Querying Imprecise Data in Moving Object Environments

Reynold Cheng, *Student Member*, *IEEE*, Dmitri V. Kalashnikov, and Sunil Prabhakar, *Senior Member*, *IEEE*

Abstract—In moving object environments, it is infeasible for the database tracking the movement of objects to store the exact locations of objects at all times. Typically, the location of an object is known with certainty only at the time of the update. The uncertainty in its location increases until the next update. In this environment, it is possible for queries to produce incorrect results based upon old data. However, if the degree of uncertainty is controlled, then the error of the answers to queries can be reduced. More generally, query answers can be augmented with probabilistic estimates of the validity of the answer. In this paper, we study the execution of probabilistic range and nearest-neighbor queries. The imprecision in answers to queries is an inherent property of these applications due to uncertainty in data, unlike the techniques for approximate nearest-neighbor processing that trade accuracy for performance. Algorithms for computing these queries are presented for a generic object movement model and detailed solutions are discussed for two common models of uncertainty in moving object databases. We study the performance of these queries through extensive simulations.

Index Terms—Data uncertainty, probabilistic queries, range queries, nearest-neighbor queries.

1 Introduction

Systems for continuous monitoring or tracking of mobile objects receive updated locations of objects as they move in space [3], [11], [5]. Due to limitations of bandwidth and the battery power of the mobile devices, it is infeasible for the database to contain the exact position of each object at each point in time. For example, if there is a time delay between the capture of the location and its receipt at the database, the location values received by the object may be different from the actual location values. An inherent property of these applications is that object locations cannot be updated continuously. Following an update, the position of the object is unknown until the next update is received. Under these conditions, the data in the database is only an estimate of the actual location at most points in time. This inherent uncertainty affects the accuracy of the answers to queries. Fig. 1a illustrates how a nearest-neighbor query for point q can yield an incorrect result. Based upon the recorded locations x_0 and y_0 of objects o_1 and o_2, the database returns "o_1" as the object closest to q. However, in reality, the objects could have moved to positions x_1 and y_1 in which case "o_2" is nearer.

Due to the inherent uncertainty in the data, providing meaningful answers seems impossible. However, one can argue that, for most moving objects, the locations of objects cannot change drastically in a short period of time. In fact, the degree and rate of movement of an object is often constrained. For example, uncertainty models have been proposed for moving object environments in order to reduce the overhead of updates [24]. Such information can help address the problem. Consider the above example again. Suppose we can provide a guarantee that, at the time the query is evaluated, o_1 and o_2 could be no further than some distances d_1 and d_2 from their locations stored in the database, respectively, as shown in Fig. 1b. With this information, we can state with confidence that o_1 is the nearest neighbor of q. In general, the uncertainty of the objects may not allow us to determine a single object as the nearest neighbor. Instead, several objects could have the possibility of being the nearest neighbor.

The notion of probabilistic answers to queries over uncertain data was introduced in [24] for range queries, where the answer consists of objects along with the probability of each object lying in the query range. We extend this idea to answer nearest-neighbor queries—the answer consists of not a single object that is closest to the object, but a set of objects, each of which have the potential of being the nearest neighbor of the query point. In addition to identifying these objects, the probability of each object being the nearest neighbor can also be evaluated. The probabilities allow the user to place appropriate confidence in the answer as opposed to having an incorrect answer or no answer at all. Note that, depending upon the application, one may choose to report only the object with the highest probability as the nearest neighbor or only those objects whose probabilities exceed a minimum threshold.

Providing probabilistic answers to nearest-neighbor queries is much more difficult than range queries. For range queries, the probability for each object can be determined independent of the other objects—it depends only upon the query and the uncertainty of the object in question. However, for nearest-neighbor queries, the

- *R. Cheng and S. Prabhakar are with the Department of Computer Science, Purdue University, West Lafayette, IN 47906. E-mail: {ckcheng, sunil}@cs.purdue.edu.*
- *D.V. Kalashnikov is with the School of Information and Computer Science, University of California at Irvine, Irvine, CA 92697. E-mail: dvk@ics.uci.edu.*

Manuscript received 17 Jan. 2003; revised 1 July 2003; accepted 28 July 2003. For information on obtaining reprints of this article, please send e-mail to: tkde@computer.org, and reference IEEECS Log Number 118159.

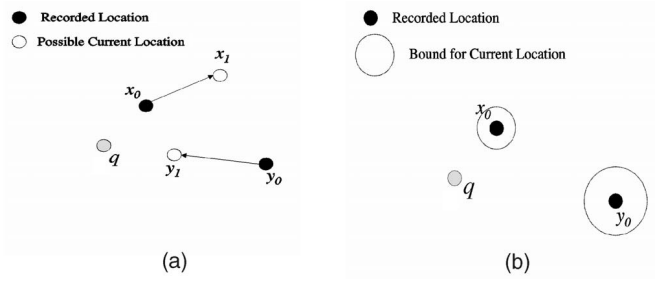


Fig. 1. (a) A nearest-neighbor query for a point q can yield false results by using the data values stored in the database. (b) Imprecision can be used to provide answers with guaranteed certainty.

interplay between objects is critical and the probability that an object is the closest to the query is greatly influenced by the position and uncertainty of the other objects. In this paper, we present a novel technique for providing probabilistic guarantees to answers of nearest-neighbor queries. As an overview, our algorithm first eliminates all the objects that have no chance of being the nearest neighbor. Then, for every object that may be the nearest neighbor, its probability is evaluated by summing up the probability of being the nearest neighbor for all its possible locations. Our solution is generic since it makes no assumption about the nature of movement or uncertainty model used to limit updates from objects. It can thus be applied to any practical object movement model. We illustrate that our algorithm can be easily applied to two of the most important classes of object movement models.

It should be noted that, in contrast to the problem of finding an approximate nearest neighbor wherein accuracy is traded for efficiency, the imprecision in the query answers is inherent in this problem. To the best of our knowledge, the problem of inherent imprecise query processing has only been addressed in [24], where the discussion is limited to the the case of range queries for objects moving in straight lines with known mean speed. We generalize this problem for range queries with a less constrained model of movement and also address the more challenging problem of nearest-neighbor queries which has not been considered earlier.

To sum up, the contributions of this paper are:

- a formal notion of probabilistic nearest-neighbor queries,
- an algorithm for answering probabilistic nearest-neighbor queries under a general model of uncertainty,
- solutions to probabilistic nearest-neighbor queries for two of the most important moving-object models, and
- methods for efficient execution of our algorithms, including the use of index structures and approximation techniques.

The rest of this paper is organized as follows: In Section 2, we describe a general model of uncertainty for moving objects and the concept of probabilistic queries. Section 3 discusses the algorithms for evaluating probabilistic queries under a general uncertainty model. Section 4 studies how to evaluate queries for two popular uncertainty models: line-segment and free-moving uncertainty. Section 5 addresses the issue of computing the answers efficiently with the use of index structures and faster query processing through approximation. In Section 6, we present detailed experiment results. Section 7 discusses related work and Section 8 concludes the paper. Special cases of the solutions are discussed in Appendix A and Appendix B (which can be found on the Computer Society Digital Library at http://computer.org/tkde/archives.htm). The analytical cost of executing probabilistic nearest-neighbor queries is studied in Appendix C (which can be found on the Computer Society Digital Library at http://computer.org/tkde/archives.htm).

2 Uncertainty Model and Probabilistic Queries

In this section, we describe a model of uncertainty for moving objects. This uncertainty model is generic in the sense that it incorporates the paradigm of most applications. Based on this uncertainty model, we introduce the concept of probabilistic range and nearest-neighbor queries.

Several specific models of uncertainty have been proposed. One popular model for uncertainty is that, at any point in time, the location of the object is within a certain distance, d, of its last reported position. If the object moves further than this distance, it reports its new location and possibly alters the distance d to a new value (known to both the object and the server) [24]. A less uncertain model is one in which objects are constrained to move along straight lines (which may correspond to road segments for example). The position of the object at any time is within a certain interval, centered at its last reported position, along the line of movement [24]. Other models include those that have no uncertainty [14], where the exact speed and direction of movement are known. This model requires updates at the server whenever the objects speed or direction change. Another model assumes that the object travels with known velocity along a straight line, but may deviate from this path by a certain distance [19], [23].

For the purpose of our discussion, the exact model of movement of the object is not important. All that is required is that, at the time of execution of the query, the uncertainty information be known for each object. The uncertainty of an object can be characterized as follows:

Definition 1. *An* **uncertainty region** *of an object O_i at time t, denoted by $U_i(t)$, is a closed region such that O_i can be found only inside this region.*

Definition 2. *The* **uncertainty probability density function** *of an object O_i, denoted by $f_i(x, y, t)$, is a probability density function of O_i's location (x, y) at time t, that has a value of 0 outside $U_i(t)$.*

In the above definitions, we assume each object is a point, i.e., its spatial extents are not considered. Also, since $f_i(x, y, t)$ is a probability density function, it has the property that $\int_{U_i(t)} f_i(x, y, t)dxdy = 1$. We do not limit how the uncertainty region evolves over time or what the probability density function of an object is inside the

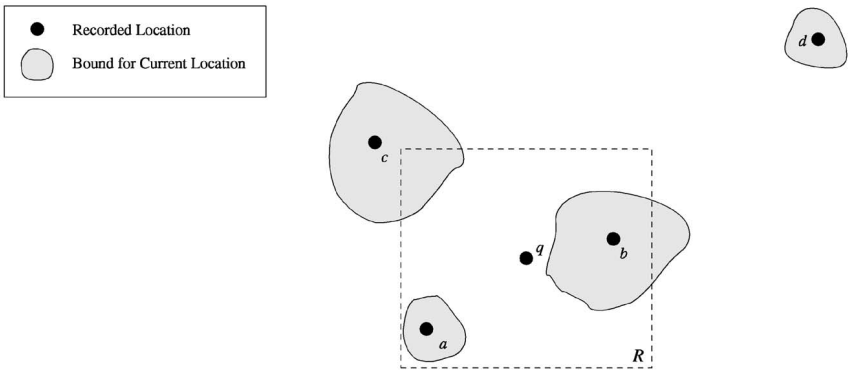


Fig. 2. Example of PRQ and PNNQ.

uncertainty region. The only requirement for the probability density function is that its value is 0 outside the uncertainty region. A trivial probability density function is the uniform density function, which depicts the worst-case or "most uncertain" scenario. Usually, the scope of uncertainty is determined by the recorded location of the moving object, the time elapsed since its last update, and other application-specific assumptions. For example, one may decide that the uncertainty region of an object contains all the points within distance $(t - t_u) \times v$ from its last reported position, where t_u is the time that the reported position was sent and v is the maximum speed of the object. One can also specify that the object location follows the Gaussian distribution inside the uncertainty region.

