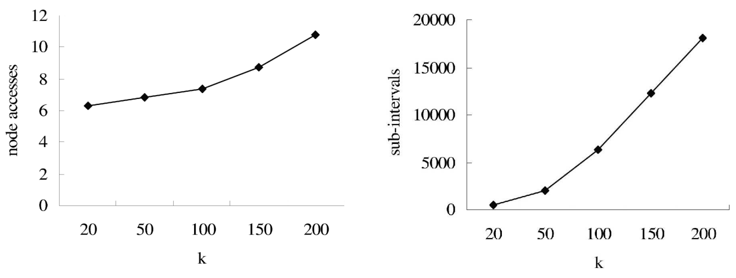


Fig. 20. Total cost of K-NNQ for $20 \leq k \leq 200$

Figure 21 shows the number of node accesses and the number of sub-intervals from the results of K-NNQs having varied k from 20 to 200. As k increases, the number of node accesses and the number of sub-intervals increase linearly with respect to k. As seen in Figure 20 and Figure 21, the processing of KNNQs is CPU bound since the number of node accesses gradually increases proportional to k whereas the total processing time remarkably decreases.



(a) Node accesses for k (b) Sub-intervals for k

Fig. 21. Node accesses and sub-intervals of K-NNQ for various k values

Next, we compare the performance of both methods by varying the length of the query from 1% to 16% where k is set to 20. When the length of the query is 4%, the total cost for EBM is 7.51 seconds, while that for SLM is 98.4 seconds.

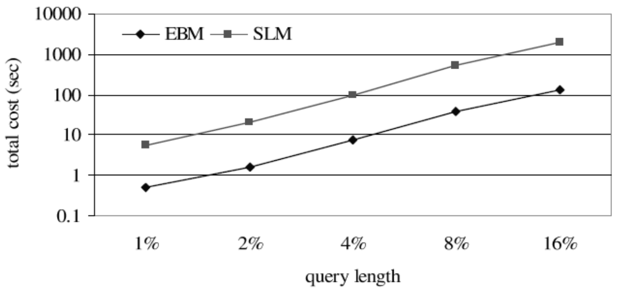


Fig. 22. Total cost of K-NNQ for query length

6 Conclusions

To maintain the order of the k closest pairs, we used a time function that can represent the change in distance between a spatial object and a moving object as time passes. The function can be also used for K-CPQs between a set of moving objects and a set of moving objects and for K-NNQs with the following various combinations of data: spatial objects and a moving query object, moving objects and a moving query object, and moving objects and a spatial query object. In this paper, we proposed an event-based structure, instead of a simple list data structure to avoid unnecessary computations, and a distance bound strategy used to prune unnecessary node accesses in order to efficiently process k closest pair queries. Our event-based method is 9 to 43 times faster, compared to a method using a simple split list structure, for spatio-temporal k closest pair queries. From various experimental results, we observed that the total cost for processing spatio-temporal K-CPQs is CPU bound rather than I/O bound.

As our future work for spatiotemporal K-CPQs, we will consider the problem in the environment of the road network. Many kinds of moving objects use the road network. The correct distance measure for those objects is not the Euclidean distance but the network distance.

References

[1] P. K. Agarwal, L. Arge, and J. Erickson. Indexing moving points. In *Proceedings of the ACM Symposium on Principles of Database Systems*, pages 175–186, 2000.

[2] P. K. Agarwal, B. Aronov, and M. Sharir. On levels in arrangements of lines, segments, planes, and triangles. In *Proceedings of the ACM Symposium on Computational Geometry*, pages 30–38, 1997.

[3] N. Beckmann, H. P. Kriegel, R. Schneider, and B. Seeger. The r*-tree: An efficient and robust access method for points and rectangle. In *Proceedings of the ACM SIGMOD Conference on Management of Data*, pages 323–331, 1990.

