Querying Imprecise Data in Moving Object Environments [*]

Reynold Cheng Dmitri V. Kalashnikov Sunil Prabhakar

Department of Computer Science, Purdue University.

Email: {ckcheng,dvk,sunil}@cs.purdue.edu

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 also study approximate evaluation of these queries to reduce their computation time.

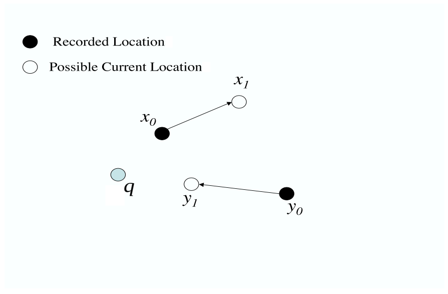
[*]Portions of this work were supported by NSF CAREER grant IIS-9985019, NSF grant 0010044-CCR and NSF grant 9972883. A short, preliminary version of this paper appeared in [3].

Index Terms: Data Uncertainty, Probabilistic Queries, Range and Nearest-Neighbor queries.

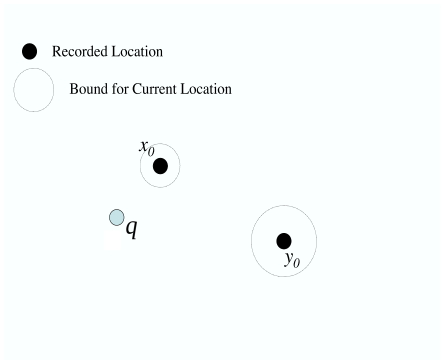
1 Introduction

Systems for continuous monitoring or tracking of mobile objects receive updated locations of objects as they move in space. 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. Figure 1(a) 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 [21]. 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 Figure 1(b). 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



(a)



(b)

Figure 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.

have the possibility of being the nearest neighbor.

The notion of probabilistic answers to queries over uncertain data was introduced in [21] 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 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 method of movement or uncertainty of objects. It can thus be applied to any practical object movement model. We illustrate 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 [21], 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.

2 Uncertainty Model and Probabilistic Queries

In this section, we describe a model of uncertainty for moving objects. This uncertainty model is a generic one, in the sense that it fits into the paradigm of most applications. Based on this uncertainty model, we introduce the concepts 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) [21]. 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 [21]. Other models include those that have no uncertainty [11] 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 [16, 20].

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 location (and uncertainty) 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 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 [21], 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 rectangle *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 non-zero 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 non-zero 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.

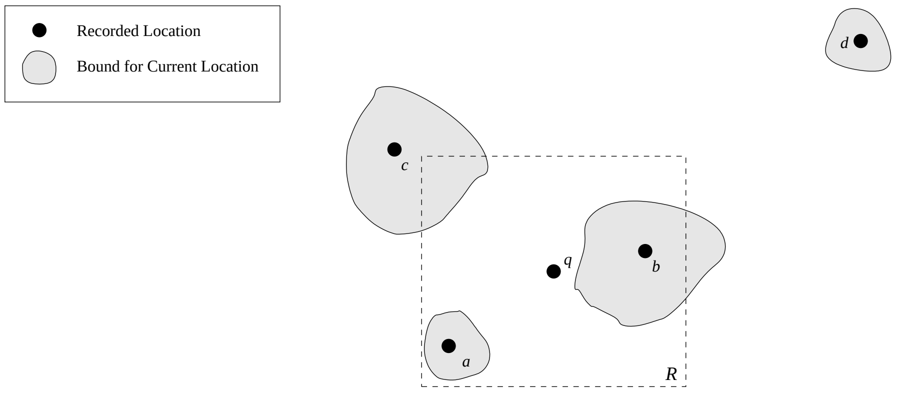


Figure 2: Example of PRQ and PNNQ.

We conclude this section with a simple example. In Figure 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 solutions 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 non-zero probability that object O_i is located in the query rectangle 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 Figure 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 compare our method with Wolfson et al.'s approach [21] 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 indepen-

```
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 $X$
```

Figure 3: PRQ Algorithm.

dent 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 non-zero 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 4 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. Figures 4(a) and 4(b) illustrate how this phase works. The last recorded locations of the objects in S are shown in Figure 4(a). The uncertainty regions "projected" onto the object space are shown in Figure 4(b). The shapes of these uncertainty regions are usually 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.

