Effectively Indexing Uncertain Moving Objects for Predictive Queries

Meihui Zhang[1] Su Chen[1] Christian S. Jensen[2] Beng Chin Ooi[1] Zhenjie Zhang[1]

[1]School of Computing
National University of Singapore
{zmeihui,chensu,ooibc,zhenjie}@comp.nus.edu.sg

[2]Department of Computer Science
Aalborg University
csj@cs.aau.dk

ABSTRACT

Moving object indexing and query processing is a well studied research topic, with applications in areas such as intelligent transport systems and location-based services. While much existing work explicitly or implicitly assumes a deterministic object movement model, real-world objects often move in more complex and stochastic ways. This paper investigates the possibility of a marriage between moving-object indexing and probabilistic object modeling. Given the distributions of the current locations and velocities of moving objects, we devise an efficient inference method for the prediction of future locations. We demonstrate that such prediction can be seamlessly integrated into existing index structures designed for moving objects, thus improving the meaningfulness of range and nearest neighbor query results in highly dynamic and uncertain environments. The paper reports on extensive experiments on the B^x-tree that offer insights into the properties of the paper's proposal.

1. INTRODUCTION

With the proliferation of location tracking and wireless communication, the management of moving object database has attracted considerable attention in the database research community over the last decade. The state-of-the-art in indexing and query processing for moving objects has reached a level where technologies have enabled moving object data management capable of supporting a wide spectrum of applications, e.g., in areas such as intelligent transport system and location-based services.

However, most existing work on moving object data management explicitly or implicitly assumes deterministic movement prediction models that require moving objects to always report accurate locations and velocities to the system. The system then predicts the location of an object until its next update according to some pre-defined class of functions. Assuming that an object reports its latest location $(x(t_0), y(t_0))$ and velocity $(v_x(t_0), v_y(t_0))$ at time t_0, Equation 1 illustrates a typical linear prediction of the object's location for time $t > t_0$.

$$\begin{pmatrix} x(t) \\ y(t) \end{pmatrix} = \begin{pmatrix} x(t_0) \\ y(t_0) \end{pmatrix} + \begin{pmatrix} v_x(t_0) \\ v_y(t_0) \end{pmatrix} \cdot (t - t_0) \qquad (1)$$

Index structures utilize simple predictions such as this to facilitate the efficient processing of timeslice queries as of the current and near-future times. A key basic assumption underlying this scheme is that the location and velocity of an object are deterministically reported to the server.

Unfortunately, the expectation of high accuracy on location and velocity measurement is unrealistic in many real-world applications. Due to the limited accuracy of available positioning systems, objects may only be able to report approximate locations or distributions on their possible locations. Further, real-world objects such as taxis and private cars often move in complex and stochastic ways. For example, Figure 1 plots sampled velocities of a school bus in the Athens metropolitan area during one hour[1]. From the

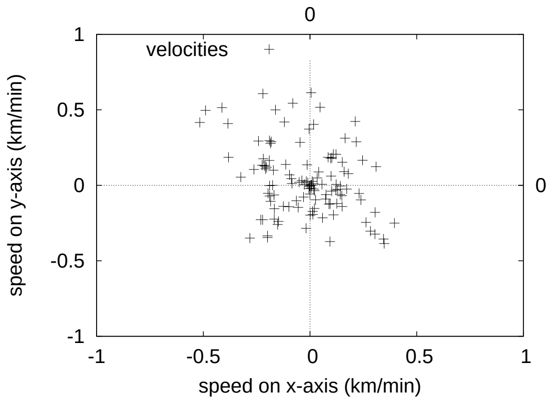


Figure 1: Velocity Distribution of a Bus During One Hour

figure, it can be concluded that the bus changes its velocity frequently. To maintain a valid prediction function in the moving object database, the bus is then forced to report its up-to-date position and velocity frequently. These updates incur high communication and computation costs.

The root of the problem is the asymmetric information gap between the object's real movement and the abstraction used to model its movement in the database. Specifically, current deterministic movement models are unable to capture the necessary information on the uncertainty of the moving objects. By introducing uncertainty models into moving object database management, it is possible to improve the robustness of query results computed over the predictions of the locations of highly dynamic moving objects, thus rectifying this problem.

Despite extensive studies in probabilistic databases [1, 4, 6] and proposals for extensions to moving-object querying [11, 21], it re-

[1]http://www.rtreeportal.org/

mains unclear how these models can be successfully integrated into existing moving object database. In particular, existing solutions exhibit three drawbacks. First, a good uncertain moving-object model should provide reasonable prediction capabilities with respect to the future motion of objects; in contrast, existing proposals offer limited capabilities in this regard, as only past motion is indexed while taking uncertainty into account [21]. Second, an uncertain moving object model is expected to be general enough to be applicable to all objects. One approach utilizes frequent behaviors of the moving object, yielding predictions that are not valid over objects with unusual movement patterns [11]. Third, to the best of our knowledge, no uncertain moving object models so far can be supported seamlessly by the existing infrastructure of database management system, rendering integration into real systems very costly.

We present a new framework for query processing over uncertain moving objects. This framework offers general prediction functionality and ease of integration into current systems. We utilize a generic movement inference model that infers the location distribution at a specified time according to the current location and velocity distributions. We show that state-of-art moving object index structures, such as B^x-tree, can be adapted to index these distributions and answer probabilistic range and k-nearest neighbor queries in an efficient manner.

The contributions of the paper are summarized as follows.

1. We present a new uncertain moving object model that takes into account the uncertainties on both location and velocity.
2. We re-formulate traditional queries to apply to uncertain moving objects.
3. We devise a new movement inference model based on the new uncertain moving object model.
4. We show how the uncertain moving object model can be incorporated into existing index structures.
5. We analyze the performance of the proposed methods with extensive experimental studies.

The remainder of the paper is organized as follows. Section 2 reviews related work and Section 3 captures the setting and problem addressed in the paper. Section 4 covers methods for movement inference. Then Section 5 integrates support for uncertain moving objects into a moving-object index structure, and Section 6 covers query processing using the index structure. Section 7 evaluates the performance of the paper's proposals, and Section 8 offers conclusions.

2. RELATED WORK

We review in turn related studies on probabilistic models and indexes for moving objects.

2.1 Models of Uncertain Moving Objects

It is instructive to classify existing uncertain moving-object models according to two categories.

The first category contains models on uncertain trajectories of moving objects. The concept of uncertain trajectory was recently studied in detail by Trajcevski et al. [21]. Considering the measurement errors when capturing object movement, all object trajectories are expanded by some predefined parameter ϵ. Queries are thus issued on the expanded trajectories, with the support of effective indexing techniques. A different model was proposed by Cheng et al. [6], in which the location uncertainties are updated at every time point. Range queries and probabilistic nearest neighbor queries are issued at the current time point also.

The second category applies uncertain prediction models to the accurate location and velocity reported by moving objects. For example, Jeung et al. [11] employ a prediction model that returns possible locations of moving objects with varying probabilities.

The existing works are unable to simultaneously address the uncertainty of location and velocity of moving objects. Therefore, their inference models rely on the accuracy of the measurements, and they suffer from costly communication and updates when the motions of the objects are stochastic in nature.

Range and nearest neighbor queries are prominent for static uncertain objects in probabilistic databases [1, 4, 5, 14]. In a moving-object database, the uncertainty of an object changes with time. A predominant approach is to model an uncertain moving object as a static object at each time point, thus reusing existing query processing algorithms. Cheng et al. [5] use pruning strategies on probabilistic records when computing probabilistic range and nearest neighbor queries. Kriegel et al. [14] apply Monte-Carlo sampling on a probabilistic database to retrieve objects with high likelihood of being the nearest neighbors of query points. Cheng et al. [4] propose verification methods for nearest neighbor probabilities, by partitioning the distances into subregions and deriving lower and upper bound probabilities for the subregions. This type of method was subsequently improved by refining the partitions depending on estimates of the costs and benefits of sub-regions [1]. However, by converting moving objects to static objects at each time point, a query must be re-evaluated at every time point, resulting in high query costs.

2.2 Index Structures for Moving Objects

Given up-to-date locations and velocities of certain moving objects, traditional moving object indexing structures typically target range and k-nearest neighbor queries at specified time points.

Most existing solutions for moving object indexing assume a linear movement model. They can be further divided into object partitioning and space partitioning solutions. The former utilize multi-dimensional index structures, typically the R-tree and its variants. The TPR-tree [17] is arguably the earliest attempt in this direction. Objects are inserted into the index using a time-parameterized extensions of the traditional R-tree insertion strategy. A later variant, the TPR*-tree [20], is similar to the TPR-tree, but, notably, uses a different insertion algorithm that puts additional efforts into attempting to insert objects at better locations in the index.

To index moving objects with space partitioning, the typical approach is to partition the entire space into cells. By mapping the 2-dimensional cells into a 1-dimensional sequence by means of a space filling curve, the B$^+$-tree can be employed to index the locations of the moving objects at reference time(s) [3, 8, 9, 15]. Given a range query, this approach first transforms the query range into cell sequences using the space filling curve. These sequences are then issued as queries to the B$^+$-tree structure, with query expansion to account for the linear movement of the objects indexed. For a k nearest neighbor query, the answer is retrieved by gradually increasing the radius of a range query until k results are found. Section 5.1 provides additional detail on a typical space partitioning index, the B^x-tree.