Based on the above model, different types of location-related queries, such as range queries and nearest-neighbor queries, can be issued. The imprecision of location values imply that some objects may satisfy a query. Therefore, it is natural to assign a probability value to each object that satisfies the query result. In [24], a probabilistic method for capturing the uncertainty information in range queries is presented. Each query returns a set of tuples in the form (O, p) where O is the object and p is the probability that O is in the "range query region" specified by the user. Only the tuples where p is greater than some minimum threshold are returned.

We now generalize their ideas with the definition of a *probabilistic range query*:

Definition 3: Probabilistic Range Query (PRQ). *Given a closed region R and a set of n objects $O_1, O_2, \ldots, O_n$ with uncertainty regions and probability density functions at time t_0, a PRQ returns a set of tuples in the form of (O_i, p_i), where p_i is the nonzero probability that O_i is located inside R at time t_0.*

A probabilistic nearest-neighbor query can be defined in a similar manner:

Definition 4: Probabilistic Nearest-Neighbor Query (PNNQ). *For a set of n objects $O_1, O_2, \ldots, O_n$ with uncertainty regions and probability density functions given at time t_0, a PNNQ for a point q is a query that returns a set of tuples of the form (O_i, p_i), where p_i is the nonzero probability that O_i is the nearest neighbor of q at time t_0.*

Note that the answer to the same probabilistic query executed at two different time instants over the same database can be different even if no updates are received by the database during the time between the two executions because uncertainty regions may change over time.

We conclude this section with a simple example. In Fig. 2, objects a, b, c, d, each with different uncertainty regions (shaded), are being queried. Assume each object has an even chance of being located in its uncertainty region, i.e., $f_a(x, y, t), f_b(x, y, t), f_c(x, y, t), f_d(x, y, t)$ are uniform density functions at any time t. A PRQ (represented by the rectangle R) is invoked at time t_0 to find out which objects are inside R. Object a is always inside the rectangle, so its probability value is 1. Object d is always outside the rectangle, thus it has no chance of being located inside R. $U_b(t_0)$ and $U_c(t_0)$ partially overlap the query rectangle. In this example, the result of the PRQ is: $\{(a, 1), (b, 0.7), (c, 0.4)\}$.

In the same example, a PNNQ is issued at point q at time t_0. We can see that a lot of points in $U_b(t_0)$ are closer to q than points in other uncertainty regions. Moreover, $f_b(x, y, t)$ is a uniform density function over $U_b(t_0)$ and, so, b has a high probability of being the nearest neighbor of q. Object d does not have any chance of being the nearest neighbor since none of the points in $U_d(t_0)$ is closer to q than all other objects. For this example, the result of the PNNQ may be: $\{(a, 0.3), (b, 0.5), (c, 0.2)\}$. We will investigate how these probability values can be obtained in the next section.

3 Evaluating Queries for Imprecise Data

In this section, we examine how PRQ and PNNQ can be answered under the uncertainty model described in the last section. Although the solution to PRQ is trivial compared with PNNQ, understanding the solution to PRQ is useful for understanding how PNNQ is evaluated.

3.1 Evaluation of PRQ

A PRQ returns a set of tuples (O_i, p_i), where p_i is the nonzero probability that object O_i is located in the query region R at time t_0. Let $S = \{O_1, \ldots, O_{|S|}\}$ be the set of all moving objects that have to be considered by the PRQ and let X be the set of tuples returned by the PRQ. The algorithm for evaluating the PRQ at time t_0 is described in Fig. 3, which basically integrates the probability distribution function in the overlapping area of $U_i(t_0)$ and R to compute p_i. In Section 7, we

1. Let S be the set of all moving objects in the database
2. $X \leftarrow \emptyset$
3. **for** $i \leftarrow 1$ **to** $|S|$ **do**
   (a) $A \leftarrow U_i(t_0) \cap R$
   (b) **if** $(A \neq 0)$ **then**
      i. $p_i \leftarrow \int_A f_i(x,y,t_0)dxdy$
      ii. **if** $(p_i \neq 0)$ **then** $X \leftarrow X \cup (O_i, p_i)$
4. **return** X

Fig. 3. PRQ Algorithm.

compare our method with Wolfson et al.'s approach [24] in evaluating a PRQ.

3.2 Evaluation of PNNQ

Processing a PNNQ involves evaluating the probability of each object being closest to a query point. Unlike the case of PRQ, we can no longer determine the probability for an object independent of the other objects. In this section, we present a framework to answer PNNQ for a generic model of uncertainty. Section 4 discusses how this solution framework can be applied easily to two of the most common uncertainty models in Section 4.

Recall that a PNNQ returns a set of tuples (O_i, p_i) for a point q, where p_i denotes the nonzero probability that O_i is the nearest neighbor of q at time t_0. Again, let $S = \{O_1, O_2, \ldots, O_{|S|}\}$ be the set of objects to be considered by q in evaluating the query and let X be the set of tuples returned by the query. The solution presented here consists of four steps: the *projection*, *pruning*, *bounding*, and *evaluation* phases. The first three phases filter out any objects in the database that have no chance of being the nearest neighbor. The last phase, namely, the *evaluation* phase, is the core part of our solution: It computes the probability of being the nearest neighbor for each object that remains after the first three phases:

1. **Projection Phase.** In this phase, the uncertainty region of each moving object is computed based on the uncertainty model used by the application. Figs. 4a and 4b illustrate how this phase works. The last recorded locations of the objects in S are shown in Fig. 4a. The uncertainty regions "projected" onto the object space are shown in Fig. 4b. The shapes of these uncertainty regions are determined by the uncertainty model used, the last recorded location of O_i, the time elapsed since the last location update, and the maximum speeds of the objects.
2. **Pruning Phase**. Consider two uncertainty regions $U_1(t_0)$ and $U_2(t_0)$. If the shortest distance of $U_1(t_0)$ to q is greater than the longest distance of $U_2(t_0)$ to q, we can immediately conclude that O_1 is not an answer to the PNNQ: Even if O_2 is at the location farthest from q in $U_2(t_0)$, O_1 cannot be closer than O_2 to q. Based on this observation, we can eliminate objects from S by the algorithm shown in Fig. 5. The key to this algorithm is to find f, the minimum of the longest distances of the uncertainty regions from q, and eliminate any object with shortest distance to q larger than f. In Section 5.1, we examine a method adopted from the nearest-neighbor search algorithm [17] to find f efficiently.

   After this phase, S contains the (possibly fewer) objects which must be considered by q. This is the minimal set of objects which must be considered by the query since any of them could be the nearest neighbor of q. Fig. 4b illustrates how this phase removes objects that are irrelevant to q from S.
3. **Bounding Phase**. For each element in S, there is no need to examine all portions in the uncertainty region. We only have to look at the regions that are located no farther than f from q. We do this conceptually by drawing a *bounding circle* C of radius f, centered at q. Any portion of the uncertainty region outside C can be ignored. Figs. 4c and 4d illustrate the result of this phase.

   The phases we have just described essentially reduce the number of objects to be evaluated, and derive an upper bound on the range of possible location values, based on the uncertainty regions of the objects and the position of q. We are now ready to present the details of the most important part of our solution—the *evaluation phase*. We will present the algorithm first before explaining how it works.
4. **Evaluation Phase**. Based on S and the bounding circle C, our aim is to calculate, for each object in S, the probability that it is the nearest neighbor of q. The solution is based on the fact that *the probability of an object o being the nearest neighbor with distance r to q is given by the probability of o being at distance r to q times the probability that every other object is at a distance of r or larger from q.* Let $C_q(r)$ denote a circle with center q and radius r. Let $P_i(r)$ be the probability that O_i is located inside $C_q(r)$ and $pr_i(r)$ be the probability density function of r such that O_i is located at the boundary of $C_q(r)$. Fig. 6 presents the algorithm for this phase.

   In order to handle zero uncertainty, i.e., $U_i(t_0)$ is the recorded location of O_i, a special procedure has to be inserted to this algorithm. To simplify discussions, we assume nonzero uncertainty throughout the paper. Appendix A (which can be found on the Computer Society Digital Library at http://computer.org/tkde/archives.htm) discusses in detail how to handle zero uncertainty.

3.2.1 Evaluation of $P_i(r)$ and $pr_i(r)$

As introduced before, $P_i(r)$ is the probability that O_i is located inside the circle $C_q(r)$ (with center q and radius r). The computation of $P_i(r)$ is shown in Fig. 7.

If $r < n_i$, we are assured that $C_q(r)$ cannot cover any part of $U_i(t_0)$, so O_i cannot lie inside $C_q(r)$ and $P_i(r)$ equals 0 (Step 1). On the other hand, if $r > f_i$, then we can be certain that $C_q(r)$ covers all parts of $U_i(t_0)$, i.e., O_i must be inside $C_q(r)$ and $P_i(r)$ equals 1 (Step 2). Steps 3 and 4 returns a nonzero $P_i(r)$ value.

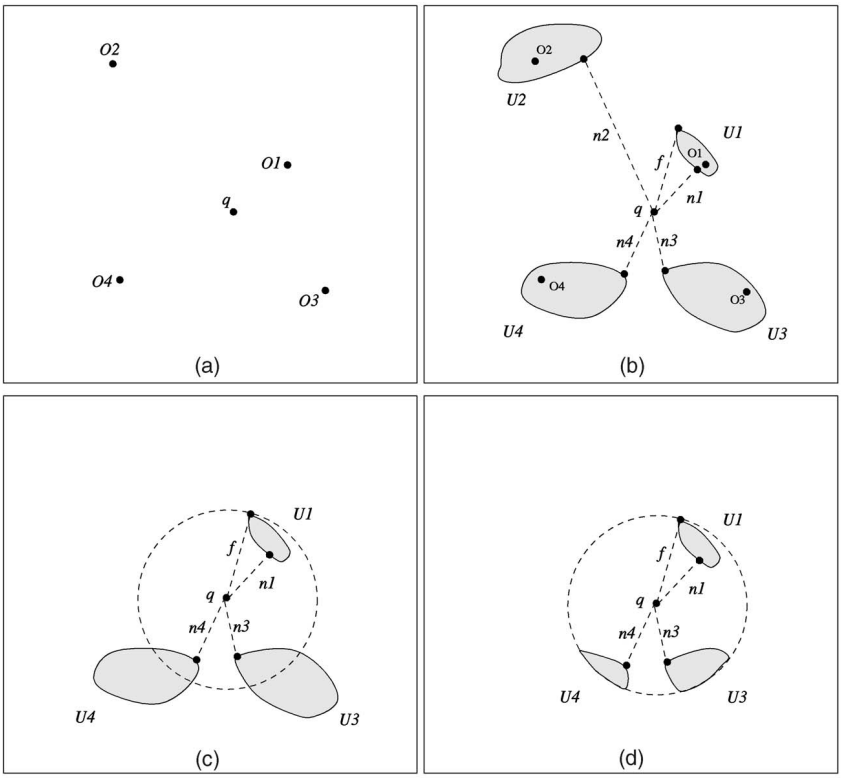


Fig. 4. An example of a PNNQ processing: (a) Locations of objects, (b) uncertainty regions and distances from q, (c) bounding circle, and (d) bounded regions.