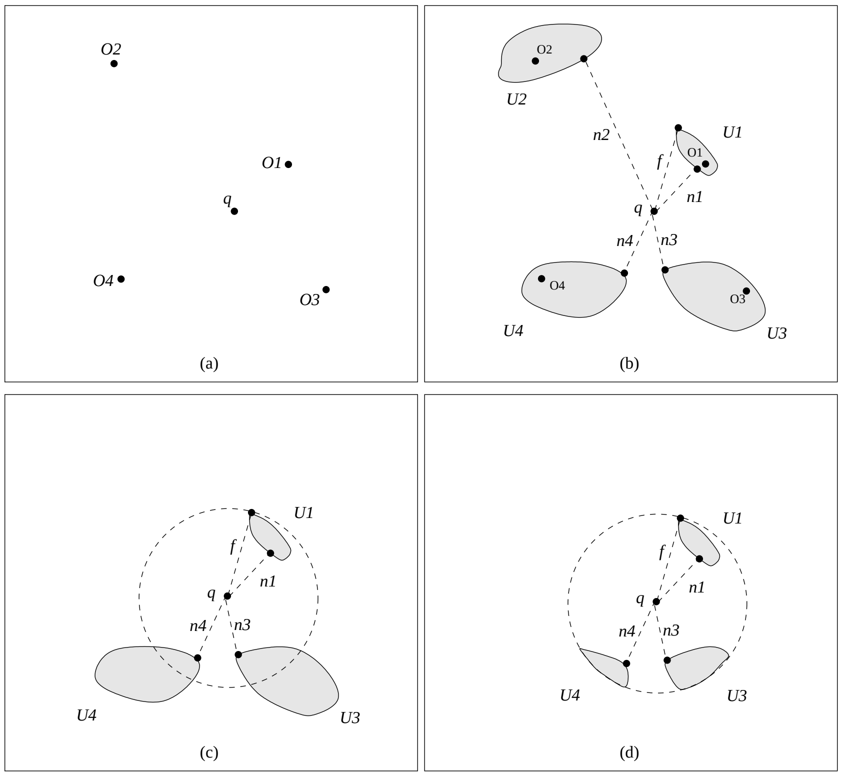


Figure 4: An example of a PNNQ processing: (a) Locations of objects; (b) Uncertainty regions and distances from q; (c) Bounding circle; and (d) Bounded regions.

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 Figure 5. The key of 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 [14] 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. Figure 4(b) illustrates how this phase removes objects that are irrelevant

```
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. $f \leftarrow min_{i=1,\dots,|S|} f_i$
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 $S$
```

Figure 5: Algorithm for the Pruning Phase.

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. Figures 4(c) and (d) 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 $C_q(r)$. Figure 6 presents the algorithm for this phase.

```
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 $X$
```

Figure 6: Algorithm for the Evaluation 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 non-zero uncertainty throughout the paper. Appendix A discusses how to handle zero uncertainty in details.

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 Figure 7.

```
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 $\int_A f_i(x, y, t_0)\, dxdy$
```

Figure 7: Computation of $P_i(r)$.

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 non-zero $P_i(r)$ value.

The evaluation phase needs another parameter called $pr_i(r)$, a 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.

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 Equation 1 illustrates the structure of our solution:

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

The correctness of Equation 1 depends on whether it can correctly consider the probability that O_i is the nearest-neighbor for every location inside $U_i(t_0)$, and then sum up all those probability values. 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 Equation 1 in terms of $pr_i(r)$ and $P_k(r)$ (where $k \neq i$):

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

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

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.

Efficient Computation of p_i We can improve the computation time of Equation 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 Equation 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 way similar to Equation 3, except that we only consider the objects with non-zero $P_k(r)$. The sum of the probabilities for these intervals is p_i. The final equation for p_i is:

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

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

Equation 4 is implemented in our evaluation phase algorithm. Step 2 sorts the objects in S in ascending order of the near distances. Step 3 assigns the value of f to $n_{|S|+1}$. Step 4 executes Equation 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 Let us use an example to illustrate how the evaluation phase works. Figure 8(a) shows 5 objects $O_1, \ldots, O_5$, captured after the bounding phase with uncertainty regions is shown. Figure 8(b) 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 $[n_i, n_6]$.