Recent benchmark studies [2, 10] compare the performance of state-of-art moving objects indexes. Techniques that build on the B$^+$-tree are capable of very good performance, and they offer ease of integration into real systems that do not support multidimensional structures such as the R-tree. Further, effective locking strategies are available for the B$^+$-tree, which is important in concurrent environments [18]. Although similar strategies exist for the R-tree and its variants [12, 13] complex index operations lock nodes for longer time, which adversely affects throughput [7, 8].

While all of the works above employ the linear movement model, more complex movement models have also been studied. Tao et al. [19] utilize high-order movement formulas to model the future location of the moving objects. This model enables the system to predict the motion more accurately, which can be fully supported by a modified TPR-tree. Jeung et al. [11] presented another prediction model that relies purely on frequent patterns discovered from previous trajectories. Their approach finds frequent movement patterns in a collection of object movements and uses these for prediction.

3. UNCERTAIN MOVING OBJECT MODEL

We assume that we are given a population of n 2D moving objects, $D = \{o_1, o_2, \ldots, o_n\}$. Following previous studies on moving object indexing, the time dimension is modeled as a set of discrete time points, $T = \{1, 2, \ldots, t, \ldots\}$. Given an object o_i, we use loc_i^t and vel_i^t to denote the exact location and velocity of o_i at time t.

We enhance existing certain moving object representations to capture uncertain information by recording distributions on location and velocity instead of exact values. Specifically, we use $\mathcal{L}_i^t$ and $\mathcal{V}_i^t$ to denote the distributions of location and velocity for object o_i at time t. Existing certain moving object representations are special cases of this general representation where the distributions degenerate to single values in the location and velocity spaces.

With this extension, the moving object database thus stores the distribution information of each uncertain moving object. In particular, each moving object o_i is associated with a tuple $\left(\mathcal{L}_i^{t_r}, \mathcal{V}_i^{t_r}, t_u\right)$, where t_u is the update time for the object and $\mathcal{L}_i^{t_r}$ and $\mathcal{V}_i^{t_r}$ are the location and velocity distributions at t_u. An update of a moving object replaces the location and velocity distribution as well as the update time currently recorded for the object.

By using the distribution information at the update time, the uncertain moving object database aims to answer queries at any time t. This requires a movement inference model for predicting the location distribution of an object. Such a model has the following signature:

$$F(\mathcal{L}_i^{t_u}, \mathcal{V}_i^{t_u}, t_u, t) \; : \; \mathcal{S}_\mathcal{L} \times \mathcal{S}_\mathcal{V} \times T \times T \mapsto \mathcal{S}_\mathcal{L} \qquad (2)$$

In other words, given the location and velocity distributions of o_i at update time t_u, as well as the query time t, the function derives a new location distribution for object o_i at non-past time t.

Using the inference model, we can reformulate the traditional range queries and k nearest neighbor queries to apply to a database of uncertain moving objects. The definitions next follow the concepts adopted in current probabilistic database research [1, 4].

DEFINITION 3.1. ***Probabilistic Range Query***
Given a spatial range R, a query time t, and a threshold θ, the probabilistic range query returns all uncertain moving objects falling into R with probability no smaller than θ at time t, i.e., $\{o_i \in D \mid \Pr(loc_i^t \in R) \geq \theta\}$.

DEFINITION 3.2. ***Top-k Probabilistic NN Query (k-PNN)***
Given a query location q and a query time t, the probabilistic nearest neighbor query returns k uncertain moving objects with the highest probabilities of being the nearest neighbor of q.

The number of uncertain moving objects returned by the k-PNN query can be less than k, if not enough objects have non-zero probability of being a nearest neighbor of q. This happens when an object's maximal distance to q is less than the minimal distances between q and all other objects.

While the notions defined above do not rely on how we represent the distributions, the problem of distribution representation becomes important when we are to manage these distributions in a real database management system. Unfortunately, not all distributions are amenable to the resulting storage and computational requirements. To balance the cost of inaccuracy and the ease of representation, we adopt a representation scheme that discretizes the space and velocity domains by means of regular grids. If B bits are specified in vector quantization, each dimension is partitioned into 2^B intervals of equal length. A grid cell can thus be represented by a vector of $2B$ bits. Probabilities are assigned to the cells, assuming a uniformly distribution in each cell. Thus, the distribution of the location and velocity is approximated by a sequence of cells with non-zero probabilities.

To further simplify the notation, we number the cells in according to a space filling curve, e.g., the Z-curve or the Hilbert curve. Without loss of generality, we employ the Hilbert curve in the rest of the paper. Based on the numbering of the cells, $C_j^\mathcal{L}$ denotes the cell with (Hilbert) number j in the space domain. Correspondingly, $C_j^\mathcal{V}$ denotes the counterpart in the velocity domain. The probability of an object in $C_j^\mathcal{L}$ ($C_j^\mathcal{V}$) is $P_i(C_j^\mathcal{L})$ ($P_i(C_j^\mathcal{V})$).

Figure 2, exemplifies an uncertain moving object o_i at time t. Its location and velocity distributions are summarized in Figures 2(a) and 2(b). As o_i has positive probabilities in 4 cells in the space domain, the location distribution can be written as $\mathcal{C}_{o_i}^\mathcal{L} = \{(C_2^\mathcal{L}, 0.2), (C_3^\mathcal{L}, 0.5), (C_4^\mathcal{L}, 0.1), (C_8^\mathcal{L}, 0.2)\}$, indicating the probabilities of the objects in the cells. Similarly, the velocity distribution is captured as $\mathcal{C}_{o_i}^\mathcal{V} = \{(C_9^\mathcal{V}, 0.7), (C_{12}^\mathcal{V}, 0.3)\}$.

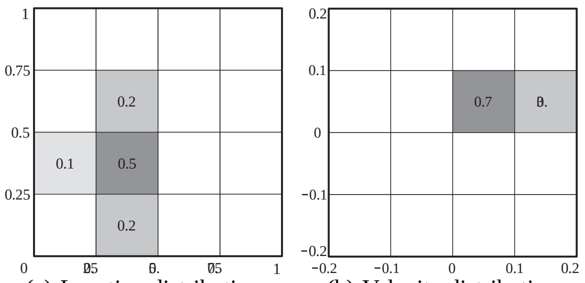


Figure 2: Example of an Uncertain Moving Object

Using the location and velocity cell sets as the underlying distribution knowledge, we can apply this knowledge to sets of objects. Given a set of moving objects $S = \{o_{s_1}, o_{s_2}, \ldots, o_{s_r}\} \subseteq D$, the location cell set $\mathcal{C}_S^\mathcal{L}$ contains all cells in the space domain with positive probability for at least one object $o_{s_i} \in S$, i.e., $\mathcal{C}_S^\mathcal{L} = \{C_j^\mathcal{L} \mid \exists i \; (P_{s_i}(C_j^\mathcal{L}) > 0)\}$. Similarly, the possible velocity cells are represented by $\mathcal{C}_S^\mathcal{V}$.

Table 1 explains the above use of P_{s_i} and summarizes the notation used in the remainder of the paper.

Conventional data management techniques fall short in accommodating this model of uncertain moving objects. In the following three sections, we introduce core components of our query processing framework. First, movement inference provides a method to derive the location distribution of an object at any near-future time from the location and velocity distributions known currently. Second, to enable efficient query processing and updates of location and velocity distributions, we re-use the B^x-tree, which was intro-

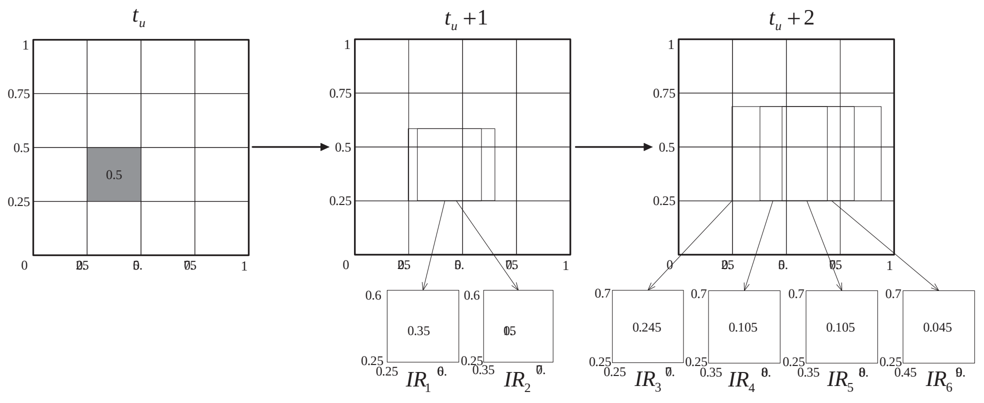


Figure 3: Recursive Movement Inference

duced for indexing certain moving objects. The insertion and deletion algorithms of the B^x-tree are amended for supporting moving objects with uncertainty. Third, based on the revised index structure, the algorithms for processing possibility range queries and k-PNN queries are presented.