The evaluation phase needs to compute $pr_i(r)$, the probability density function of r where O_i lies on an infinitesimally narrow ring of radius r centered at q. If $P_i(r)$ is a differentiable function, then $pr_i(r)$ is the derivative of $P_i(r)$. Note that $pr_i(r)$ is undefined at t_0 if $U_i(t_0)$ is a point (i.e., zero uncertainty) because $P_i(r)$ becomes a step function and the derivative of $P_i(r)$ is undefined at t_0. These subtle details are discussed in Appendix A (which can be found on the Computer Society Digital Library at http://computer.org/tkde/archives.htm).

3.2.2 Evaluation of p_i

We can now explain how p_i is computed. Let $Prob(r)$ denote the probability density function that O_i lies on the boundary of $C_q(r)$ and is the nearest neighbor of q. Then, (1) illustrates the structure of our solution:

1. **for** $i \leftarrow 1$ **to** $|S|$ **do**
   (a) Let n_i be the shortest distance of $U_i(t_0)$ from q
   (b) Let f_i be the longest distance of $U_i(t_0)$ from q
2. $f \leftarrow min_{i=1,\ldots,|S|} f_i$
3. $m \leftarrow |S|$
4. **for** $i \leftarrow 1$ **to** m **do**
   (a) **if** $(n_i > f)$ **then** $S \leftarrow S - O_i$
5. **return** S

Fig. 5. Algorithm for the pruning phase.

$$p_i = \int_{n_i}^{f} Prob(r)\, dr. \tag{1}$$

Equation (1) computes the sum of the probability that O_i is the nearest neighbor over all locations inside $U_i(t_0)$. Recall that n_i represents the shortest distance of $U_i(t_0)$ from q, while f is the radius of the bounding circle, beyond which we do not need to consider. Equation (1) expands $C_q(r)$ with radius n_i to f. Therefore, each point in $U_i(t_0)$ must lie on some circular ring of width dr, center q, and radius r, where $r \in [n_i, f]$. Essentially, we consider all the points in $U_i(t_0)$ that are equidistant from q and evaluate the chance that they are nearest to q.

For each ring, the event that O_i is the nearest neighbor of q occurs when: 1) O_i lies on the ring and 2) O_i is the nearest neighbor of q. Assuming that these two events are independent, we can rewrite (1) in terms of $pr_i(r)$ and $P_k(r)$ (where $k \neq i$):

$$p_i = \int_{n_i}^{f} Prob(O_i \; lies \; on \; the \; boundary \; of \; C_q(r)) \cdot Prob(other \; objects \; lie \; outside \; C_q(r))\, dr \tag{2}$$

$$= \int_{n_i}^{f} p_i(r) \cdot \prod_{k=1 \wedge k \neq i}^{|S|} (1 - P_k(r))\, dr. \tag{3}$$

Observe that each $1 - P_k(r)$ term registers the probability that object O_k (where $k \neq i$) lies at a distance greater than r.

1. $X \leftarrow \emptyset$
2. Sort the elements in S in ascending order of n_i, and rename the sorted elements in S as $O_1, O_2, \ldots, O_{|S|}$
3. $n_{|S|+1} \leftarrow f$
4. **for** $i \leftarrow 1$ **to** $|S|$ **do**
   (a) $p_i \leftarrow 0$
   (b) **for** $j \leftarrow i$ **to** $|S|$ **do**
      i. $p \leftarrow \int_{n_j}^{n_{j+1}} pr_i(r) \cdot \prod_{k=1 \wedge k \neq i}^{j} (1 - P_k(r))\, dr$
      ii. $p_i \leftarrow p_i + p$
   (c) $X \leftarrow X \cup (O_i, p_i)$
5. **return** X

Fig. 6. Algorithm for the evaluation phase.

3.2.3 Efficient Computation of p_i

We can improve the computation time of (3). Note that $P_k(r)$ has a value of 0 if $r \leq n_k$. This means when $r \leq n_k$, $1 - P_k(r)$ is always 1 and O_k has no effect on the computation of p_i. Instead of always considering $|S| - 1$ objects in the $\prod$ term of (3) throughout $[n_i, f]$, we may actually consider fewer objects in some ranges of values. First, we sort the objects according to their shortest distances (n_i) from q. Next, the integration interval $[n_i, f]$ is broken down into a number of intervals with end points defined by n_i. The probability of an object being the nearest neighbor of q is then evaluated for each interval in a manner similar to (3), except that we only consider the objects with nonzero $P_k(r)$. The sum of the probabilities for these intervals is p_i. The final equation for p_i is:

$$p_i = \sum_{j=i}^{|S|} \int_{n_j}^{n_{j+1}} pr_i(r) \cdot \prod_{k=1 \wedge k \neq i}^{j} (1 - P_k(r))\, dr. \qquad (4)$$

Here, we let $n_{|S|+1}$ be f for notational convenience. Instead of considering $|S| - 1$ objects in the $\prod$ term, (4) only handles $j - 1$ objects in interval $[n_j, n_j + 1]$.

Equation (4) is implemented in our evaluation phase. Step 2 sorts the objects in S in ascending order of near distances. Step 3 assigns the value of f to $n_{|S|+1}$. Step 4 executes (4) once for every object O_i in S and puts the tuples (O_i, p_i) into X, which is returned in Step 5.

Example. Fig. 8a shows five objects $O_1, \ldots, O_5$, captured after the bounding phase with uncertainty regions. Fig. 8b shows the result after these objects have been sorted in ascending order of n_i, with the r-axis being the distance of the object from q, and n_6 equals f. The probability p_i of each object O_i being the nearest neighbor of q is the sum of the probability that O_i is the nearest neighbor at each point in the line interval $[n_i, n_6]$. Using (4), p_i is evaluated by considering the subintervals in $[n_i, n_6]$. For example, p_3 is computed by considering the objects in three subintervals: O_1, O_2, O_3 in $[n_3, n_4]$, O_1, O_2, O_3, O_4 in $[n_4, n_5]$, and O_1, O_2, O_3, O_4, O_5 in $[n_5, n_6]$.

1. **if** $(r < n_i)$ **then return** 0
2. **if** $(r > f_i)$ **then return** 1
3. $A \leftarrow U_i(t_0) \cap C_q(r)$
4. **return** $\int_A f_i(x, y, t_0)\, dx dy$

Fig. 7. Computation of $P_i(r)$.

4 Querying with Line-Segment and Free-Moving Uncertainty

In this section, we apply the generic PRQ and PNNQ algorithms presented in the last section to two practical models of uncertainty: line-segment and free-moving uncertainty.

4.1 Line-Segment and Free-Moving Uncertainty Models

We first discuss the two important types of uncertainty based upon the popular models proposed in the literature [24], [19]:

- **Line-segment uncertainty**. For objects that move along straight line paths, the uncertainty at any time is given by a line-segment. The line along which the object is currently moving is known to the database. The center and length of this segment is determined by the exact model of movement used. For example, the length of the segment may be fixed or may vary over time based upon a maximum allowed speed of movement.
- **Free-moving uncertainty**. For objects that are free to move in any direction, the uncertainty at any time is given by a circle. The center and radius of the circle are determined by the last reported location and the exact model of movement used. For example, the radius could be fixed or may be determined by the product of the maximum speed of the object and time since the last update. It may also be a predefined value evaluated by dead-reckoning policies [24]. The center of the circle could be the last reported location or could be determined by the

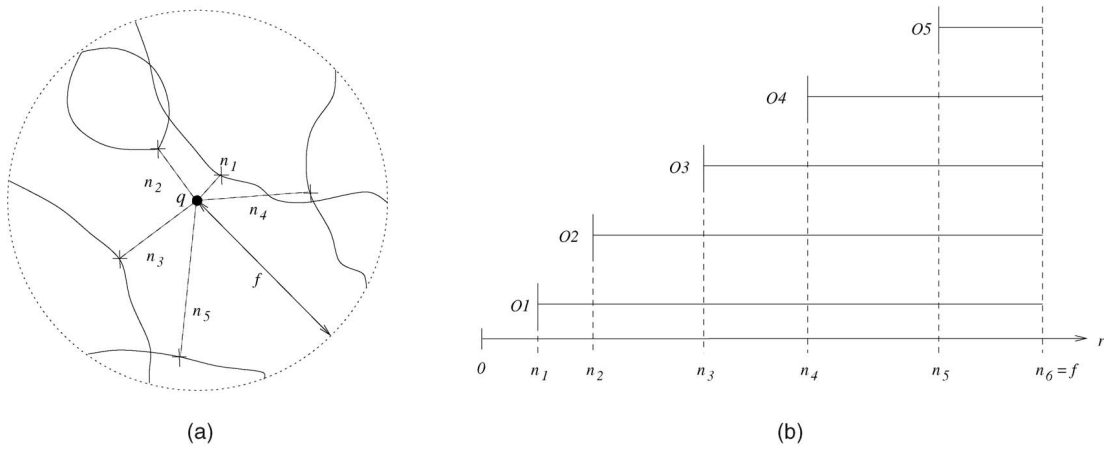


Fig. 8. This example illustrates how the evaluation phase works. (a) Uncertainty regions of five object, and the bounding circle with center q and radius f. (b) O_i sorted in ascending order of n_i.