[4] T. Brinkho, H. P. Kriegel, and B. Seeger. Effcient processing of spatial joins using r-trees. In *Proceedings of the ACM SIGMOD Conference on Management of Data*, pages 237–246, 1993.

[5] A. Corral, Y. Manolopoulos, Y. Theodoridis, and M. Vassilakopoulos. Closest pair queries in spatial databases. In *Proceedings of the ACM SIGMOD Conference on Management of Data*, pages 189–200, 2000.

[6] A. Corral, Y. Manolopoulos, Y. Theodoridis, and M. Vassilakopoulos. Algorithms for processing K-closest-pair queries in spatial databases. In *Data and Knowledge Engineering* 49, pages 67–104, 2004.

[7] A. Guttman. R-tree: A dynamic index structure for spatial searching. In *Proceedings of the ACM SIGMOD Conference on Management of Data*, pages 47–57, 1984.

[8] G. R. Hjaltason and H. Samet. Incremental distance join algorithms for spatial databases. In *Proceedings of the ACM SIGMOD Conference on Management of Data*, pages 237–248, 1998.

[9] G. Kollios, D. Gunopulos, and V. J. Tsotras. On indexing mobile objects. In *Proceedings of the ACM Symposium on Principles of Database Systems*, pages 261–272, 1999.

[10] U. S. B. of Census. Tiger/lines precensus files: 1994 technical documentation, technical report, 1994.

[11] D. Papadias, J. Zhang, N. Mamoulis, and Y. Tao. Query processing in spatial network databases. In *Proceedings of the International Conference on Very Large Data Bases*, pages 802–813, 2003.

[12] S. Saltenis and C. S. Jensen. Indexing of moving objects for locationbased services. In *Proceedings of the IEEE International Conference on Data Engineering*, pages 166–175, 2002.

[13] S. Saltenis, C. S. Jensen, S. T. Leutenegger, and M. A. Lopez. Indexing the positions of continuously moving objects. In *Proceedings of the ACM SIGMOD Conference on Management of Data*, pages 331–342, 2000.

[14] A. P. Sistla, O. Wolfson, S. Chamberlain, and S. Dao. Modeling and querying moving objects. In *Proceedings of the IEEE International Conference on Data Engineering*, pages 422–432, 1997.

[15] Z. Song and N. Roussopoulos. K-nearest neighbor search for moving query point. In *Proceedings of the Symposium on Spatial and Temporal Databases*, pages 79–96, 2001.

[16] M. Stonebraker, J. Frew, K. Gardels, and J. Meredith. The sequoia 2000 storage benchmark. In *Proceedings of the ACM SIGMOD Conference on Management of Data*, pages 2–11, 1993.

[17] H. Tamaki and T. Tokuyama. How to cut pseudo-parabolas into segments. In *Proceedings of the ACM Symposium on Computational Geometry*, pages 230–237, 1995.

[18] Y. Tao and D. Papadias. Time-parameterized queries in spatio-temporal databases. In *Proceedings of the ACM SIGMOD Conference on Management of Data*, pages 334–345, 2002.

[19] Y. Tao, D. Papadias, and Q. Shen. Continuous nearest neighbor search. In *Proceedings of the International Conference on Very Large Data Bases*, pages 287–298, 2002.

[20] O. Wolfson, S. Chamberlain, S. Dao, L. Jiang, and G. Mendez. Cost and imprecision in modeling the position of moving objects. In *Proceedings of the IEEE International Conference on Data Engineering*, pages 588–596, 1998.

[21] J. Zhang, M. Zhu, D. Papadias, Y. Tao, and D. L. Lee. Location-based spatial queries. In *Proceedings of the ACM SIGMOD Conference on Management of Data*, pages 443–454, 2003.

[22] B. Zheng and D. Lee. Semantic caching in location-dependent query processing. In *Proceedings of the Symposium on Spatial and Temporal Databases*, pages 97–116, 2001.