Notation	Explanation
D	uncertain moving object set
n	the cardinality of object set D
o_i	one uncertain moving object in D
t_u	update time of some moving object
t_r	reference time of the index structure
$\mathcal{L}_i^{t_r}$	location distribution of o_i at time t_r
$\mathcal{V}_i^{t_r}$	velocity distribution of o_i at time t_r
$\mathcal{S}_{\mathcal{L}}$	location distribution space
$\mathcal{S}_{\mathcal{V}}$	velocity distribution space
R	query range in the spatial space
θ	probability threshold of range query
q	query point for k-PNN query
k	number of objects returned by k-PNN query
$P_i(C_j^{\mathcal{L}})$	the probability of o_i in location cell $C_j^{\mathcal{L}}$
$P_i(C_j^{\mathcal{V}})$	the probability of o_i in velocity cell $C_j^{\mathcal{V}}$
S	a subset of objects in D
o_{s_i}	ith object in S
$\mathcal{C}_{o_i}^{\mathcal{L}}$	spatial cells with positive probability of o_i
$\mathcal{C}_{o_i}^{\mathcal{V}}$	velocity cells with positive probability of o_i
$\mathcal{C}_S^{\mathcal{L}}$	spatial cells with positive probability of any object in S
$\mathcal{C}_S^{\mathcal{V}}$	velocity cells with positive probability of any object in S

Table 1: Notation

4. MOVEMENT INFERENCE

A key component of the paper's proposals is a method for inferring the location distribution of an uncertain moving object based on the most recently reported location and velocity distributions. In this section, we derive two methods, called *Rectangle Inference* and *Monte-Carlo Simulation*, that combine to provide an appropriate inference method. Given a range query, the former estimates an upper bound on the probability of an object being in the answer, while the latter is an approximate solution used for the final verification in our implementation. The extension of these methods to supporting k-PNN queries is covered in later sections.

4.1 Rectangle Inference

The rectangle inference method is motivated by the following observations. Consider the cell covering the rectangular region $C_3^{\mathcal{L}} = [0.25, 0.5] \times [0.25, 0.5]$ in Figure 2(a), in which the object o_i appears with probability 0.5 at update time t_u. The inference model estimates the distribution of this location information for the object given that the object's velocity follows the distribution in Figure 2(b). Assuming the query time t is later than the update time t_u, we present a simple example on the inference of the location distribution in Figure 3.

Between times t_u and $t_u + 1$, the movement of object o_i follows the velocity distribution by assumption. With probability 0.7, the velocity is in the velocity cell $C_9^{\mathcal{V}}$. If this the case, the movement of o_i along both the x-axis and the y-axis is in the interval $[0, 0.1]$. This implies that the location of o_i is in the rectangle with diagonal corners at $(0.25, 0.25)$ and $(0.6, 0.6)$, with probability $0.5 \times 0.7 = 0.35$. Similarly, if the velocity of o_i falls in the cell $C_{12}^{\mathcal{L}} = [0.1, 0.2] \times [0, 0.1]$, the location of o_i at the next time point is in the rectangle $[0.35, 0.7] \times [0.25, 0.6]$ with probability $0.5 \times 0.3 = 0.15$.

Figure 3 shows the two rectangles at time t_u+1, marked IR_1 and IR_2. Note that the inferred rectangles overlap in the space domain. To extend the analysis to subsequent time points, it is necessary to derive new inferred rectangles from IR_1 and IR_2 separately, taking into account the velocity distribution. This leads to four inferred rectangles at time $t_u + 2$, namely $\{IR_3, IR_4, IR_5, IR_6\}$, as is shown to the right in the figure.

We formalize rectangle inference as follows. If knowing that object o_i belongs to an *inferred rectangle* $IR = [IR.l[1], IR.u[1]] \times [IR.l[2], IR.u[2]]$ in the space domain with probability $IR.p$ at time t, the following lemma gives a recursive derivation of the inferred rectangles at time $t + 1$.

LEMMA 4.1. *Given an inferred range IR at time t of object o_i and a velocity cell $C_j^{\mathcal{V}} \in \mathcal{C}_i^{\mathcal{V}}$ with positive probability, o_i belongs to another inferred rectangular region IR' with probability $p_i(C_j^{\mathcal{V}}) \cdot IR.p$ at time $t + 1$, with diagonal corners at $(IR.l[1] + C_j^{\mathcal{V}}.l[1], IR.l[2] + C_j^{\mathcal{V}}.l[2])$ and $(IR.u[1] + C_j^{\mathcal{V}}.u[1], IR.u[2] + C_j^{\mathcal{V}}.u[2])$.*

While the distribution at update time is supposed to be uniform in the original spatial cells, uniformity is no longer guaranteed in the inferred cells. To understand why, consider the example in Figure 4 for an object moving in 1D. Assume some object is uniform in the space interval $[0, 0.5]$ and the velocity interval $[0, 0.1]$. The spatial distribution of the object at the next time point follows the grey density function plotted in the figure, which is uniform only between 0.1 and 0.5. This phenomenon is due to the boundary effect of the uniform distributions at the previous time point. Thus, the inference on the basis of the lemma does not yield a precise lo-

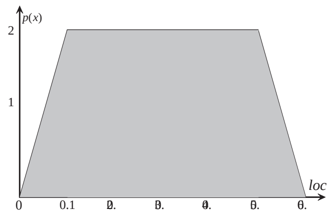


Figure 4: Spatial Distribution After Inference

cation distribution, but only provides an upper bound on the probability of the moving object falling into a region R. In other words, if the inferred rectangle overlaps with the query region, the probability of the object appearing in the query region is no smaller than the probability on the inferred rectangle.

While the inference method thus introduces distribution information loss, the good news is that Lemma 4.1 does not rely on the condition of uniform distributions in the space or velocity cells. This allows us to recursively apply the lemma to construct inferred rectangles from a reference (update) time t_u to any time $t > t_u$.

Algorithm 1 present the details on how inferred rectangles are used to evaluate the upper bound of an object with respect to a range query. The algorithm generates inferred rectangles with the elapse of time. For each rectangle IR' generated, line 8 determines whether it is able to infer any rectangle that overlaps with the query range R at time t, by expanding the region of IR' with the maximum and minimum speeds of the uncertain moving object in both dimensions. After all inferred rectangles are generated at time t,

Algorithm 1 IR-based Query Verification
$(\mathcal{C}_i^{\mathcal{L}} = \{(C_j^{\mathcal{L}}, P_i(C_j^{\mathcal{L}})\}, \mathcal{C}_i^{\mathcal{V}} = \{(C_l^{\mathcal{V}}, P_i(C_l^{\mathcal{V}})\}, t_u, R, t, \theta)$

```
 1: Calculate the maximum and minimum speeds on the x-axis and the y-
    axis, i.e., ${maxX, maxY, minX, minY}$.
 2: Construct the inferred region set $IRS_{t_u} = \mathbb{C}_i^{\mathcal{L}}$.
 3: for $j$ from $t_u + 1$ to $t$ do
 4:     Construct an empty inferred region set $IRS_j = \emptyset$
 5:     for each $IR \in IRS_{j-1}$ do
 6:         for each $(C_l^{\mathcal{V}}, P_i(C_l^{\mathcal{V}}) \in \mathcal{C}_i^{\mathcal{V}}$ do
 7:             Construct a new inferred region $IR'$ according to Lemma 4.1
 8:             if $IR'.l[1] + minX \cdot (t - j) \leq R.u[1]$
                $IR'.u[1] + maxX \cdot (t - j) \geq R.l[1]$
                $IR'.l[2] + minY \cdot (t - j) \leq R.u[2]$
                $IR'.u[2] + maxY \cdot (t - j) \geq R.l[2]$ then
 9:                 Insert $IR'$ into $IRS_j$
10: $Sum = 0$
11: for each $IR \in S_t$ do
12:     $Sum = Sum + IR.p$
13: if $Sum \geq \theta$ then
14:     Return TRUE
15: Return FALSE
```

their probabilities are summed up and compared with the threshold θ. If it is no smaller than θ, the moving object may possibly belong to the range query result; otherwise, it is discarded.

A query time t that is smaller than the update time t_u may be allowed. To support verification with such times, a scheme is deployed where the velocity domain is reversed. For example, a velocity cell $C_9^{\mathcal{V}} = [0, 0.1] \times [0, 0.1]$ is reversed to the symmetric cell $C_3^{\mathcal{V}} = [-0.1, 0] \times [-0.1, 0]$. Having reversed the domain, the previous algorithm can be used with query time t replaced by $2t_u - t$.

4.2 Monte Carlo Simulation

Monte Carlo simulation is a randomized method that simulates the motion of an uncertain moving object between time t_u and a query time t. Again, we first assume that the query time t is after time t_u.

The Monte Carlo simulation procedure is summarized in Algorithm 2. Given an error rate ϵ and a system-specified confidence δ,

Algorithm 2 Monte Carlo Query Verification
$(\mathcal{C}_i^{\mathcal{L}} = \{(C_j^{\mathcal{L}}, p_i(C_j^{\mathcal{L}})\}, \mathcal{C}_i^{\mathcal{V}} = \{(C_l^{\mathcal{V}}, p_i(C_l^{\mathcal{V}})\}, t_u, R, t, \theta, \epsilon, \delta)$

```
 1: Clear Success Counter $SC = 0$
 2: Calculate $N = 2\ln(1/\delta)/\epsilon^2\theta$
 3: for sample a number $l$ from $1$ to $N$ do
 4:     Randomly pick a location $loc$ depending on the location distribution
        of $o_i$ at update time $t_u$
 5:     for time point $z$ from $t_u + 1$ to $t$ do
 6:         Randomly pick a velocity $vel$ depending on the velocity distribu-
            tion of $o_i$
 7:         $loc = loc + vel$
 8:     if $loc \in R$ then
 9:         $SC = SC + 1$
10: if $SC \geq \theta N$ then
11:     Return TRUE
12: Return FALSE
```

the algorithm first calculates a simulation number N that indicates how many simulation steps are necessary. In each step, the algorithm picks one location based on the location distribution of the object at update time t_u. The movement of the object is then simulated from time $t_u + 1$ to the query time t. At each time point, the algorithm selects the velocity of the object following the velocity distribution. The object moves to its next location according to the selected velocity. After finishing the motion simulation between times $t - 1$ and t, the algorithm determines whether the object is within the query region R. If so, the success counter SC is incremented by 1. The object is included in the result if the total times of success during the simulation is no less than θN.