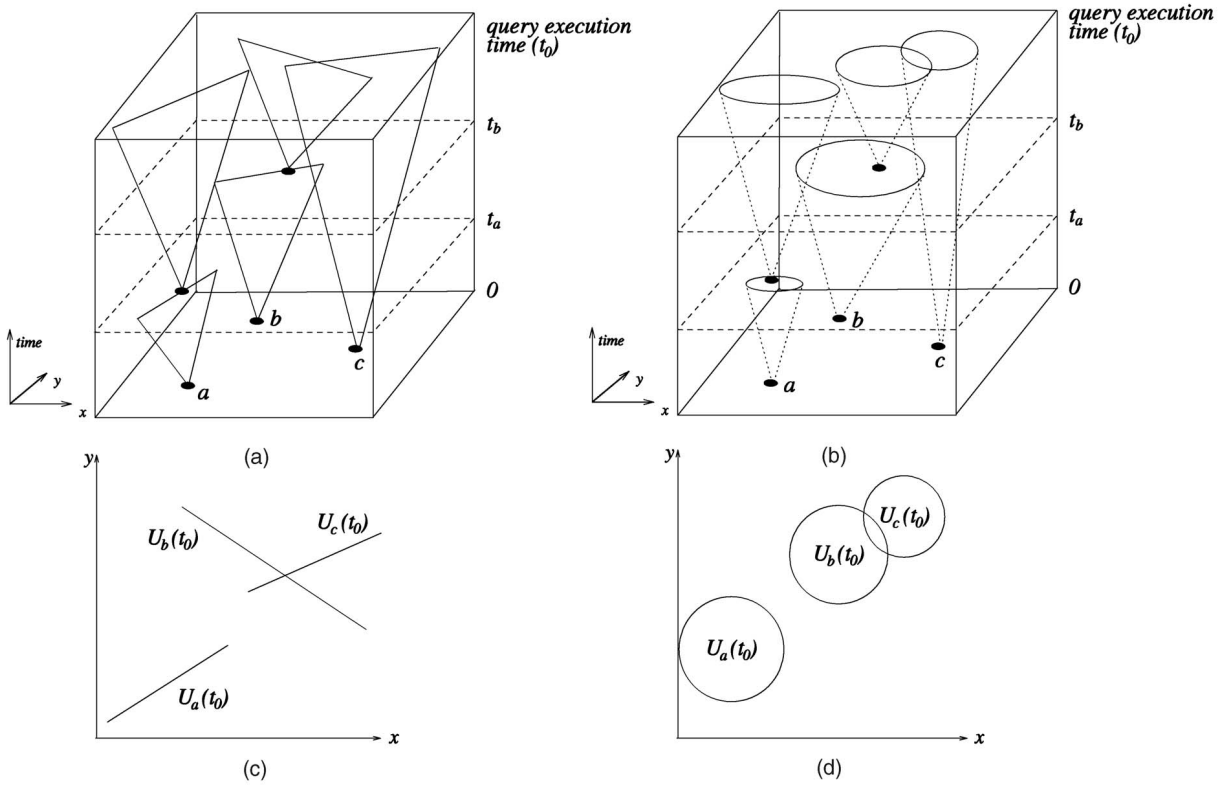


Fig. 9. Example of uncertainty model under the assumption that objects specify a maximum speed with each update. (a) Line-moving objects. (b) The uncertainty regions of objects a, b. (c) Free-moving objects. (d) The uncertainty regions of objects a, b, and c.

last reported location, direction, and speed of movement, and the time since the last update.

From now on, we assume that the uncertainty regions at time $t(U_i(t))$ for object O_i are in the form of either line segments or circles. If $U_i(t)$ is a line segment, then $|U_i(t)|$ is the length of $U_i(t)$; if $U_i(t)$ is a circle, then $|U_i(t)|$ is the area of $U_i(t)$. Furthermore, we assume that an object has the same chance of being located anywhere within $U_i(t)$, i.e., the probability density function $f_i(x, y, t)$ of $U_i(t)$ is a bounded uniform distribution:

$$f_i(x,y,t) = \begin{cases} 1/|U_i(t)| & \text{if } (x,y) \in U_i(t) \\ 0 & \text{otherwise.} \end{cases}$$

The bounded uniform distribution is important in situations when we have no information about the location of the object within the uncertainty region—the worst-case uncertainty. The best guess is then that the object has the same chance of being located in every point in the uncertainty region. This is also the case when probabilistic queries are the most useful—when querying old data under a high level of uncertainty is prone to error. Due to its simplicity and practicability, we will illustrate how to evaluate probabilistic queries using uniform distribution in this section. We note that, however, it is easy to extend our generic solution in Section 3 to other kinds of distributions, such as the normal distribution for dead-reckoning policies [24].

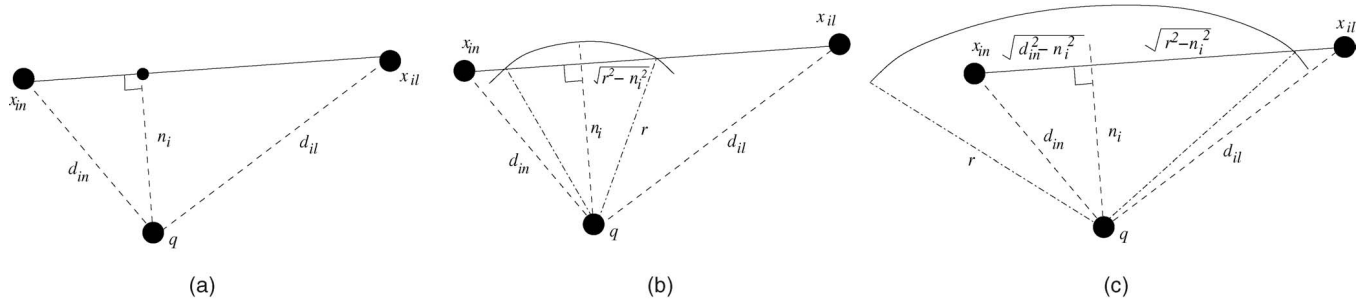


Fig. 10. An example of Case 1 ($n_i < d_{in}$) Line-Segment Uncertainty. (b) Intersection by $C_q(r)$ with radius r, centered at q, such that $n_i \leq r \leq d_{in}$. (c) Intersection by $C_q(r)$ with radius r, centered at q, such that $d_{in} < r < d_{il}$.

Figs. 9a and 9c illustrate the notions of uncertainty for these two models. For the case of line-moving objects, an object reports its current location, line of movement, and maximum speed, S_m. Following this location report, the object is free to move along the reported line of movement at any speed not exceeding the maximum reported speed. Thus, the uncertainty of the object t seconds after the location report is received is given by a line segment centered at the previous reported location, of length equal to $2S_m t$ along the line of movement. Since the next update from the object will provide a new location, line of movement, and maximum speed, the uncertainty of the objects is a line segment with length increasing from one update until the next update is received. For the case of the free moving object, an update reports the current location and a maximum speed. The uncertainty region is a circle with radius $S_m t$ that increases over time until the next update occurs. Figs. 9b and 9b demonstrate that evaluating a probabilistic query is equivalent to looking at the uncertainty regions in Figs. 9a and 9c, respectively, at time t_0, when the query is issued.

In the rest of this section, we will discuss the evaluation of probabilistic queries in line-segment and free-moving uncertainty models. Readers are reminded that our approach is also applicable to other uncertainty models.

4.2 PRQ for Line-Segment and Free-Moving Uncertainty

Answering a PRQ for our line-uncertainty and free-moving uncertainty models is easy. Assume that a PRQ is evaluated at time t_0. First, notice that the probability density function $f_i(x, y, t_0)$ of the uncertainty region $U_i(t_0)$ is uniform and, so, O_i has an equal opportunity of being located anywhere in $U_i(t_0)$. Thus, p_i is simply equal to the fraction of $U_i(t_0)$ that overlaps the query region R. The following two equations derive p_i for line-segment and free-moving uncertainty. They can be used to replace Step 3(b)i of the generic PRQ algorithm shown in Fig. 3.

Line-moving uncertainty. Since $U_i(t_0)$ is a straight line,

$$p_i = \frac{length\ of\ U_i(t_0)\ that\ overlaps\ R}{length\ of\ U_i(t_0)}.$$

Free-moving uncertainty. In this case, $U_i(t_0)$ is a circle,

$$p_i = \frac{Area\ of\ U_i(t_0)\ that\ overlaps\ R}{Area\ of\ U_i(t_0)}.$$

4.3 PNNQ for Line-Segment and Free-Moving Uncertainty

Recall that the PNNQ solution presented in Section 3.2 is generic, i.e., it can be applied to different uncertainty models. In order to evaluate PNNQ for a particular uncertainty model, we need to parametize the generic solution according to the specifications of the uncertainty model. Once all parameters are defined appropriately, the parameterized generic PNNQ solution will correctly evaluate the queries for that particular model.

Suppose a PNNQ is evaluated at time t_0. The parameters that need to be found to adapt the generic PNNQ solution to a particular uncertainty model, for every object O_i, are:

1. $U_i(t_0)$ and $f_i(x, y, t_0)$,
2. n_i and f_i, the shortest and longest distance of $U_i(t_0)$ from q, respectively, and
3. $P_i(r)$ and $pr_i(r)$.

We have already discussed what $U_i(t_0)$ and $f_i(x, y, t_0)$ are for both the line-segment and free-moving uncertainty models. In the rest of this section, we discuss the technical details involved in obtaining n_i, f_i, $P_i(r)$, and $pr_i(r)$ for line-segment and free-moving uncertainty. Also, we use $d(A, B)$ to denote the distance between two points A and B. For clarity, we only illustrate the derivation of $P_i(r)$ and $pr_i(r)$ with the assumption that q does not lie on $U_i(t_0)$ for any object i. In Appendix B (which can be found on the Computer Society Digital Library at http://computer.org/tkde/archives.htm), we will show how to derive $P_i(r)$ and $pr_i(r)$ for the case where q is inside $U_i(t_0)$.

4.3.1 Parametizing Generic PNNQ Solution for Line-Segment Uncertainty

Let x_{in} be the endpoint of the line-segment uncertainty $U_i(t_0)$ with a shorter distance from q, and let this distance be d_{in}. Let x_{il} be the end point (before the bounding phase) of the line-segment uncertainty with a longer distance from q, and let this distance be d_{il}. These parameters are illustrated in Fig. 10a.

Obtaining n_i and f_i. When q does not lie on $U_i(t_0)$, $n_i(t_0)$ is equal to either 1) the perpendicular distance between $U_i(t_0)$ and q (Fig. 10a) or 2) the distance between one of the end points and $U_i(t_0)$ (Fig. 11a). In the former case, $0 < n_i < d_{in}$; in the latter, $0 < n_i = d_{in}$. If q lies on $U_i(t_0)$, then $n_i = 0$. In any case, $f_i = d_{il}$.