We now show how p_3 is evaluated. Equation 4 tells us that p_3 is evaluated by applying an integration on $[n_3, n_6]$. Since objects are sorted according to n_i, we do not need to consider all 5 objects throughout $[n_3, n_6]$; Instead, we break down $[n_3, n_6]$ into 3 smaller intervals, and consider (possibly) fewer objects in each small interval: 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

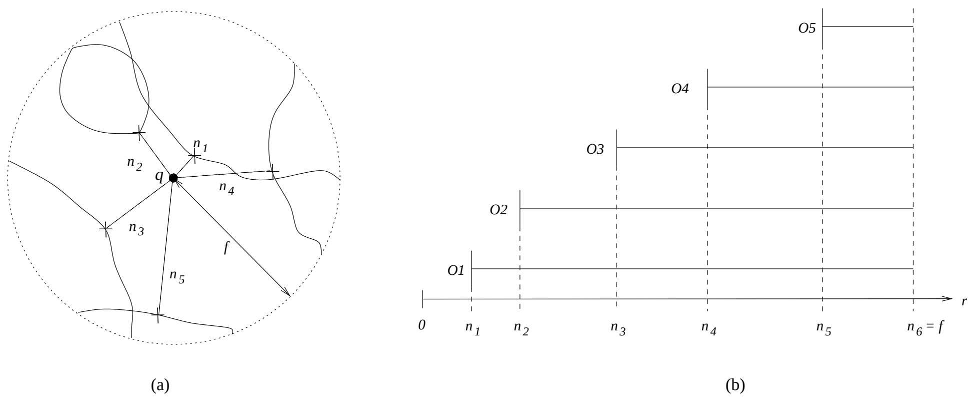


Figure 8: This example illustrates how the evaluation phase works. (a) Uncertainty regions of 5 objects, and the bounding circle with center q and radius f. (b) O_i sorted in ascending order of n_i.

O_1, O_2, O_3, O_4, O_5 in $[n_5, n_6]$.

Let us examine how p_3 is evaluated in interval $[n_3, n_4]$. For O_3 to be the nearest neighbor of q in $[n_3, n_4]$, we require that (1) O_3 is inside $[n_3, n_4]$, and (2) O_1 and O_2 are farther than O_3 to q. The first condition is captured by the term $pr_i(r)\ dr$, while the second condition is met by the term $\prod_{k=1 \wedge k \neq i}^{j}(1 - P_k(r))$ of Equation 4. Therefore, probability that O_3 is the nearest neighbor in $[n_3, n_4]$ is $\int_{n_3}^{n_4} pr_3(r) \cdot (1 - P_1(r)) \cdot (1 - P_2(r))\ dr$.

The probability values are evaluated similarly in $[n_4, n_5]$ and $[n_5, n_6]$. The final value of p_3 is equal to the sum of the probability values over the 3 intervals:

$$\begin{aligned} p_3 &= \int_{n_3}^{n_4} pr_3(r) \cdot (1 - P_1(r)) \cdot (1 - P_2(r))\ dr \\ &+ \int_{n_4}^{n_5} pr_3(r) \cdot (1 - P_1(r)) \cdot (1 - P_2(r)) \cdot (1 - P_4(r))\ dr \\ &+ \int_{n_5}^{n_6} pr_3(r) \cdot (1 - P_1(r)) \cdot (1 - P_2(r)) \cdot (1 - P_4(r)) \cdot (1 - P_5(r))\ dr \end{aligned}$$

We can see that p_3 is evaluated based on the expressions of $pr_i(r)$ and $P_i(r)$. We will talk about how $pr_i(r)$ and $P_i(r)$ can be found for two common uncertainty models in Section 4. We then discuss how to efficiently evaluate PNNQ in Section 5.

4 Querying with Line-Segment and Free-Moving Uncertainty

In this section, we demonstrate how the PRQ and PNNQ algorithms presented in the last section can be adapted to two practical models of uncertainty: line-segment and free-moving uncertainty.

4.1 Line-Segment and Free-Moving Uncertainty Models

We now present two of the most important types of uncertainty based upon the popular models proposed in the literature [21, 16]:

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 pre-defined value evaluated by dead-reckoning policies [21]. The center of the circle could be the last reported location or could be determined by the 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 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 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 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 [21].