By the following lemma from sampling theory, the algorithm has high probability of determining whether an object is in the result of a probabilistic range query with high confidence if the sampling number N is sufficiently large [16].

LEMMA 4.2. *When $N \geq 2\ln(1/\delta)/\epsilon^2\theta$, Algorithm 2 discovers objects with probability no less than $(1 - \epsilon)\theta$ in query range R at time t, with confidence no less than $1 - \delta$.*

When the query time t is earlier than t_u, a similar technique to that of reversing the velocity domain is adopted, which facilitate the inference of probabilities of the objects in the query range.

5. MOVING OBJECT INDEXING

With movement inference in place, we proceed to cover the index structure for uncertain moving objects that enables querying and update. The basic structure for uncertain moving objects follows the B^x-tree, developed for certain moving objects [3,8,9]. We first cover the principles underlying the B^x-tree, then integrate our uncertain moving object model with the B^x-tree.

5.1 The Standard B^x-Tree

The B^x-tree [8] is the first proposal for using the B$^+$-tree to index moving objects. It applies a procedure that maps a 2D moving point object represented as a linear function to a point location in 1D space. This point is then indexed by a B$^+$-tree.

The B^x-tree assumes that object locations are updated at least every T time units, and it partitions the time dimension into intervals of length T. Each interval has a so-called reference time that belongs to the interval. A logical sub-tree, with a consecutive key range, is maintained for each interval. An object is inserted into the sub-tree with the interval that overlaps with the object's update time. This is done as follows: First, the position of the object's linear function as of its interval's reference time is determined. Second, this 2D point location is mapped to a 1D location by means of a Hilbert curve. To enable the Hilbert curve, the space in which the objects move is discretized by means of a regular grid. Third, the 1D location is prefixed by an identifier of the object's logical sub-tree.

An update of an object first deletes the old entry and then inserts the new entry into the sub-tree with the interval that overlaps the update time. Observe that due to the assumption about updates, only two sub-trees contain objects at any point in time. When objects are updated, they disappear from old sub-trees and are inserted into the most recent sub-tree.

To process a range query, the query is applied to each sub-tree in turn. For a sub-tree, the query range is expanded so that it takes into account the reference time used in the sub-tree and the maximum velocity of all the objects. This way, the enlarged query is guaranteed to retrieve all objects that may qualify for the query, in addition to some false positives. The enlarged query region is intersected with the Hilbert curve, which results in a sequence of 1D range queries. These are then issued against the index. The results for all sub-trees are combined, and filtering is applied to eliminate false positives.

5.2 Structure for Uncertain Moving Objects

Following the standard B^x-tree, we partition the time domain into equal-length intervals T and assume that each object is updated at least once during any such interval. At each point in time, the index maintains a logical sub-tree for two consecutive intervals. Figure 5 illustrates the logical sub-trees and the idea of time partitioning. Sub-trees BT_0 and BT_1 are responsible for moving objects with updates during odd and even time intervals, respectively. Specifically, BT_0 (BT_1) stores all objects updated during intervals $[2jT, (2j+1)T)$ ($[(2j+1)T, (2j+2)T)$) for $j > 0$.

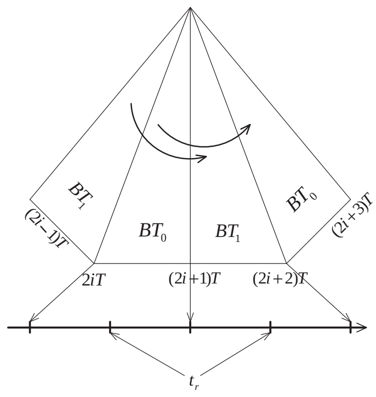


Figure 5: Index Rollover Operation

All objects in a sub-tree BT_l ($l \in \{0, 1\}$) are indexed as of a reference time t_r in the time interval of the sub-tree. If an object's update time is different from the reference time of the sub-tree, the inference method is run to transform the object's distributions to the reference time.

A roll-over is invoked at time $2jT$ ($(2j+1)T$), to reconstruct the sub-tree BT_1 (BT_0) to switch to a new time interval $[(2j+1)T, (2j+2)T)$ ($[(2j+2)T, (2j+3)T)$) from the previous interval $[(2j-1)T, 2jT)$ ($[2jT, (2j+1)T)$). Since each object by assumption is updated once in any interval of length T, the sub-tree becomes empty before being replaced. This assumption can be removed by forcing updates of all objects that remain in the sub-tree at destruction time.

When an object with location distribution $\mathcal{L}_i^{t_u}$ and velocity distribution $\mathcal{V}_i^{t_u}$ is updated at time t_u, each location cell with non-zero probability for $\mathcal{L}^{t_u}$ is indexed separately. The example shown in Figure 2 has 4 such cells. Specifically, the non-zero cells in the space domain are indexed along with the probability and the complete information on the velocity distribution.

To avoid redundant data records, the exact information on an object, including its location distribution $\mathcal{L}_i^{t_u}$, velocity distribution $\mathcal{V}_i^{t_u}$, and update time t_u, are stored in a data file. Leaf entries in the B$^+$-tree reference the corresponding object records in that file.

Similar to the B^x-tree, the velocity information on the objects in the index is kept together with each logical subtree in the modified index. In particular, for each subtree, a main-memory 2D histogram of the maximum and minimum velocities of all objects is maintained and updated at every insertion. Note that the deletion operation does not alter the velocity bounds maintained in the histograms, due to the excessive cost on updating them by finding the objects with the maximum or minimum velocities. The details of the velocity histogram can be found elsewhere [8].

5.3 Index Update

When an uncertain moving object is to be updated at time t_u, all leaf entries referencing the object are removed first, and new references are inserted based on the updated information of the object using Algorithm 3.

To insert an uncertain moving object, the system first identifies the sub-tree in which to insert the object. The system then estimates the location probability distribution for the object at the sub-tree's reference time t_r, by running the IR-based inference method from Section 4. While Algorithm 1 tests the probability that a moving object qualifies for a range query, it can also be extended easily to discover the spatial cells with non-zero probabilities of the object at the reference time t_r.

Specifically, as summerized in Algorithm 3, for an insertion, the system first infers the distribution of the object at the reference time t_r based on the location distribution $\mathcal{C}_i^{\mathcal{L}}$, velocity distribution $\mathcal{C}_i^{\mathcal{V}}$, and the update time t_u (Lines 1–2). Then all spatial cells with non-zero probabilities are extracted in a cell set CS (Line 3). Finally, for each such cell, a leaf entry is inserted that references the record of the object in the data file (Lines 4–5). The insertion operation follows the standard strategy used in the B^x-tree, enabling reuse of the B$^+$-trees in commercial databases.

Algorithm 3 Insertion (Location Cells $\mathcal{C}_i^{\mathcal{L}}$, Velocity Cells $\mathcal{C}_i^{\mathcal{V}}$, update time t_u, sub-tree BT_i)

```
1: Get the reference time $t_r$ of the sub-tree $BT_i$
2: Generate inferred rectangles at time $t_r$ by Algorithm 1 with $\mathcal{C}_i^{\mathcal{L}}$, $\mathcal{C}_i^{\mathcal{V}}$ and $t_u$
3: Find all spatial cells that overlap with the inferred rectangles and store them in cell set $CS$
4: for each cell $C_j^{\mathcal{L}} \in CS$ do
5:     Insert $o_i$'s reference into the cell $C_j^{\mathcal{L}}$ indexed in $BT_i$
```

The deletion procedure for the index is similar to the insertion in Algorithm 3. Specifically, the system first locates the record of the object in the data file and gets the location and velocity distribution of the object at the update time t_u. Then the deletion performs by executing the same steps (Line 1-4) as in Algorithm 3, except that the cell is deleted from the index (Line 5). Due to space constraints, we omit the details on the implementation of deletion.

5.4 Velocity-Based Partitioning

The efficiency of the query processing depends on the minimum and maximum speeds of the objects, which are maintained at the roots of the sub-trees. In our uncertainty model, the velocity of each moving object covers a range in the velocity domain, leading to larger query expansions and worse pruning effectiveness.

Motivated by the location partitioning technique in the ST^{2}B-tree [3], we propose a velocity-based partitioning method to decrease query expansion. As in the ST^{2}B-tree, a sub-tree BT_j is partitioned into K logical sub-trees $BT_{j1}, \ldots, BT_{jK}$, each of which is used to index the moving objects in a specified velocity range.

The ST^{2}B-tree applies density-based clustering to partition the space domain into K parts. We are unable to follow the same strategy for our velocity partitioning, for two reasons. First, the ST^{2}B-tree indexes velocities that are exact points in the velocity space. With uncertain moving objects, each velocity is represented by a distribution, rendering any direct clustering of them impossible. Second, the purpose of the space domain partitioning is to find regions with similar moving-object densities. In our case, we partition the velocity space to decrease the query expansion during query processing, which is decided by the tightness on the velocity bounds. If we use *Velocity Minimal Bounding Rectangles* (VMBRs) to denote the minimal rectangles in the velocity space covering the distributions of all uncertain moving objects, it is important to reduce the volume of the VMBRs recorded on each logical sub-tree BT_{jl}.