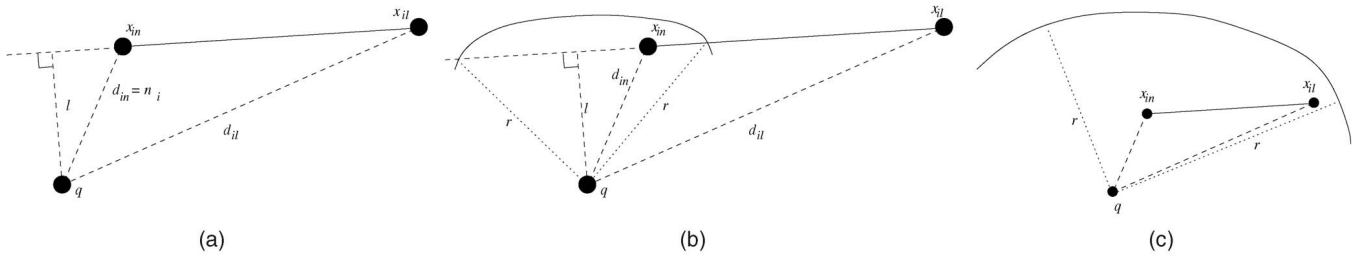


Fig. 11. An example of Case 2 ($n_i = d_{in}$) Line-Segment Uncertainty. (b) Intersection by $C_q(r)$ with radius r, centered at q, such that $d_{in} \leq r \leq d_{il}$. (c) Intersection by $C_q(r)$ with radius r, centered at q, such that $r > d_{il}$.

Obtaining $P_i(r)$ and $pr_i(r)$. Assume that q does not lie on $U_i(t_0)$, i.e., $n_i \neq 0$ (Appendix B (which can be found on the Computer Society Digital Library at http://computer.org/tkde/archives.htm) discusses $P_i(r)$ for $n_i = 0$.). There are two cases to consider: $n_i < d_{in}$ and $n_i = d_{in}$. In both cases, since $f_i(x, y, t_0)$ is a uniform probability density function, $P_i(r)$ is given by:

$$\frac{Length\ of\ U_i(t_0)\ inside\ C_q(r)}{Length\ of\ U_i(t_0)}.$$

Case 1: $n_i < d_{in}$. Fig. 10a illustrates this case. $P_i(r)$ has the following characteristics:

- When $r < n_i$, $U_i(t_0)$ is not contained in $C_q(r)$. Hence, $P_i(r) = 0$.
- When $n_i \leq r \leq d_{in}$, the length of the line-segment uncertainty intersected by $C_q(r)$ is $2\sqrt{r^2 - n_i^2}$, as shown in Fig. 10b. Thus,

$$P_i(r) = \frac{2\sqrt{r^2 - n_i^2}}{d(x_{in}, x_{il})}.$$

- When $d_{in} < r < d_{il}$, as illustrated by Fig. 10c,

$$P_i(r) = \frac{\sqrt{r^2 - n_i^2} + \sqrt{d_{in}^2 - n_i^2}}{d(x_{in}, x_{il})}.$$

- When $r \geq d_{il}$, the line-segment uncertainty of O_i is covered entirely by $C_q(r)$. This implies O_i is ensured to be inside $C_q(r)$ and, thus, $P_i(r)$ equals to 1.

Case 2: $n_i = d_{in}$. Let the perpendicular distance between q and $U_i(t_0)$ be l. An example of this case is shown in Fig. 11a.

- When $r < d_{in}$, $U_i(t_0)$ is not contained in $C_q(r)$. Hence, $P_i(r) = 0$.
- When $d_{in} \leq r \leq d_{il}$, as shown in Fig. 11b,

$$P_i(r) = \frac{\sqrt{r^2 - l^2} - \sqrt{d_{in}^2 - l^2}}{d(x_{in}, x_{il})}.$$

- When $r > d_{il}$, the line-segment uncertainty of O_i is covered entirely by $C_q(r)$. As shown in Fig. 11c, O_i is sure to be inside $C_q(r)$ and, thus, $P_i(r)$ equals to 1.

We now summarize $P_i(r)$ of both cases. We also give the the equation of $pr_i(r)$, which are the derivatives of $P_i(r)$.

Case 1 ($n_i < d_{in}$):

$$P_i(r) = \begin{cases} 0 & r < n_i \\ \frac{2\sqrt{r^2 - n_i^2}}{d(x_{in}, x_{il})} & n_i \leq r \leq d_{in} \\ \frac{\sqrt{r^2 - n_i^2} + \sqrt{d_{in}^2 - n_i^2}}{d(x_{in}, x_{il})} & d_{in} < r < d_{il} \\ 1 & \text{otherwise} \end{cases}$$

$$pr_i(r) = \begin{cases} 0 & r < n_i \text{ or } r \geq d_{il} \\ \frac{2r}{\sqrt{r^2 - n_i^2} d(x_{in}, x_{il})} & n_i \leq r \leq d_{in} \\ \frac{r}{\sqrt{r^2 - n_i^2} d(x_{in}, x_{il})} & d_{in} < r < d_{il}. \end{cases}$$

Case 2 ($n_i = d_{in}$):

$$P_i(r) = \begin{cases} 0 & r < d_{in} \\ \frac{\sqrt{r^2 - l^2} - \sqrt{d_{in}^2 - l^2}}{d(x_{in}, x_{il})} & d_{in} \leq r \leq d_{il} \\ 1 & \text{otherwise} \end{cases}$$

$$pr_i(r) = \begin{cases} 0 & r < d_{in} \text{ or } r > d_{il} \\ \frac{r}{\sqrt{r^2 - l^2} d(x_{in}, x_{il})} & d_{in} \leq r \leq d_{il}. \end{cases}$$

4.3.2 Parametizing Generic PNNQ Solution for Free-Moving Uncertainty

Let $L_i(t_u)$ be the latest recorded location of O_i in the database at time t_u. At time $t_0 > t_u$, the uncertainty region $U_i(t_0)$ is given by a circle with center $L_i(t_u)$ and radius R_i, where $R_i = S_i(t_0 - t_u)$ and S_i is the maximum speed of object O_i. Let $d_i = d(q, L_i(t_u))$. These parameters are illustrated in Fig. 12a.

Obtaining n_i and f_i. Depending on the relative positions of q and $U_i(t_0)$, there are two scenarios:

Case 1: q is outside $U_i(t_0)$. From Fig. 12a, we see that n_i and f_i can be obtained by considering the intersections of $U_i(t_0)$ and the line joining q and $L_i(t_u)$.

Case 2: q is inside $U_i(t_0)$. This situation is illustrated in Fig. 12b. Since object O_i can be anywhere in $U_i(t_0)$, the closest possible location of O_i to q is when O_i coincides with q. $n_i = 0$ in this case.

Obtaining $P_i(r)$ and $pr_i(r)$. The key to derive $P_i(r)$ is to observe that if an object O_i is located inside the $C_q(r)$, it must be situated in the overlapping region of the circles $C_q(r)$ and $U_i(t_0)$. Since $f_i(x, y, t_0)$ is a uniform probability density function, we can deduce that:

$$P_i(r) = \frac{Overlapping\ area\ of\ C_q(r)\ and\ U_i(t_0)}{Area\ of\ U_i(t_0)}. \quad (5)$$

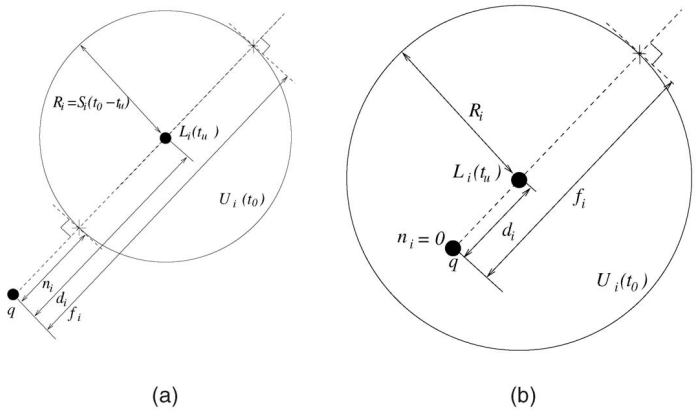


Fig. 12. Free-moving uncertainty. (a) q outside $U_i(t_0)$ and (b) q inside $U_i(t_0)$.

The problem of finding $P_i(r)$ is therefore reduced to the problem of finding the overlapping area of $C_q(r)$ and $U_i(t_0)$. As discussed earlier, the lengths of n_i and f_i depend on whether q is inside or outside $U_i(t_0)$. This results in different overlapping area equations. Here, we only present the derivation of $P_i(r)$ by assuming q is located outside $U_i(t_0)$. A discussion of the derivation of $P_i(r)$ when q is inside $U_i(t_0)$ can be found in Appendix B (which can be found on the Computer Society Digital Library at http://computer.org/tkde/archives.htm).

Fig. 13 shows the overlapping area of $C_q(r)$ and $U_i(t_0)$ when q is outside $U_i(t_0)$. By cosine rule,

$$\theta = arccos\frac{{d_i}^2 + r^2 - R_i^2}{2d_i r} \text{ and } \alpha = arccos\frac{{d_i}^2 + R_i^2 - r^2}{2d_i R_i}.$$

The overlapping area can then be evaluated as follows:

$$\left(\frac{1}{2}r^2(2\theta) - \frac{1}{2}r^2\sin(2\theta)\right) + \left(\frac{1}{2}R_i^2(2\alpha) - \frac{1}{2}R_i^2\sin(2\alpha)\right) \quad (6)$$

$$= r^2\left(\theta - \frac{1}{2}\sin(2\theta)\right) + R_i^2\left(\alpha - \frac{1}{2}\sin(2\alpha)\right). \quad (7)$$

Since the area of $U_i(t_0)$ is πR_i^2, by (5) and (7), we have

$$P_i(r) = \begin{cases} 0 & r < n_i \\ \frac{r^2}{\pi R_i^2}\left(\theta - \frac{1}{2}\sin(2\theta)\right) + \frac{1}{\pi}\left(\alpha - \frac{1}{2}\sin(2\alpha)\right) & n_i \le r \le f_i \\ 1 & \text{otherwise.} \end{cases}$$

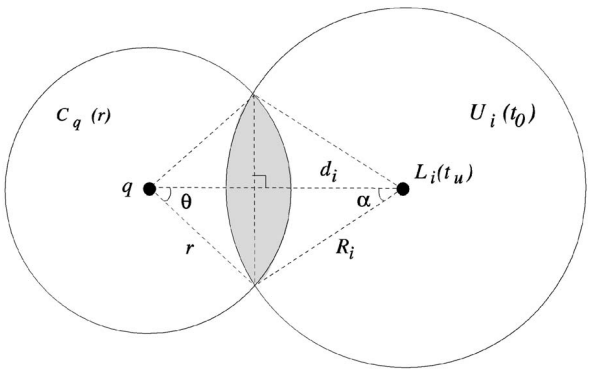


Fig. 13. Intersection of $C_q(r)$ and $U_i(t_0)$.