Figures 9(a) and 9(c) 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 $2S_mt$ 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_mt that increases over time, until the next update occurs. Figures 9(b) and 9(d) demonstrate that evaluating a probabilistic query is equivalent to looking at the uncertainty regions in Figures 9(a) and 9(c), 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.

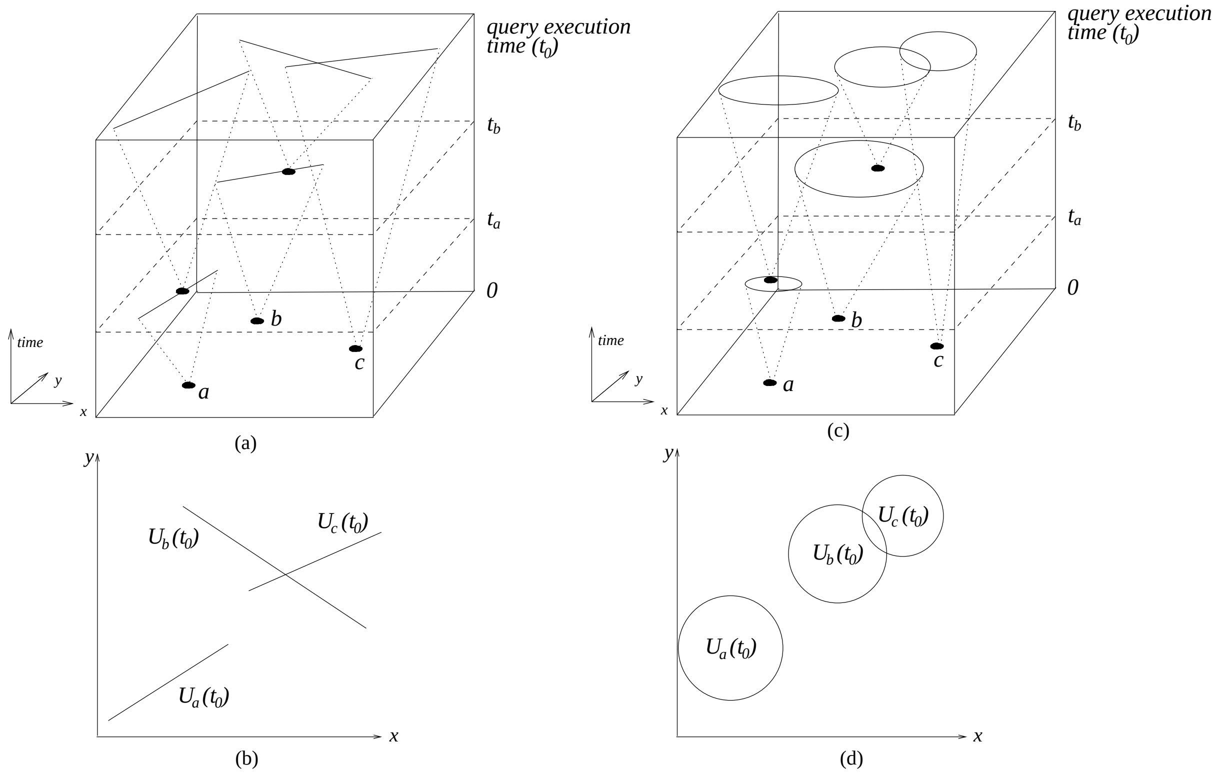


Figure 9: Example of uncertainty model under the assumption that objects specify a maximum speed with each update. (a) Line-moving objects. (b) The uncertainty regions of objects *a*, *b*, and *c*. (c) Free-moving objects. (d) The uncertainty regions of objects *a*, *b* and *c*.

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 uncertainty region $U_i(t_0)$ is uniform, and so O_i has equal opportunity of locating anywhere in $U_i(t_0)$. Thus p_i is simply equal to the fraction of $U_i(t_0)$ that overlaps the query rectangle 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 Section 3.1.