When constructing a sub-tree BT_j, the system needs to initialize the VMBRs of its K sub-tree partitions. A new moving object o_i to be inserted into BT_j is assigned to the sub-tree partition BT_{jl} that minimizes the enlargement of the VMBRs among all sub-trees. To achieve tight bounds on the VMBRs, we borrow the idea underlying the R-tree's split strategy. We use the velocity ranges of exactly K moving objects sampled from the previous sub-tree as the initial VMBRs of the new sub-tree's partitions. Specifically, the greedy selection method in Algorithm 4 is adopted to pick these seeds of the k partitions. An object o_i is randomly selected from the sample.

Algorithm 4 Select Seeds (Moving object set sample D', number of sub-tree paritions K)

```
1: Empty seed set $S$
2: Randomly pick an object $o_i$ from $D'$
3: Insert $o_i$ into $S$
4: **for** $j$ from 2 to $K$ **do**
5:     Pick the object $o_i \in D' - S$, with maximal VMBR enlargement
       w.r.t. any seed in $S$
6: Initialize the VMBRs of the sub-tree with the velocity ranges of the
   objects in $S$
```

In the following $K - 1$ iterations, the new object with the largest VMBR expansion with respect to the currently selected ones is chosen as a seed. The algorithm stops with exactly K moving objects, whose velocity ranges are recorded at the roots of the sub-tree partitions.

6. QUERY PROCESSING

We proceed to cover in turn the processing of probabilistic range queries and k-PNN query.

6.1 Probabilistic Range Query

Recall from Section 3 that a probabilistic range query specifies a probability threshold θ, a spatial range R, and a time t and retrieves all objects that belong to R at time t with probability θ. To process this query, we employ a two-step method that comprises a growing step and a verification step.

In the growing step, the system constructs a candidate object list consisting of objects that may have at least probability θ of being in range R at time query range a time t. This is accomplished by issuing the range query on the index, retrieving all cells whose movement may satisfy the range query. Since the object id and pointer to the physical storage are kept in the leaf nodes, a list of candidate ids can be produced. An object may be included in the result due to several cells. This redundancy is removed by checking whether an identical id has already been added to the candidate list when retrieving a moving object from a leaf node.

In the verification step, two algorithms are employed in order. The IR-based verification (Algorithm 1) is run first as a filter because of its high efficiency. Objects that pass the filter are subjected to Monte-Carlo simulation verification (Algorithm 2). The objects that also pass this filter are added to the result set. The procedure for range queries is summarized in Algorithm 5.

Algorithm 5 Range Query Search (Query range R, query time t, probability threshold θ, index tree Tr)

```
1: Clear result set $RS$ and construct a candidate id list $CL$ by retrieving
   the ids of all objects with all spatial cells in $Tr$ that satisfy the range
   query
2: **for** each object $o_i \in CL$ **do**
3:     **if** $o_i$ passes the IR-based verification **then**
4:         **if** $o_i$ passes the Monte-Carlo verification **then**
5:             Add $o_i$ to the results set $RS$
6: Return all objects in $RS$ as the result
```

6.2 k-PNN Query

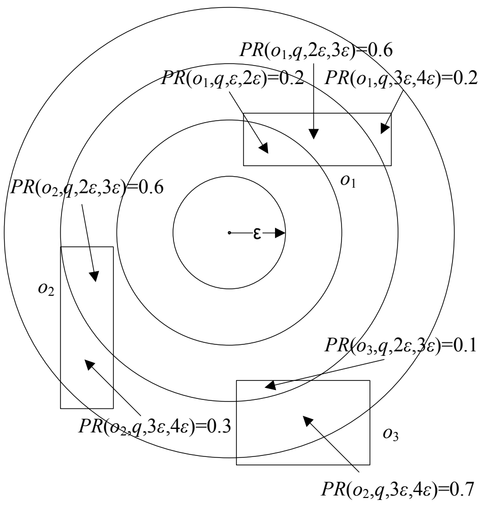
In index structures for certain moving objects, a k-nearest neighbor query retrieves k objects with minimal distance to a query point q at query time t. The query can be processed by issuing a series of range queries centered at q with a radius r that is gradually increased 0 towards ∞ until exactly k objects are found. This method falls short for the k-PNN query on uncertain moving objects because the locations of the moving objects are no longer single points.

Compared with existing k-PNN query processing on static probabilistic databases [1, 4], it is more difficult to answer the k-PNN query on uncertain moving objects because the optimization techniques proposed in previous works rely on an oracle that can arbitrarily retrieve the probability of objects in any distance interval. In uncertain moving object databases, unfortunately, the computation of the exact distribution of an object at query time is very expensive.

We proceed to propose an algorithm that depends only on the results of a series of range queries, each of which is a circular region centered at query point q. Since queries with smaller regions are expected to have the lower computation costs, the queries are issued in order of increasing radius, thus reducing the I/O and CPU costs. A pair of a lower and an upper bound probability is maintained for each object. The algorithm terminates when the kth highest lower bound probability is larger than the upper bound probability of any other object.

Before delving into the details of the algorithm, the exact nearest neighbor probability of an uncertain moving object o_i is defined based on its spatial distribution at query time t. Given a query q, we use $PC(o_i, q, r)$ to denote the probability of o_i belonging to the circle centered at q with radius r, and we use $PR(o_i, q, r_1, r_2)$ to denote the probability of o_i belonging to the ring centered at q with radius between r_1 and r_2 ($r_1 \leq r_2$).

Figure 6 depicts the distributions of three moving objects o_1, o_2, and o_3 at query time t. The notation $PR(o_1, q, 2\epsilon, 3\epsilon)$ is the probability that object o_1 is located in the ring between 2ϵ and 3ϵ of query q. In the figure, the probability is 0.6. The circle $PC(o_1, q, 3\epsilon)$ can be expressed as $PR(o_1, q, 0, \epsilon) + PR(o_1, q, \epsilon, 2\epsilon) + PR(o_1, q, 2\epsilon, 3\epsilon) = 0 + 0.2 + 0.6 = 0.8$.



$PC(o_1,q,2\varepsilon)$=0.2	$PC(o_1,q,3\varepsilon)$=0.8	$PC(o_1,q,4\varepsilon)$=1
$PC(o_2,q,2\varepsilon)$=0	$PC(o_2,q,3\varepsilon)$=0.6	$PC(o_2,q,4\varepsilon)$=0.9
$PC(o_3,q,2\varepsilon)$=0	$PC(o_3,q,3\varepsilon)$=0.1	$PC(o_3,q,4\varepsilon)$=0.8

Figure 6: Example k-PNN Query

Thus, the exact nearest neighbor probability can be calculated as [1, 4]:

$$NNP_i = \int_{r=0}^{\infty} \frac{\partial PC(o_i, q, r)}{\partial r} \prod_{j \neq i} (1 - PC(o_j, q, r))\ dr$$

This equation is hard to compute precisely, but can be approximated by replacing the integral with a summation that splits the space into rings.

$$ANNP_i = \sum_{l=1}^{\infty} PR(o_i, q, \epsilon(l-1), \epsilon l) \prod_{j \neq i} (1 - PC(o_j, q, \epsilon l))$$

Algorithm 6 processes the k-PNN query. For each uncertain moving object o_i, the algorithm maintains a lower bound low_i, an upper bound up_i, as well as an accumulated probability acc_i on the probability of o_i being the nearest neighbor. With the increase of the query radius from $\epsilon(m-1)$ to ϵm for some positive finished iteration number m, the algorithm updates the lower bound, the upper bound, and the accumulated probabilities according to the following lemma.

LEMMA 6.1. *For any positive integer m, we have the lower bound and upper bound on $ANNP_i$ as follows:*

$$ANNP_i \geq \sum_{l=1}^{m} PR(o_i, q, \epsilon(l-1), \epsilon l) \prod_{j \neq i} (1 - PC(o_j, q, \epsilon l))$$

$$ANNP_i \leq \sum_{l=1}^{m} PR(o_i, q, \epsilon(l-1), \epsilon l) \prod_{j \neq i} (1 - PC(o_j, q, \epsilon l)) + \prod_{j} (1 - PC(o_j, q, \epsilon m))$$

Algorithm 6 **k-PNN Query** (Query location q and result size k)

```
1:  Construct $up_i$, $low_i$, and $acc_i$ for each object $o_i$
2:  Set radius $r = 0$ and clear result set $RS$
3:  while stopping condition is not reached do
4:      Increment radius $r$ by $\epsilon$
5:      Issue a probabilistic range query centered at $q$ with radius $r$
        and probability threshold 0
6:      Update $acc_i$ for any $o_i$ in the range query result
7:      Update $up_i$ and $low_i$ according to Lemma 6.1
8:      Put $k$ objects with highest lower bound probabilities into $RS$
9:      if the smallest $low_i$ in $RS$ is higher than the $up_j$ of all ob-
        jects $o_j \notin RS$ then
10:         Set stopping condition to TRUE
11:     if $acc_i = 1$ for any $o_i$ then
12:         Set stopping condition to TRUE
13: Return all objects in $RS$ as the result
```