The probability density function, $pr_i(r)$, is the derivative of $P_i(r)$:

$$pr_i(r) = \begin{cases} 0 & r < n_i \text{ or } r > f_i \\ \frac{2r}{\pi R_i^2}\left(\theta - \frac{1}{2}\sin(2\theta)\right) + \frac{r^2\theta'}{\pi R_i^2}(1 - \cos(2\theta)) + \frac{\alpha'}{\pi}(1 - \cos(2\alpha)) & n_i \le r \le f_i, \end{cases}$$

where $\theta' = \frac{d\theta}{dr} = \frac{1}{2d_i \sin\theta}\left(\frac{d_i^2 - R_i^2}{r^2} - 1\right)$, and $\alpha' = \frac{d\alpha}{dr} = \frac{r}{d_i R_i \sin\alpha}$.

5 Efficient Query Processing

In this section, we address the problem of computing the answers to a PNNQ efficiently. First, we discuss the use of index structures for facilitating the execution of the pruning phase. Then, we discuss how to execute the evaluation phase in an efficient manner.

5.1 Efficient Execution of the Pruning Phase Using VCI

The execution time for the queries is significantly affected by the number of objects that need to be considered. With a large database, it is impractical to evaluate each point for answering the query—this is especially true for the nearest-neighbor queries since, in Step 4 of the evaluation phase, they are quadratic in the number of points considered. It is therefore important to reduce the number of points. As with traditional queries, indexes can be used for this purpose.

The key challenge for any indexing solution for moving objects is efficient updating of the index as the object locations change. Any of the index structures proposed for moving objects can be used for efficiently processing nearest-neighbor queries. We present details for the Velocity-Constrained Index, which is particularly suited for handling uncertainty of free-moving objects. We describe it only briefly here; details can be found in [16].

The only restriction imposed on the movement of objects is that they do not exceed a certain speed. This speed could potentially be adjusted if the object wants to move faster than its current maximum speed. The maximum speeds of all objects are used in the construction of the index. The velocity constrained index (VCI) is an R-tree-like index structure. It differs from the R-tree in that each node has an additional field: v_{max}—the maximum possible speed of movement over all the objects that fall under that node. The index uses the locations of objects at a given point in time, t_0. Construction is similar to the R-tree except that the velocity field is always

adjusted to ensure that it is equal to the largest speed of any object in the subtree. Upon the split of a node, the v_{max} entry is simply copied to the new node. At the leaf level, the maximum velocity of each indexed object is stored (not just the maximum velocity of the leaf node).

As objects move, their locations are noted in the database. However, no change is made to the VCI. When the index is used at a later time, t, to process a query, the actual positions of objects would be different from those recorded in the index. Also, the minimum bounding rectangles (MBR) of the R-tree would not contain these new locations. However, no object under the node in question can move faster than the maximum velocity stored in the node. Thus, if we expand the MBR by $v_{max}(t - t_0)$, then the expanded region is guaranteed to contain all the points under this subtree. Thus, the index can be used without being updated. A range query can be easily performed using this structure. When the search reaches the leaf nodes, the uncertainty of the object is used to compute the probability that it intersects the range.

For nearest-neighbor queries, we use an algorithm similar to the well-known algorithm proposed in [17]. The algorithm uses two measures for each MBR to determine whether or not to search the subtree: *mindist* and *minmax dist*. Given a query point and an MBR of the index structure, the *mindist* between the two is the minimum possible distance between the query point and any other point in the subtree with that MBR. The *minmaxdist* is the minimum distance from the query point for which we can guarantee that at least one point in the subtree must be at this distance or closer. This distance is computed based upon the fact that, for an MBR to be minimal, there must be at least one object touching each of the edges of the MBR. When searching for a nearest neighbor, the algorithm keeps track of the guaranteed minimum distance from the query point. This is given by the smallest value of *minmaxdist* or distance to an actual point seen so far. Any MBR with a *mindist* larger than this distance does not need to be searched further.

This algorithm is easily adapted to work for uncertain data with VCI. Instead of finding the nearest object, the role of the index is now to identify the subset of objects that could possibly be the nearest neighbors of the query point due to their uncertainty regions. This is exactly the set of objects that intersect the circle centered at the query point with radius equal to the shortest maximum distance from the query point. In other words, the index is used to perform the **Pruning Phase** of the probabilistic nearest-neighbor query.

The search algorithm proceeds in exactly the same fashion as the regular algorithm [17], except for the following differences: When it reaches a leaf node, it computes the maximum distance of each object (based upon its uncertainty) from the query. The minimum such value seen so far is called the *pruning distance*. When it encounters an index node, it computes *mindist* and *minmaxdist*. These two are adjusted to take into account the fact that objects may have moved. Thus, *mindist* (*minmaxdist*) is reduced (increased) by $v_{max}(t - t_0)$, where v_{max} is the maximum velocity stored in the node. During the search, each object that could possibly be closer than the pruning distance (based upon the uncertainty in the object) is recorded. At the end of the search, these objects are returned as the pruned set of objects.

5.2 Efficient Execution of the Evaluation Phase

Since the query evaluation algorithms frequently employ costly integration operations, one needs to implement them carefully to optimize the query performance. If the algebraic expressions of $P_i(r)$ and $pr_i(r)$ are simple, we can easily evaluate integrals like those in Step 4(b)(i) of Fig. 6. We may also replace the trigonometric terms of $P_i(r)$ and $pr_i(r)$ with mathematical series such as Taylor's series. We then truncate the series according to the desired degree of accuracy and handle simpler integration expressions.

In general, we have to rely on numeric integration methods to get approximate answers. To integrate a function $f(x)$ over an integration interval $[a, b]$, numeric methods divide the area under the curve of $f(x)$ into small stripes, each with equal width Δ. Then, $\int_a^b f(x)dx$ is equal to the sum of the area of the stripes. The answer accuracy depends on the width of the stripe Δ. One may therefore use Δ to trade off accuracy and execution time. However, choosing a right value of Δ for a query can be difficult. In the algorithms, we evaluate integrals with end points defined by n_is. The interval width of each integral can differ and, if Δ is used to control the accuracy, then all integrals in the algorithm will employ the same value of Δ. A large Δ value may not be accurate for a small integration interval, while a small Δ may make integration using a large interval unnecessarily slow. Thus, Δ should be adaptive to the length of integration interval. For this purpose, we define ε, the inverse of the number of small stripes used by a numeric method:

$$\Delta = integration\ interval\ width\ \cdot \varepsilon = [n_{i+1} - n_i] \cdot \varepsilon. \quad (8)$$

For example, if $\varepsilon = 0.1$, then $\frac{1}{0.1} = 10$ stripes are used by the numeric method. If the integration interval is $[2, 4]$, $\Delta = (4 - 2) \cdot 0.1 = 0.2$. Therefore, ε controls the precision by adjusting the number of stripes and is adaptive to the length of integration interval. We study how to choose ε experimentally in Section 6.

Another method to speed up the evaluation phase at the expense of a lesser degree of accuracy is to reduce the number of candidates after we obtain the circle C. For example, we can set a threshold h and remove any uncertainty interval whose fraction of overlap with C is less than h.

6 Performance Studies

In this section, we present our simulation results. We discuss the experimental setup and important performance results for PNNQ on objects with free-moving uncertainty.

6.1 Simulation Model

Table 1 summarizes parameters used in the simulation. The locations of our moving objects follow the skewed distribution described in [7], a model commonly used in the literature. The data set consists of 100,000 objects—a collection of five normal distributions, each with 20,000 points, distributed in a unit square. The mean values of the normal distribution are

TABLE 1
Parameters Used in the Experiments

Parameter	*Meaning*	*Values*
$\mathcal{D}$	Domain	$[0,1]^2$ (1,000 × 1,000 miles)
N_{obj}	Number of objects	100,000
N_{qry}	Number of queries	100 to 500
$N_{cluster}$	Number of clusters	5
σ_{obj}	Standard deviation of objects	0.05
σ_{qry}	Standard deviation of queries	0.10
V_{max}	Overall maximum speed for any object	0.00007 (250mph)
P_{dir}	Probability to change direction	0.001 to 0.5
P_{speed}	Probability to change speed	0.001 to 0.5
P_{low}	Probability to generate low speed	0.50
P_{med}	Probability to generate medium speed	0.25
P_{high}	Probability to generate high speed	0.25
$\mathcal{T}_{quiet}$	Maximum time without sending update	10 to 1000 sec
d_{quiet}	Maximum distance without sending update	0.0001 to 0.5

uniformly distributed and the standard deviation is 0.05. The query points of the PNNQs are assumed to follow the same distribution with standard deviation 0.1. The total number of queries is between 100 and 500.

The model of object movement is based on the discussions in [16]. The maximum velocities of objects follow the uniform distribution with an overall maximum value of V_{max}, which is set to 0.00007. This is equivalent to an overall maximum velocity of 250 miles an hour with the assumption that the data space represents a square of size 1,000 miles [9]. Each object moves according to its current speed and direction. When the speed is changed, an object is assigned a *slow* speed with a 50 percent chance, a *medium* speed with a 25 percent chance, and a *fast* speed with a 25 percent chance. The *slow*, *medium*, and *fast* classes correspond to speeds ranging from 0 to $\frac{V_{max}}{3}$, from $\frac{V_{max}}{3}$ to $\frac{2V_{max}}{3}$, and from $\frac{2V_{max}}{3}$ to V_{max}, respectively.

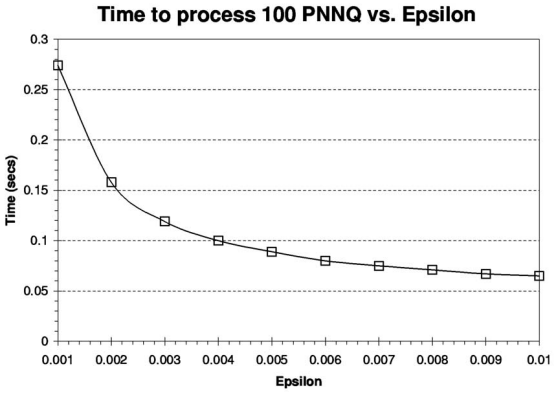


Fig. 14. Execution time versus ε.