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

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

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

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

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;
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 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, we will show how to derive $P_i(r)$ and $pr_i(r)$ for the case where q is inside $U_i(t_0)$. f

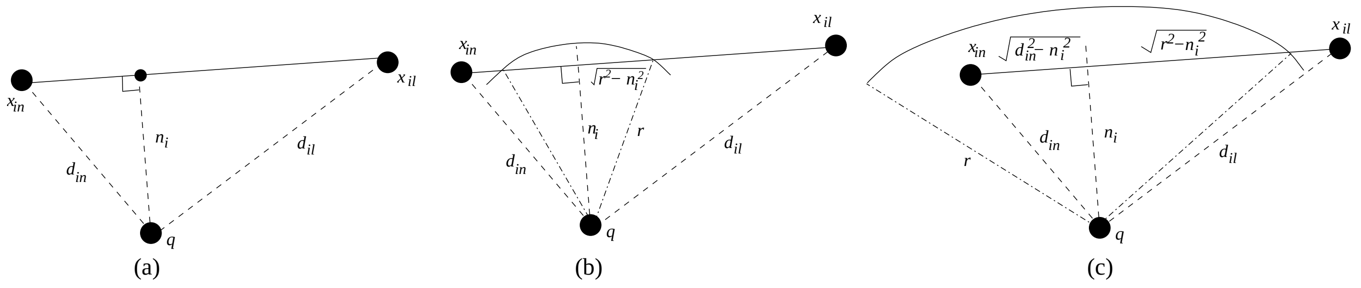


Figure 10: An example of Case 1 ($n_i < d_{in}$) Line-Segment Uncertainty. (b) Intersection by $C_q(r)$ with radius r, centered at q, such that $n_i \le r \le d_{in}$. (c) Intersection by $C_q(r)$ with radius r, centered at q, such that $d_{in} < r < d_{il}$.

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 Figure 10(a).

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 (Figure 10(a)), or (2) the distance between one of the end points and $U_i(t_0)$ (Figure 12(a)). 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, $f_i = d_{il}$.

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 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:

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

Case 1: $n_i < d_{in}$**.** Figure 10(a) 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 Figure 10(b). 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 Figure 10(c), $P_i(r) = \frac{\sqrt{r^2-n_i^2}+\sqrt{d_{in}^2-n_i^2}}{d(x_{in},x_{il})}$.
- 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.

The shape of the resulting $P_i(r)$ in this case is shown in Figure 11(a). Note that the graph of $P_i(r)$ is composed of two quadratic pieces of a piecewise curve.

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 Figure 12(a).

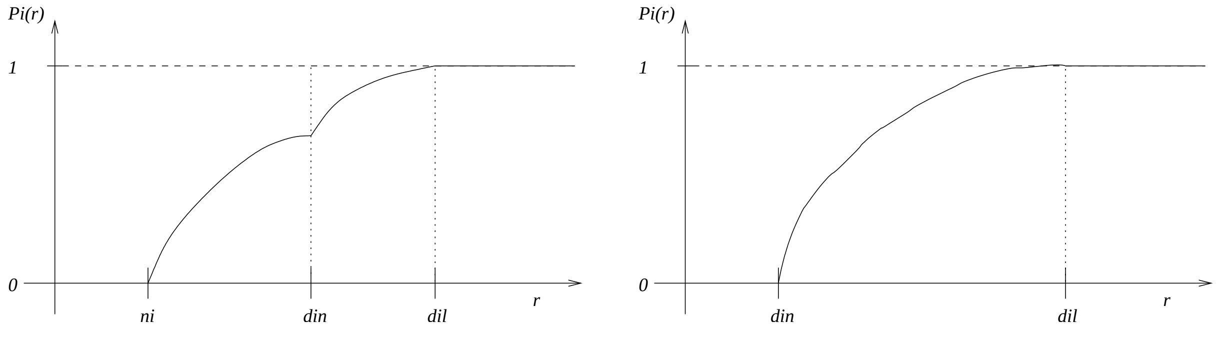


Figure 11: $P_i(r)$ of (a) Case 1 and (b) Case 2.

- 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 Figure 12(b), $P_i(r) = \frac{\sqrt{r^2-l^2}-\sqrt{d_{in}^2-l^2}}{d(x_{in},x_{il})}$.
- When $r > d_{il}$, the line-segment uncertainty of O_i is covered entirely by $C_q(r)$. As shown in Figure 12(c), O_i is sure to be inside $C_q(r)$, and thus $P_i(r)$ equals to 1.

The shape of this $P_i(r)$ is illustrated in Figure 11(b). We can observe that it is only composed of 1 quadratic curve, as opposed to case 1, which contains 2 quadratic pieces of a piecewise curve.