PROOF. Based on the definition of $ANNP_i$, we can derive:

$$ANNP_i = \sum_{l=1}^{m} PR(o_i, q, \epsilon(l-1), \epsilon l) \prod_{j \neq i} (1 - PC(o_j, q, \epsilon l)) + \sum_{l=m+1}^{\infty} PR(o_i, q, \epsilon(l-1), \epsilon l) \prod_{j \neq i} (1 - PC(o_j, q, \epsilon l))$$

Since both parts are positive, we reach the lower bound by eliminating the second part from the equation. In addition, since $1 - PC(o_i, q, \epsilon l) \leq 1 - PC(o_i, q, \epsilon m)$ for any $l \geq m$, the second part can be upper bounded as:

$$\begin{aligned}
&\sum_{l=m+1}^{\infty} PR(o_i, q, \epsilon(l-1), \epsilon l) \prod_{j \neq i} (1 - PC(o_j, q, \epsilon l)) \\
\leq\ & \sum_{l=m+1}^{\infty} PR(o_i, q, \epsilon(l-1), \epsilon l) \prod_{j \neq i} (1 - PC(o_j, q, \epsilon m)) \\
=\ & (1 - PC(o_i, q, \epsilon m)) \prod_{j \neq i} (1 - PC(o_j, q, \epsilon m)) \\
=\ & \prod_{j} (1 - PC(o_j, q, \epsilon m))
\end{aligned}$$

This completes the proof of the lemma. $\square$

Since $acc_i = PC(o_i, q, \epsilon l)$, the probability in the ring $PR(o_i, q, \epsilon(l-1), \epsilon l)$ is the difference between the new accumulated probability and the previous one. Thus, the new lower and upper bound probabilities can be updated as implied by the formulas above. The algorithm stops if the lowest lower bound probability of the top-k objects is larger than the upper bound probabilities of all other objects or if the accumulated probability of some object o_i reaches 1.

Recall the example of the uncertain moving objects in Figure 6. Table 2 lists the statuses of the objects after iterations 2–4. Since no object has positive probability in the circle around query q, the first iteration is not interesting. In the second iteration, object o_1 has probability 0.2 in the ring; the probability of o_1 being nearest neighbor is thus at least 0.2 by the lower bound rule in Lemma 6.1. The upper bound probabilities of other two objects are both 0.8 because the algorithm has no idea of their distributions so far. In the third iteration, the algorithm further tightens the bounds after observing the probabilities of the objects in the new ring. Another iteration is still necessary, since the upper bound of o_3 is larger than

Iteration	Object	acc_i	up_i	low_i
Iteration 2	o_1	0.2	0.2	1
	o_2	0	0	0.8
	o_3	0	0	0.8
Iteration 3	o_1	0.8	0.524	0.596
	o_2	0.6	0.108	0.18
	o_3	0.1	0.08	0.152
Iteration 4	o_1	1	0.528	0.528
	o_2	0.9	0.108	0.108
	o_3	0.8	0.08	0.08

Table 2: Algorithm Running on Figure 6

Parameter	Setting
Max Update Time (sec.)	120
Number of objects(K)	**100**, 200, 300, 400, 500
θ in range query	0.1, 0.15, **0.2**, 0.25, 0.3
Range query length (km)	1, 1.5, **2**, 2.5, 3
Query time (sec)	10, 20, **30**, 40, 50
k in k-PNN Query	10, 20 **40**, 80, 160
K in velocity partition	2,3,4,**5**,6,7,8
number of bits on spatial dimension	5, 6, **7**, 8, 9
number of bits on velocity dimension	2, **3**, 4, 5

Table 3: Experimental Parameters and Settings

the lower bound of o_2. This is resolved after the fourth iteration in which o_1 is found to be outside the circle with zero probability.

7. EMPIRICAL STUDY

In Section 7.1, we discuss the settings of the experiments. In Section 7.2, we compare the motion prediction effectiveness of the certain and uncertain moving object models. In Section 7.3, we study the performance of the range query and k-PNN query.

7.1 Experimental Settings

We generate synthetic data sets to test the effectiveness and efficiency of the paper's proposal. Objects move in a 100 km x 100 km work square. The objects are initially distributed uniformly in this space. The directions and speeds of the objects are subsequently generated randomly at each each time point. To simulate a real environment, the objects are divided into 5 classes with different maximum speed $V_i^{\max}$: 30, 60, 90, 120, 150 km/hour.

Assuming that the velocities of the objects rely on the traffic conditions at their locations, the generated velocities follow a function that depends on the population of the objects' current neighborhoods. Specifically, given an object o_i, we count the number $count_i$ of objects appearing in a range of 6 km of o_i. A reference speed $V_i = \max\{0, V_i^{\max} - 0.2 \times count_i\}$ is calculated. Given the reference speed, a Gaussian distribution $N(\mu, \sigma)$ (it is re-sampled if meeting a negative sample value) is adopted to model the uncertainty on the speed, with $\mu = V_i$ and $\sigma = V_i/9$.

If an object is about to leave the work square, a reverse direction is taken to keep it from exiting the square. After the velocity of the moving object is decided, 10 other velocity samples are taken to approximate the velocity distribution. The velocity cells of the object are determined by counting the sample velocities falling in the cells. The location distribution at the next time point is also generated with the velocity samples, by simulating the location with these velocities. By default, the spatial dimensions and velocity dimension is represented by 7 and 3 bits, respectively.

In the study, we vary the data generation parameters as well as the index structure settings. In particular, we list the tested parameters and settings in Table 3, where the default values are shown in bold.

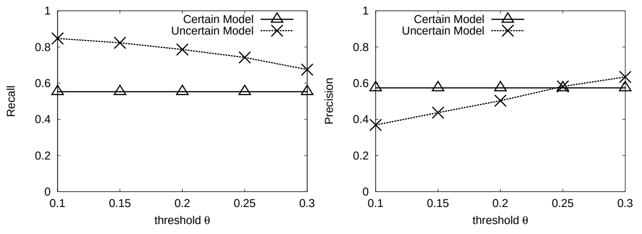
In the experiments, we consider the algorithms introduced in Section 6 for answering probabilistic range queries and k-PNN queries. We emphasize that although query processing techniques for probabilistic databases have been proposed, these do not match well our setting. Most of them utilize operators that retrieve probabilities for arbitrary distance intervals to a query point. To make this possible in our setting, the complete spatial distributions of all moving objects at the query time must be generated before beginning the query processing, which incurs very high CPU and I/O costs.

All code used in our experiments is written in C++. The page size is fixed at 4KB, and a 50-page LRU buffer is used. All experiments are conducted on a PC with a 2.33GHz Core2 Duo CPU and 3.25GB of memory, running Windows XP.

7.2 Certainty Versus Uncertainty

In the next experiments, we validate the robustness of motion prediction results for the uncertain moving object model when compared to a traditional certain model. Only range queries are considered, since the k-PNN query is not comparable to a conventional k-NN query on a certain model. A simple linear motion function is adopted as the certain model. It uses the average velocity and location from the distribution of uncertain moving objects at a reference time.

Assume that at querying time t, the correct answer to a range query R is a subset of moving objects, $S \subseteq D$, while the uncertain model (or certain model) returns an answer set A. We report the recall and precision of these answers, i.e., $\frac{|S \cap A|}{|S|}$ and $\frac{|S \cap A|}{|A|}$, respectively. The impact of three parameters are tested, including the probability threshold θ, hte querying time from the reference time, and the length of the range query.

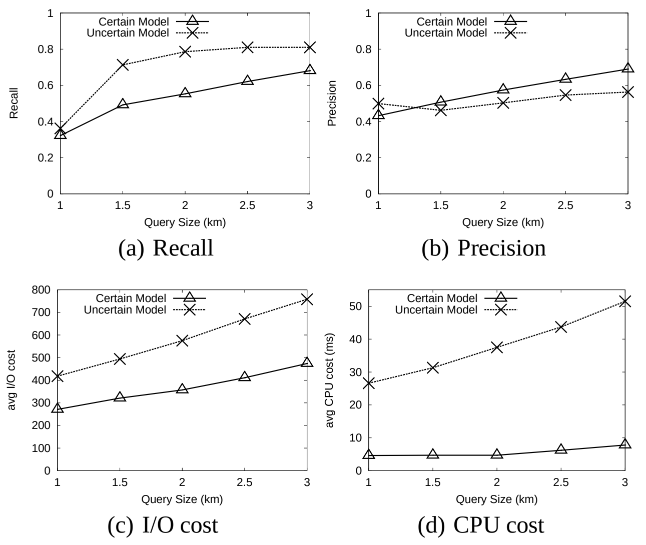


(a) Recall (b) Precision

Figure 7: Threshold θ vs. Prediction Error

In Figure 7, we evaluate the influence of the probability threshold θ. Since it does not affect prediction with exact location and velocity, recall and precision remain constant in the certain model. The uncertain model, on the other hand, obtains a lower recall and a higher precision as the threshold is increased. This is due to the shrinkage on the query result in the uncertain model. From the figure, we can conclude that the uncertain model provides more robust results than the certain model, showing an advantage for recall. The precision of the uncertain model is also competitive with that of the certain model. When θ reaches 0.3, the uncertain model is better than the certain model on both measurements.

In Figure 8, we present the results of varying the range query size. Again, the uncertain model provides results with the best recall, which is almost 0.8 when the query size is larger than 1.5 km. Both models exhibit an increasing precision with the expansion of the range queries. The precision loss incurred by the adoption of the uncertain model is always less than 10%.