An object sends its location update to the server when it moves more than a distance threshold or when the time since its last update exceeds a time threshold. Each update includes the time of update t_{upd}, the object's current location x_{upd} and the maximum speed v_{upd} that the object promises not to exceed. At time instant $t > t_{upd}$, the current location of an object is assumed to be uniformly distributed inside a circle with radius $(t - t_{upd}) \cdot v_{upd}$ and center x_{upd}. An object also sends an update in case it is about to leave its declared uncertainty region. We maintain an in-memory version of VCI index proposed in [16] for moving objects to support the execution of the pruning phase.

6.2 Performance Results

Effect of ε. Our first experiment is to study how the time required to process a PNNQ changes with ε. Recall that ε was defined in Section 5.2 for controlling the precision of a PNNQ result. Fig. 14 shows the execution time (in seconds) needed to process 100 PNNQs with ε ranging from 0.001 to 0.01. As ε increases, the precision decreases and, thus, the query takes less time to compute. The curve drops sharply when ε increases from 0.001 to 0.003. Furthermore, when $\varepsilon > 0.007$, the precision of the result drops significantly due to excessive approximation. Therefore, using a value of 0.003 for ε is a good choice to balance the execution time and precision in our experiments.

PNNQ versus NNQ_old. The next experiment examines the quality of a PNNQ result, compared with a nearest-neighbor query executed on recorded data in the database (thereby referred to as `NNQ_old`). We have given an example (Fig. 1) to illustrate a scenario where obsolete data in the database produces incorrect results. When will this happen and can PNNQ help in this situation?

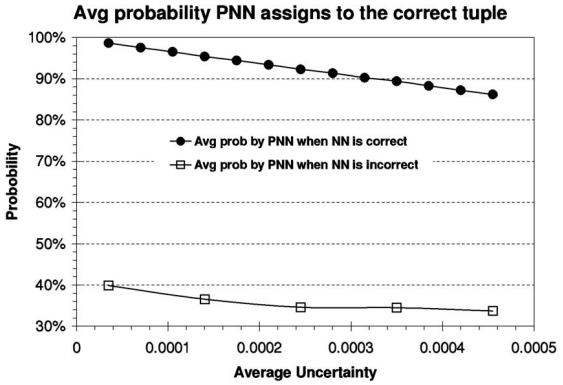


Fig. 15. Quality of PNNQ versus uncertainty.

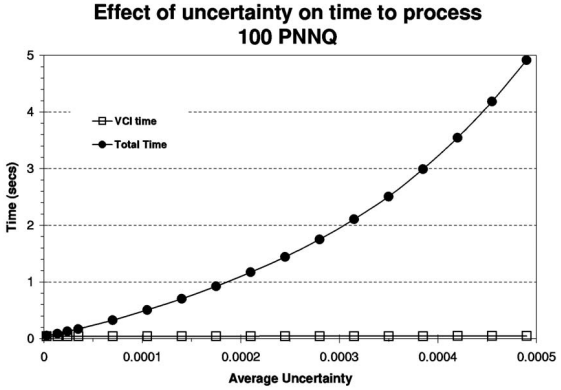


Fig. 16. Execution time versus uncertainty.

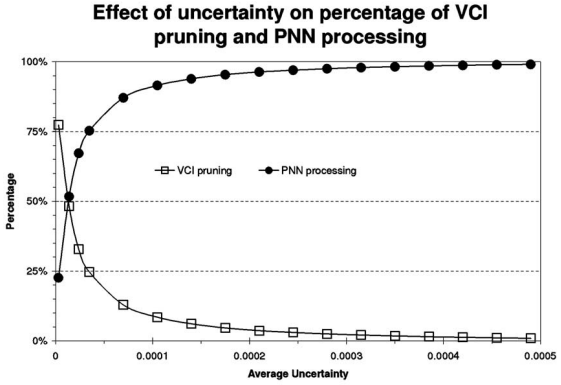


Fig. 17. Breakdown of execution time.

To answer these questions, suppose we also know that the *real* nearest neighbor is z. This true answer, z, can be different from the answer obtained by `NNQ_old`, which only examines the recorded data in the database. However, a PNNQ always assign a nonzero probability to z. To measure how well a PNNQ performs when `NNQ_old` fails to provide the true answer, we define a metric called *IncorrectProb*, which is the probability PNNQ assigns to z when `NNQ_old` fails to identify z as the true nearest neighbor. Intuitively, if *IncorrectProb* is high, it indicates PNNQ is much better than `NNQ_old` in finding out z; a low *IncorrectProb* value implies that PNNQ also fails to identify z as the real nearest neighbor. On the other hand, if `NNQ_old` manages to find z, can PNNQ identify z successfully? Here, we define the metric *CorrectProb* as the probability that PNNQ assigns to z when `NNQ_old` manages to find z. If *CorrectProb* is close to 1, it means PNNQ is almost as good as a `NNQ_old` in finding z.

We study the effect of varying the degree of moving object uncertainty on *IncorrectProb* and *CorrectProb*. Fig. 15 shows the graphs for the mean *IncorrectProb* and *CorrectProb* values of queries as a function of average uncertainty. Here, the average uncertainty refers to the mean radius of the free-moving uncertainty region. As we can see, both curves drop as the average uncertainty increases. This happens because, as uncertainty increases, the cardinality of the answer set increases, with more overlapping in the uncertainty regions. It thus becomes harder to assign a higher probability to a particular candidate. Another important observation is that, under a wide range of average uncertainty, the PNNQ assigns an average probability value between 30 and 40 percent to the real nearest neighbor even when `NNQ_old` fails. When `NNQ_old` succeeds, the PNNQ assigns an average probability value from 80 to 100 percent to the real nearest neighbor. These results reveal the fact that a PNNQ is able to identify the true nearest neighbor with a nontrivial probability value when the result of `NNQ_old` is wrong and is unlikely to miss when `NNQ_old` successfully finds out the true nearest neighbor. In critical situations where it is not allowed to return incorrect nearest neighbor, a PNNQ stands out as a better candidate than `NNQ_old`.

Analysis of PNNQ. The final set of experiments analyzes the time components of a PNNQ. The execution time of a PNNQ is mainly contributed by the pruning and evaluation phases. As explained in Section 5.1, the use of the VCI can effectively improve the time spent on the pruning phase. This is reflected in Fig. 16, which shows the effect of the average uncertainty on the total execution time and pruning time. The pruning time remains relatively inert to the increasing uncertainty. However, the total time still increases with uncertainty. To understand why this happens, let us look at Fig. 17 that records the percentage of the total time spent on the pruning and evaluation phases. As the average uncertainty increases, the fraction of time spent on the evaluation phase increases. This is because the size of the bounding circle increases with uncertainty so that more objects are involved and, subsequently, more integrations with larger intervals have to be handled. This time component dominates the total execution, explaining why the total time increases even when the pruning time is small.

7 Related Work

The uncertainty model described in this paper is based on [24]. In that paper, each moving object is equipped with a facility to detect the deviation of its actual location from the location value in the DBMS. A threshold value, called *uncertainty*, is defined in such a way that if the deviation is larger than it, then an update of the location of that object is sent to the DBMS. The uncertainty value depends on various update policies proposed by the authors, as well as the object movement behavior. An object can move on a predefined route or move freely without following any route. In the former case, a route is a line-spatial object and

the object's motion is characterized by motion vectors in the form (*direction, speed*). The uncertainty is a line segment on the line representing the route. For the latter, a route does not need to be defined and the uncertainty is a circle bounding the possible location of the object.

Another important study on the issues of uncertainty in moving-object database systems is described by Pfoser and Jensen [13]. They introduce a framework to represent moving objects in a relational database and describe the error sources that occur during the sampling of positions of objects: measurement and sampling error. Measurement error is the result of inaccurate instruments, while sampling error occurs because the system only captures the continuous movement of an object periodically, bringing *uncertainty* between two consecutive observations. The authors point out that, in a GPS, sampling error is a more serious problem than measurement error. Assuming the maximum velocity of an object is known, they prove that all possible locations of an object during the time interval between two consecutive observations lie on an *error ellipse*. A complete *trajectory* of any object is obtained by using linear interpolation between two adjacent samples, i.e., a trajectory is approximated by piecewise linear line segments. By using the error ellipse, the authors demonstrate how to process uncertainty range queries for trajectories.

Querying trajectories over uncertain data is also considered in [23]. The uncertainty of object locations is modeled as a 3D cylindrical body around the trajectory. The authors argue that such an uncertainty model facilitates efficient spatial-temporal range querying. The problem of how to improve the speed of range query executions on trajectories using a spatial index was studied in [14]. The work assumes that there exist static objects, called *infrastructures*, that limit the movement of moving objects. In a spatial index such as an R-tree, a line segment is usually approximated by a minimum bounding box. This introduces a lot of "dead-space"—areas where the spatial index is unaware that there is no trajectory at all. As a result, unnecessary searching may be performed on these regions. The utilization of the infrastructure information makes it possible to reduce the searching effort on dead-space. If an infrastructure does not change over time, it implies that none of the moving objects can exist within the space occupied by the infrastructure at any time. Therefore, a preprocessing step can be done to discover which parts of the query window are occupied by the infrastructure. Those parts will be chopped off from the query window, resulting in a smaller query window size and faster index retrieval speed.

Numerous papers have addressed the linguistic issues of moving object database queries. A spatio-temporal query language, called the Future Temporal Logic (FTL), has been proposed in [19] for querying moving object databases. It is a spatio-temporal query that allows future values of dynamic attributes[1] to be queried in a natural way. Due to the inherent uncertain nature of object locations, the authors define the "may" and "must" semantics for FTL: The former semantic specified that the answer to a query has a probability of being incorrect, while the latter one requires the answer to be correct. The paper also describes how to implement FTL on top of an existing relational database. Other works on the specification of spatio-temporal queries include Abdessalem et al.'s paper [1], which uses Pfoser and Jensen's uncertainty model [13] to develop a new set of database operations for answering queries of moving objects. They propose three semantics in the new operations that capture uncertainty:

1. *possibly* semantics, in which the answer to a query certainly contains all correct results, but may also contain some incorrect ones,
2. *surely* semantics, in which the answers are subsets of correct results, and
3. *probably* semantics, in which each answer has a certain probability of being correct.