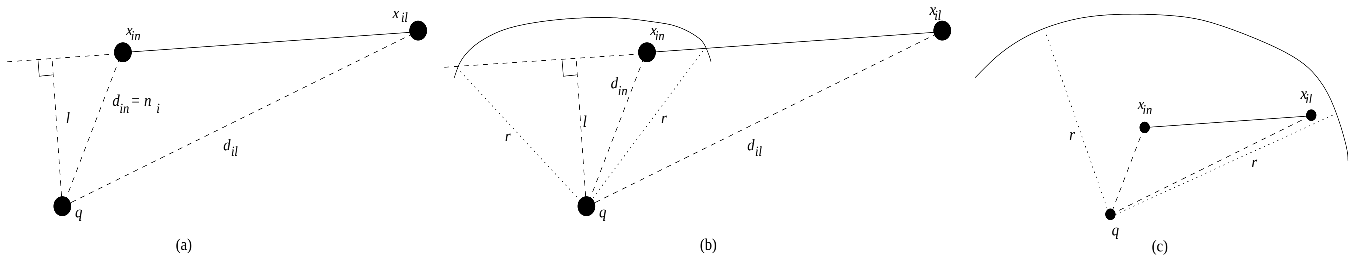


Figure 12: An example of Case 2 ($n_i = d_{in}$) Line-Segment Uncertainty. (b) Intersection by $C_q(r)$ with radius r, centered at q, such that $d_{in} \le r \le d_{il}$. (c) Intersection by $C_q(r)$ with radius r, centered at q, such that $r > d_{il}$.

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 ($n_i < d_{in}$):

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

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

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

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 Figure 13(a).

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 Figure 13(a), 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 13(b). 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.

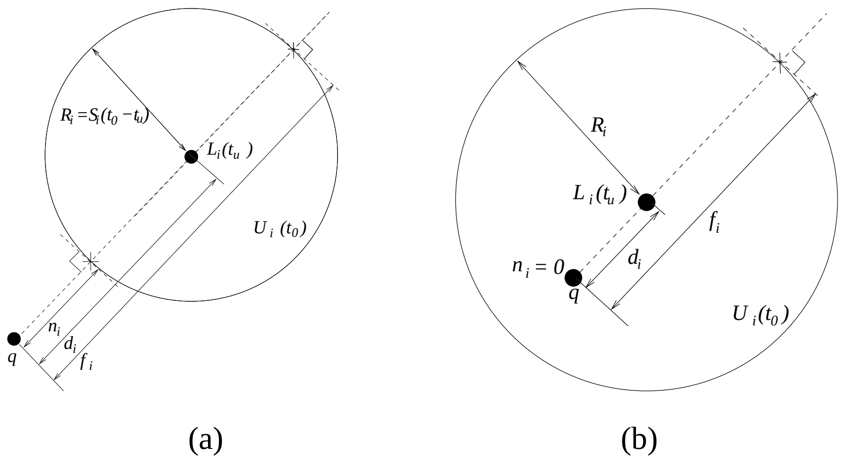


Figure 13: Free-Moving Uncertainty. (a) q outside $U_i(t_0)$, and (b) q inside $U_i(t_0)$.

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

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

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.

Figure 14 shows the overlapping area of $C_q(r)$ and $U_i(t_0)$ when q is outside $U_i(t_0)$. By cosine

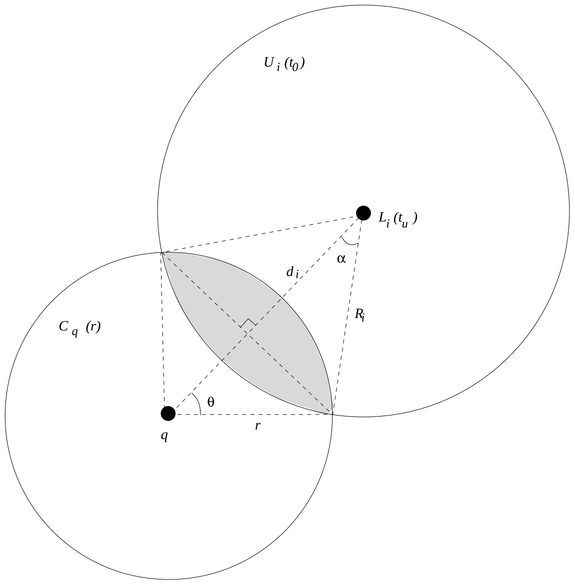


Figure 14: Intersection of $C_q(r)$ and $U_i(t_0)$

rule,

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

The overlapping area can then be evaluated as follows:

$$\begin{aligned} & (\frac{1}{2}r^2(2\theta) - \frac{1}{2}r^2\sin(2\theta)) + (\frac{1}{2}R_i^2(2\alpha) - \frac{1}{2}R_i^2\sin(2\alpha)) && (6) \\ = \;& r^2(\theta - \frac{1}{2}\sin(2\theta)) + R_i^2(\alpha - \frac{1}{2}\sin(2\alpha)) && (7) \end{aligned}$$

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

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

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

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

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

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 [13].