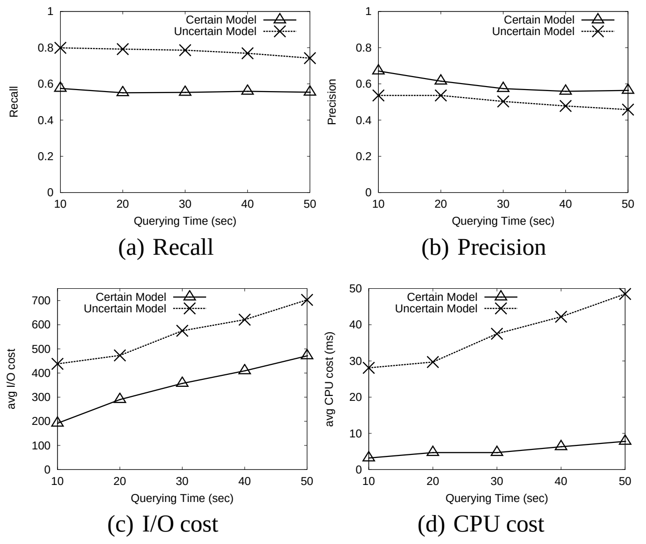
(a) Recall (b) Precision

(c) I/O cost (d) CPU cost

Figure 8: Query Size vs. Prediction Error

We also consider the query processing costs of the two models for varying range query sizes. The figure shows that the uncertain model incurs higher I/O cost than the certain model, i.e., 70% to 90% higher. This is because data pages have a higher fan-out for the certain model than for the uncertain model; these are about 150 and 25, respectively. In addition, a query retrieves more objects with the uncertain model than with the certain model. Thus the query processing I/O with the uncertain model is expected to be much higher than for the certain model. However, since object records are saved in data pages in random order, each object retrieval performs a random access. This leverages the negative effect of a smaller fan-out and explains why the difference of I/O costs between two models is less than expected.

The query processing time for the uncertain model is about 5–9 times longer than for the certain model due to the additional computation (i.e., rectangle inference and Monte Carlo simulation). With the uncertainty model, throughput is 20–40 sequential queries per second.



(a) Recall (b) Precision

(c) I/O cost (d) CPU cost

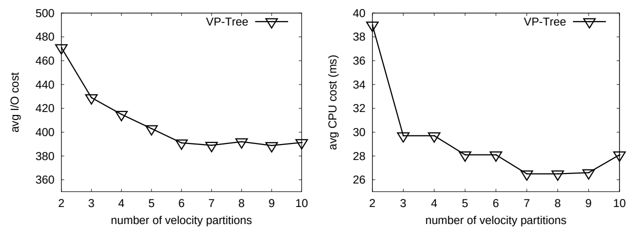
Figure 9: Query Time vs. Prediction Error

Figure 9 tests the query time parameter, which is measured as the number of seconds since the last update of the objects. Surprisingly, recall and precision are affected only by the query time. Even when predicting locations 50 seconds after the update time, the recall of the uncertain model is still at 0.75, meaning that 75% of the correct answers are captured with the uncertain model.

Again, we examine the query processing costs for the two models. The findings in Figure 9(c)-9(d) mirror those shown in Figure 8.

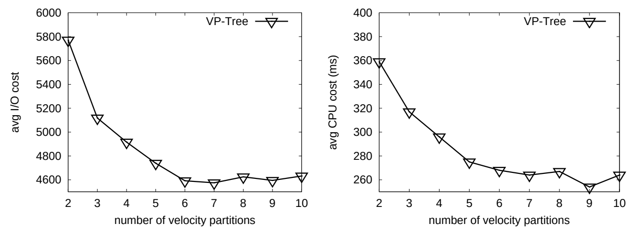
7.3 Efficiency Tests

We proceed to study the performance of the index without velocity partitions (NP-tree) and with velocity partitions (VP-tree). First, Figures 10 and 11 cover the effects of the number of velocity partitions in the VP-tree on range and k-PNN query performance.



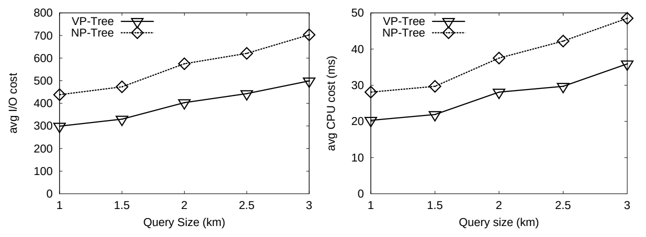
(a) I/O cost (b) CPU cost

Figure 10: Velocity Partitions vs. Range Query Efficiency



(a) I/O cost (b) CPU cost

Figure 11: Velocity Partitions vs. k-PNN Query Efficiency



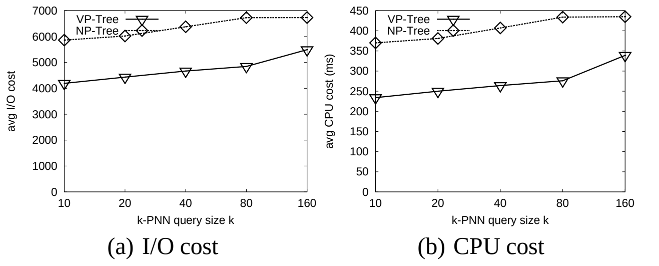
(a) I/O cost (b) CPU cost

Figure 12: Range Query Size vs. Efficiency

The figures report average I/O and CPU costs while varying the number of velocity partition from 2 to 10. The best overall performance for range queries is obtained when using more than 7 velocity partitions. With velocity partitions, we maintain velocity ranges at the roots of the logical sub-trees. This yields reduced query enlargements.

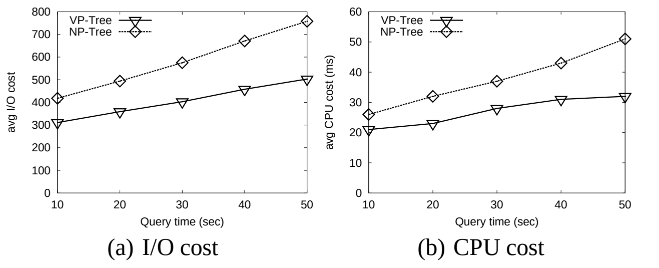
A query needs to search all velocity sub-tree partitions. When increasing the number of partitions beyond 7, the query processing costs starts increasing slightly. This is because the costs of the increased number of sub-tree traversals start to offset the benefit of the reduced query enlargement.

Next, we compare the VP-tree and the NP-tree. Figures 12–15 report on the performance of range and k-PNN queries when varying the range query size, the parameter k, and the query time. The velocity-based partitioning in the VP-tree yields the best performance of both types of queries and under all the parameter settings considered.



(a) I/O cost (b) CPU cost

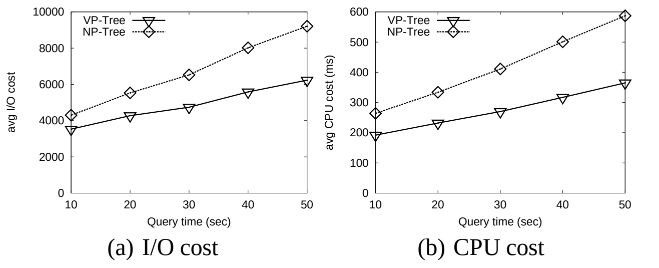
Figure 13: k-PNN Query Size k vs. Efficiency



(a) I/O cost (b) CPU cost

Figure 14: Query Time vs. Range Query Efficiency

Figure 12 shows the average range query cost when using the two tress and varying the query size from 1 km to 3 km. As expected, the cost increases with the query size.



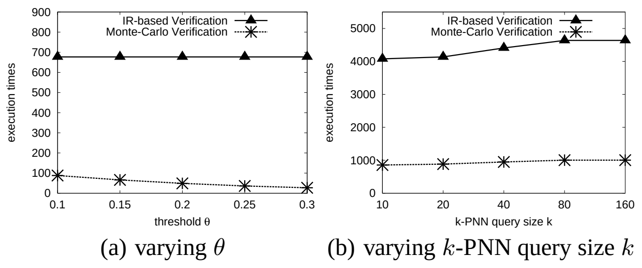
(a) I/O cost (b) CPU cost

Figure 15: Query Time vs. k-PNN Query Efficiency

Figure 13 concerns the k-PNN query performance when varying the number k of requested nearest neighbors candidates from 10 to 160. As expected, the I/O and CPU costs increase when using both indexing methods. However, the VP-tree incurs the lowest costs.

Next, we consider the query processing performance when varying the query time from 10 to 50 seconds after the last update of the objects. The costs for the range query and k-PNN query are shown in Figure 14 and Figure 15, respectively. We find that as the query time increases, both the I/O and CPU costs increase. This is natural, as a larger query time yields a larger query expansion. As before, the VP-tree outperforms the NP-tree.