An example query is "retrieve the location of an object that is *probably* 0.2 miles from a given object." In [23], range queries for trajectories have been proposed, with certain quantifiers defined: 1) A trajectory *sometimes* or *always* satisfies the range query within a time interval specified by the user and 2) a trajectory is satisfied *everywhere* or *somewhere* within the query region. Notice that these three papers take a qualitative approach in the form of specifying the uncertainty in the query by using keywords like "may" and "surely" in the query constructs. We adopt a quantitative presentation of the answers, i.e., probability values to specify the answers to queries.

As far as we know, there is no work addressing a comprehensive discussion of probabilistic methods for specifying and processing moving-object queries as done in this paper. In [24], Wolfson et al. discuss how to process range queries that give probability values as answers. They define the range query as one that finds the objects within a region R and the answers are given by the pairs (o, p), with p being the probability that object o is in R. They assume that the objects move in straightline routes, with mean speed v. The location of every object on its route is modeled as a random variable, with a normal density function; its mean is derived from v and the standard deviation is a function of the uncertainty threshold. The intervals of the route that are inside R are then found out and the probability density function is integrated over these intervals to give the probability p for each object o. In our paper, we do not assume that the mean speed is known. Also, [24] only considers objects traveling on straightline routes and assume normal distributions, while our solutions are capable of handling most practical uncertainty models and are not limited to normal distributions. To the best of our knowledge, we are unaware of any work that addresses the handling of nearest-neighbor queries over uncertain data.

Recently, new types of queries for moving-object databases have been proposed. Lazaridis et al. [10] propose a new query type called *dynamic query*, which is executed continuously by the observer as it moves in space. Since the query results are close in nearby locations, the authors propose techniques for reducing disk I/O.

The problems of indexing and efficient access of spatio-temporal objects have been addressed in [2], [9], [6], [20], [22]. The issues of dynamic attributes indexing were discussed in [18], [21]. A spatial index for trajectories has

1. Dynamic attributes are database attributes that have their values change over time, even if there is no explicit update to the database.

been developed in [14], [15]. In [12], [17], the use of spatial indexes for execution of nearest-neighbor queries is discussed. The processing of nearest-neighbor queries in a moving-object environment is discussed in [8]. Song and Roussopoulos [20] investigate how to execute k-nearest neighbor queries for moving query point efficiently.

8 Conclusions

In this paper, we studied the execution of probabilistic range and nearest-neighbor queries over uncertain data for moving objects. We define a generic model of uncertainty, and then present algorithms for computing these queries for this model. We further illustrate how this solution can be applied to two common models of uncertainty in moving object databases: line-segment and free-moving uncertainty. We studied evaluation of these queries that allow a trade off between execution time and accuracy. The use of indexes for efficient execution of approximate queries over large collections of moving objects is also presented. Our experiments illustrate the effectiveness of probabilistic nearest-neighbor queries over traditional nearest neighbor queries. To the best of our knowledge, with the exception of [24], which addresses probabilistic range queries for objects moving in straight lines with fixed speed, there is no work on probabilistic queries over uncertain data. We address the problem of range queries as well as the more complicated nearest-neighbor queries under a more relaxed model.

There are several interesting avenues for future work. One important research direction is to optimize the time spent on the evaluation phase. An extension is to study how to answer other probabilistic queries like k-nearest neighbor queries and reverse nearest-neighbor queries. Another problem is to answer continuous probabilistic queries over imprecise data efficiently. One important issue is to define the quality metrics for probabilistic queries—how reliable are the results returned by probabilistic queries if we do not know the true results to the queries? Finally, a moving-object database model discussed in this paper belongs to a vast class of sensor-based applications, where sensors are responsible to monitor constantly-changing attributes (e.g., temperature, locations of moving objects). Database readings in these applications are imprecise and it will be interesting to extend the algorithms developed in this paper to apply to such an environment.

Acknowledgments

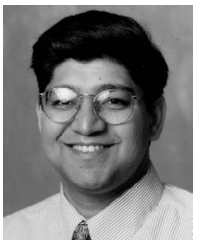
Portions on this work were supported by a US National Science Foundation (NSF) CAREER grant IIS-9985019, NSF grant 0010044-CCR, and NSF grant 9972883. A short preliminary version of this paper appeared in the *Proceedings of the International Conference on Data Engineering*, 2003.

References

[1] T. Abdessalem, J. Moreira, and C. Ribeiro, "Movement Query Operations for Spatio-Temporal Databases," *Proc. 17èmes Journées Bases de Données Avancées,* Oct. 2001.
[2] P. Agarwal, L. Arge, and J. Erickson, "Indexing Moving Points," *Proc. 19th Conf. Principles of Database Systems (PODS),* 2000.
[3] S. Buckingham, "What Is General Packet Radio Service?" http://www.gsmworld.com/technology/gprs/intro.shtml, 2000.
[4] R. Cheng, S. Prabhakar, and D.V. Kalashnikov, "Querying Imprecise Data in Moving Object Environments," *Proc. Int'l Conf. Data Eng. (ICDE '03),* 2003.
[5] P.H. Dana, "Global Positioning System Overview," technical report, Univ. of Texas at Austin, 2000, also available at http://www.colorado.edu/geography/gcraft/notes/gps/gps_f.html.
[6] M. Hadjieleftheriou, G. Kollios, V.J. Tsotras, and D. Gunopulos, "Efficient Indexing of Spatiotemporal Objects," *Proc. Eighth Int'l Conf. Extending Database Technology,* 2002.
[7] D.V. Kalashnikov, S. Prabhakar, S. Hambrusch, and W. Aref, "Efficient Evaluation of Continuous Range Queries on Moving Objects," *Proc. 13th Int'l Conf. and Workshop Database and Expert Systems Applications,* 2002.
[8] G. Kollios, D. Gunopulos, and V.J. Tsotras, "Nearest Neighbour Queries in a Mobile Environment," *Proc. Spatio-Temporal Database Management,* 1999.
[9] G. Kollios, D. Gunopulos, and V.J. Tsotras, "On Indexing Mobile Objects," *Proc. 19th Conf. Principles of Database Systems (PODS),* 1999.
[10] I. Lazaridis, K. Porkaew, and S. Mehrotra, "Dynamic Queries over Mobile Objects," *Proc. Eighth Int'l Conf. Extending Database Technology,* pp. 269-286, 2002.
[11] Navman, iCN630 Vehicle Navigation GPS, http://www.navmanusa.com/land/icn630/, 2003.
[12] A. Papadopoulos and Y. Manolopoulos, "Performance of Nearest Neighbor Queries in R-Trees," *Proc. Int'l Conf. Database Theory,* pp. 394-408, 1997.
[13] D. Pfoser and C.S. Jensen, "Capturing the Uncertainty of Moving-Objects Representations," *Proc. Int'l Conf. Scientific and Statistical Database Manage,emt (SSDBM),* pp. 123-132, 1999.
[14] D. Pfoser and C.S. Jensen, "Querying the Trajectories of On-Line Mobile Objects," *Proc. MobiDE 2001,* pp. 66-73, 2001.
[15] D. Pfoser, C.S. Jensen, and Y. Theodoridis, "Novel Approaches in Query Processing for Moving Object Trajectories," *Proc. 26th Int'l Conf. Very Large Data Bases,* 2000.
[16] S. Prabhakar, Y. Xia, D. Kalashnikov, W. Aref, and S. Hambrusch, "Query Indexing and Velocity Constrained Indexing: Scalable Techniques for Continuous Queries on Moving Objects," *IEEE Trans. Computers,* vol. 51, no. 10, pp. 1124-1140, Oct. 2002.
[17] N. Roussopoulos, S. Kelley, and F. Vincent, "Nearest Neighbor Queries," *Proc. ACM SIGMOD Int'l Conf. Management of Data,* pp. 71-79, 1995.
[18] P.A. Sistla, O. Wolfson, S. Chamberlain, and S. Dao, "Modeling and Querying Moving Objects," *Proc. Int'l Conf. Data Eng.,* pp. 422-432, 1997.
[19] P.A. Sistla, O. Wolfson, S. Chamberlain, and S. Dao, "Querying the Uncertain Position of Moving Objects," *Temporal Databases: Research and Practice,* no. 1399, 1998.
[20] Z. Song and N. Roussopoulos, "k-Nearest Neighbor Search for Moving Query Point," *Proc. Symp. Spatial and Temporal Databases,* pp. 79-96, 2001.
[21] J. Tayeb, O. Ulusoy, and O. Wolfson, "A Quadtree-Based Dynamic Attribute Indexing Method," *The Computer J.,* vol. 41, no. 3, pp. 185-200, 1998.
[22] Y. Theodoridis, T. Sellis, A.N. Papadopoulos, and Y. Manolopoulos, "Specifications for Efficient Indexing in Spatiotemporal Databases," *Proc. 11th Int'l Conf. Scientific and Statistical Database Management,* 1999.
[23] G. Trajcevski, O. Wolfson, F. Zhang, and S. Chamberlain, "The Geometry of Uncertainty in Moving Object Databases," *Proc. Eighth Int'l Conf. Extending Database Technology,* Mar. 2002.
[24] O. Wolfson, P.A. Sistla, S. Chamberlain, and Y. Yesha, "Updating and Querying Databases that Track Mobile Units," *Distributed and Parallel Databases,* vol. 7, no. 3, pp. 257-387, 1999.

Reynold Cheng received the BEng degree in computer engineering and the MPhil degree in computer science from The University of Hong Kong, in 1998 and 2000, respectively. He is currently a PhD student at Purdue University. His research interests are in the areas of moving-object databases, sensor databases, and real-time databases. He is a student member of the IEEE and and a member of the ACM.

Dmitri V. Kalashnikov received the diploma cum laude in applied mathematics and computer science from Moscow State University, Russia, in 1999 and the PhD degree in computer science from Purdue University in 2003. Currently, he is a postdoctoral researcher at the University of California at Irvine. He has received several scholarships, awards, and honors, including an Intel Fellowship and Intel Scholarship. His research interests are in several areas including sensor and moving object databases, data cleaning, and crisis response management. He is a member of the ACM.



Sunil Prabhakar received the BTech degree in electrical engineering from the Indian Institute of Technology, Delhi, in 1990 and the MS and PhD degrees in computer science from the University of California, Santa Barbara in 1998. He is an assistant professor of computer sciences at Purdue University. Dr. Prabhakar's research interests are in large-scale data management, parallel, multimedia and spatio-temporal databases, and digital watermarking. His research has been supported by US National Science Foundation (NSF), CRANE, Indiana State, Microsoft Corp., IBM Corp., and the Center for Education and Research in Information Assurance and Security (CERIAS) at Purdue University. He is a recipient of the NSF CAREER award. He is a member of the editorial board for the *Journal of Database Management* and a senior member of the IEEE and a member of the ACM.