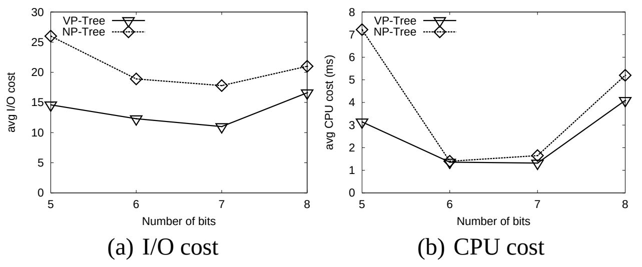
Figure 16 shows the average numbers of times IR-based verification and Monte-Carlo verification are invoked during the processing of a query. Monte-Carlo verification is computational costly, while IR-based verification incurs significantly lower cost. Figure 16 shows that IR-based verification is executed about 700 (for



(a) varying θ (b) varying k-PNN query size k

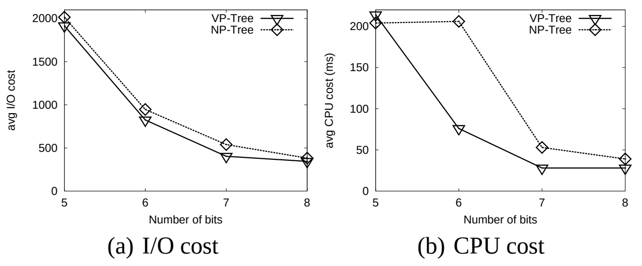
Figure 16: IR-Based vs. Monte-Carlo Verification

range queries, varying θ) and 4000 (for k-PNN queries, varying k) times. Due to the resulting filtering of candidates, the costly Monte-Carlo verification is executed less than 100 times for range queries and about 1,000 times for k-PNN queries.



(a) I/O cost (b) CPU cost

Figure 17: Number of Bits in the Spatial Dimension vs. Update Efficiency



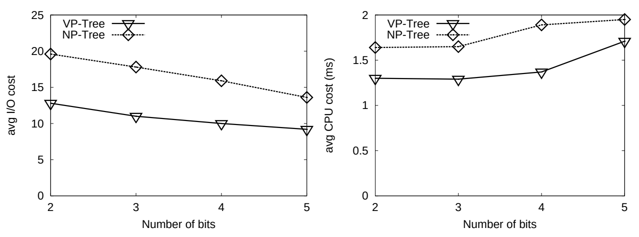
(a) I/O cost (b) CPU cost

Figure 18: Number of Bits in the Spatial Dimension vs. Range Query Efficiency

7.4 Partition Granularity

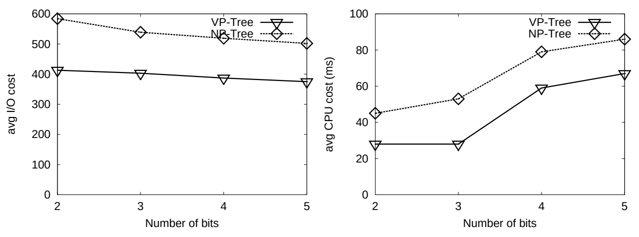
Our indexing approach relies on a grid partitioning of the location and velocity spaces of the moving objects. The granularity of this partitioning is expected to affect the index performance. In all the previous experiments, the location and velocity spaces were partitioned using a fixed granularity, namely 2^7 and 2^3, respectively. We proceed to investigate the performance implications of different partitioning granularities.

We first consider the location space. The partitioning granularity is represented by the number of bits used to generate the partitioning and the space filling curve. For example, if the number of bits is 5, the space is partitioned into $2^5 \times 2^5$ cells. Figures 17 and 18 show that when the number of bits is 7, both trees exhibit the best update costs. With fewer bits, the update cost is higher because each cell contains a large number of objects. With more bits, the uncertainty region of an object is partitioned into more cells, which incurs additional key insertions and deletions. Therefore, the I/O and CPU costs per update increase when more bits are added. As



(a) I/O cost (b) CPU cost

Figure 19: Number of Bits in the Velocity Dimension vs. Update Efficiency



(a) I/O cost (b) CPU cost

Figure 20: Number of Bits in the Velocity Dimension vs. Range Query Efficiency

for range queries, using more bits yields smaller cells. Thus, a query retrieves fewer false positives, which reduces the query cost.

Figures 19 and 20 show the effects of varying the number of bits used for partitioning the velocity space.

The I/O costs decrease for both updates and queries as more bits are used. With a finer partitioning of velocity space, the uncertain velocity of objects fall into more cells, each of which represents a smaller velocity range. The resulting more accurate information for doing movement inference yields tighter inferred rectangles that intersect with fewer spatial cells. Thus, fewer key insertions and deletions occur during updates, and less data is accessed during query processing. In contrast, the CPU costs increases slightly with more bits. Although the I/O counts decrease with more bits, more computation is needed during movement inferencing.

8. CONCLUSION

While the indexing of the current positions of moving objects has received substantial attention, the majority of previous proposals assume that the position of an object is represented by a near-past position or by a linear function of time based on an exact near-past position and velocity. In contrast, this paper makes the realistic assumption that the current and near-future position of an object is to be determined from a near-past position and velocity for which only a stochastic distribution is known. Thus, positions are uncertain. The paper presents techniques that enable the efficient inferencing of current and near-future uncertain locations from past uncertain velocity and location information. Further, the paper demonstrates how it is possible to index the resulting uncertain moving objects by means of an adapted B^x-tree. And it provides techniques for processing probabilistic range and nearest neighbor queries. An empirical study offers insight into pertinent design properties of the paper's proposals, demonstrating their practicality.

Acknowledgments

Meihui Zhang, Su Chen, Beng Chin Ooi and Zhenjie Zhang were supported by Singapore NRF grant R-252-000-376-279. Christian S. Jensen was a visiting scientist at Google Inc. from September 2008 to August 2009.

9. REFERENCES

[1] G. Beskales, M. A. Soliman, and I. F. Ilyas. Efficient Search for the Top-k Probable Nearest Neighbors in Uncertain Databases. *PVLDB*, 1(1):326–339, 2008.

[2] S. Chen, C. S. Jensen, and D. Lin. A Benchmark for Evaluating Moving Objects Indexes In *VLDB*. pp. 1574–1585, 2008.

[3] S. Chen, B. C. Ooi, K.-L. Tan, and M. A. Nascimento. ST^{2}B-Tree: A Self-Tunable Spatio-Temporal B$^+$-Tree Index for Moving Objects. In *SIGMOD Conference*, pp. 29–42, 2008.

[4] R. Cheng, J. Chen, M. F. Mokbel, and C.-Y. Chow. Probabilistic Verifiers: Evaluating Constrained Nearest-Neighbor Queries over Uncertain Data. In *ICDE*, pp. 973–982, 2008.

[5] R. Cheng, D. V. Kalashnikov, and S. Prabhakar. Evaluating Probabilistic Queries over Imprecise Data. In *SIGMOD Conference*, pp. 551–562, 2003.

[6] R. Cheng, D. V. Kalashnikov, and S. Prabhakar. Querying Imprecise Data in Moving Object Environments. *IEEE TKDE*, 16(9):1112–1127, 2004.

[7] S. Guo, Z. Huang, H. V. Jagadish, B. C. Ooi, and Z. Zhang. Relaxed Space Bounding for Moving Objects: A Case for the Buddy Tree. *SIGMOD Record*, 35(4):24–29, 2006.

[8] C. S. Jensen, D. Lin, and B. C. Ooi. Query and Update Efficient B$^+$-Tree Based Indexing of Moving Objects. In *VLDB*, pp. 768–779, 2004.

[9] C. S. Jensen, D. Tiešytė, N. Tradišauskas. Robust B+-Tree-Based Indexing of Moving Objects. In *MDM*, 9 pages, 2006.

[10] C. S. Jensen, D. Tiešytė, N. Tradišauskas. The COST Benchmark—Comparison and Evaluation of Spatio-Temporal Indexes. In *DASFAA*, pp. 125–140, 2006.

[11] H. Jeung, Q. Liu, H. T. Shen, and X. Zhou. A Hybrid Prediction Model for Moving Objects. In *ICDE*, pp. 70–79, 2008.

[12] M. Kornacker and D. Banks. High-Concurrency Locking in R-Trees. In *VLDB*, pp. 134–145, 1995.

[13] M. Kornacker, C. Mohan, and J. M. Hellerstein. Concurrency and Recovery in Generalized Search Trees. In *SIGMOD Conference*, pp. 62–72, 1997.

[14] H.-P. Kriegel, P. Kunath, and M. Renz. Probabilistic Nearest-Neighbor Query on Uncertain Objects. In *DASFAA*, pp. 337–348, 2007.

[15] D. Lin, C. S. Jensen, B. C. Ooi, and S. Šaltenis. Efficient Indexing of the Historical, Present, and Future Positions of Moving Objects. In *MDM*, pp. 59–66, 2005.

[16] M. Mitzenmacher and E. Upfal. *Probability and Computing: Randomized Algorithms and Probabilistic Analysis*. Cambridge University Press, 2005.

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

[18] V. Srinivasan and M. J. Carey. Performance of B-Tree Concurrency Control Algorithms. In *SIGMOD Conference*, pp. 416–425, New York, NY, USA, 1991. ACM Press.

[19] Y. Tao, C. Faloutsos, D. Papadias, and B. Liu. Prediction and Indexing of Moving Objects with Unknown Motion Patterns. In *SIGMOD Conference*, pp. 611–622, 2004.

[20] Y. Tao, D. Papadias, and J. Sun. The TPR*-Tree: An Optimized Spatio-Temporal Access Method for Predictive Queries. In *VLDB*, pp. 790–801, 2003.

[21] G. Trajcevski, O. Wolfson, K. Hinrichs, and S. Chamberlain. Managing Uncertainty in Moving Objects Databases. *ACM Trans. Database Syst.*, 29(3):463–507, 2004.