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 sub-tree. 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 sub-tree. Thus the index can be used without being updated. A range query can easily be 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 [14]. The algorithm uses two measures for each MBR to determine whether or not to search the sub-tree: *mindist* and *minmaxdist*. 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 sub-tree 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 sub-tree 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 [14], 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 need 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 evaluate integrals like those in Step 4(b)(i) of Figure 6 easily. 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_i's. 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:

$$\Delta = \textit{integration interval width} \cdot \varepsilon = [n_{i+1} - n_i] \cdot \varepsilon \tag{8}$$

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 have done some sensitivity experiments on our simulation data to decide a good value of ε that ensures both precision and efficiency.

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 Experimental Results

In this section we study the performance of the proposed approach for PNN queries. Since real data are not available, all tests are done on synthetic data. The synthetic data distribution is similar to the skewed distribution of [5], model of object movement are similar to that of [13]. Such data distribution and model of object movement are very common in the literature. The dataset used consists of 100,000 objects composed of a collection of 5 normal distributions each with 20,000 objects. The mean values for the normal distribution are uniformly distributed, and the standard deviation is 0.05 (the points are all in the unit square). The centers of PNN queries are also assumed to follow the same distribution but with a standard deviation of 0.1. The total number of queries is varied between 100 and 500 in our experimentation.

The maximum velocities of objects follow the uniform distribution with an overall maximum value of V_{max}. Note that in [13] a Zipf distribution was used (instead of the uniform) making average object speed there smaller and higher speed cases are typically harder to handle.

For most experiments overall maximum value of V_{max} was set to 0.00007 – if we assume that the data space represents a square of size 1000 miles (as in [7]), this corresponds to an overall maximum velocity of 250 miles an hour. Objects move according to their current speeds and directions. There are parameters that control how often each object changes its speed and direction of movement. When the speed is changed, new speed can be generated according to three categories: slow (generated 50% of time), medium (25%), and fast (25%). Slow category corresponds to speeds from 0 to $\frac{V_{max}}{3}$, medium to from $\frac{V_{max}}{3}$ to $\frac{2V_{max}}{3}$, and fast from $\frac{2V_{max}}{3}$ to V_{max}. Objects send updates to the server when they move more than a certain distance specified as a system threshold and when the time since the last update exceed a certain threshold. Each update include time of update t_{upd}, objects current location x_{upd} and the maximum speed v_{upd} object promises not to exceed, by which server determines the uncertainty region of that object. At later time instant t, the current location of the objects is assumed to be uniformly distributed anywhere inside the circle with radius $(t - t_{upd}) \cdot v_{upd}$ around x. Object also send an update in case it about to leave its declared uncertainty region. While there are many simulation parameters mentioned above, combination of them control average uncertainty among all objects, which in turn determines the performance of PNN queries. We plot average uncertainty as x-axis of many of our graphs. For the rest of the graphs the parameters were chosen such that reasonable average uncertainty is achieved.

We maintain an in-memory version of VCI index proposed in [13] on moving objects.

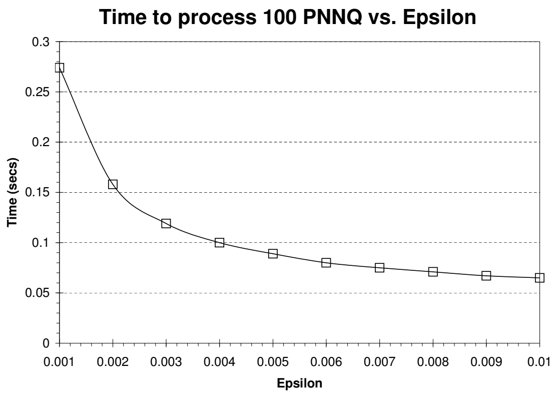


Figure 15:

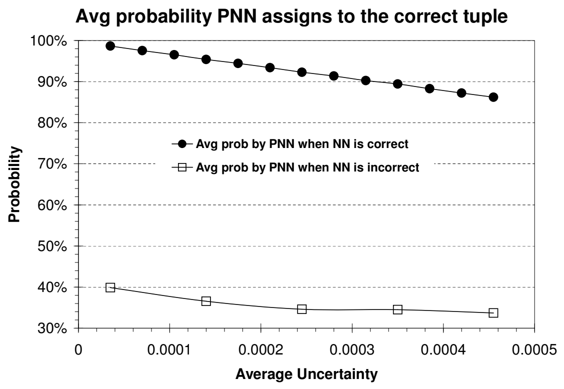


Figure 16:

We begin with an evaluation of time needed to process PNN query as function of parameter ε which control the number of integration steps. Figure 15 shows the time in seconds needed to process 100 of PNN queries where ε is varied from 0.001 to 0.01. From the figure we can observe a clear trade-off between the quality of calculated result and the time needed to process the queries. Value 0.001 corresponds to high precision and value 0.01 to low precision. As epsilon becomes larger and precision requirements lower the queries take less time to compute.

As was noted in the previous sections, the nearest neighbor algorithm on old data might produce incorrect results. Unlike NN algorithm, PNN algorithm always produce correct result by constructing candidate set and giving probabilities that each candidate can be the true nearest neighbor to the query point. Figure 16 shows the average probability PNN algorithms assigns to the real nearest neighbor in two cases (i) when NN algorithm on old data guesses the real nearest neighbor correctly; and (ii) when NN on old data guesses incorrectly. Both curves predictably decreases as average uncertainty increases. This is so because as uncertainty increases the cardinality of candidate set tends to increase and because with higher uncertainty it becomes harder to give a preference to a particular candidate. Figure 16 shows that for wide range of uncertainties, ranged from very low to high, PNN algorithm assigns probability between 30% and 40% to the real nearest neighbor even when NN algorithm on old data makes mistake. This figure also shows that if NN algorithm guesses correctly then PNN algorithm will assign high probability from 80% to 100% to the real nearest neighbor.

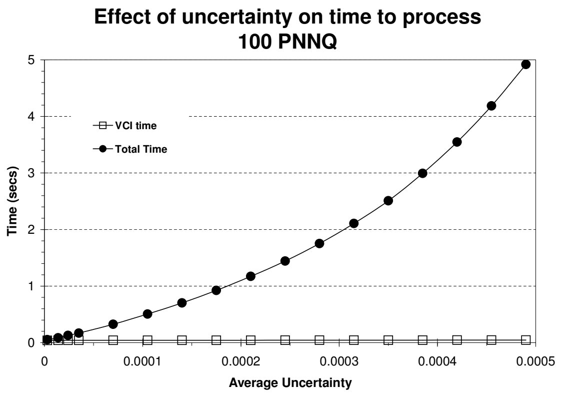


Figure 17:

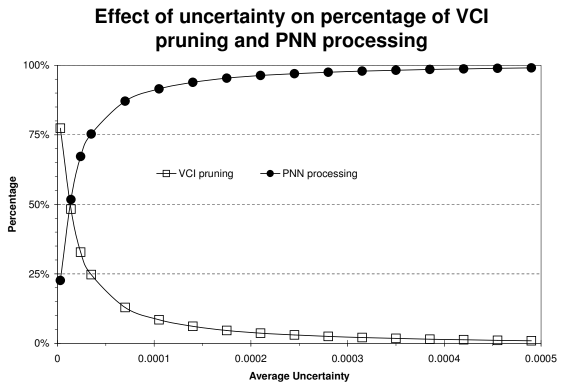


Figure 18:

7 Related Work

The uncertainty model described in this paper is based on [21]. 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 et al. [10]. 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 [20]. 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 [11]. 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 pre-processing 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 [16] 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 et al.'s uncertainty model [10] 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 [20], range queries for trajectories

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

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; (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 [21], 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 straight-line 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. The query region of a range query in our paper is a rectangle for simplifying discussions, although our methods can be extended to query regions of any shape. Also, the authors only consider the objects traveling on straight-line routes, while our solutions are capable of handling most practical uncertainty models. 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. [8] 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, 7, 4, 17, 19]. The issues of dynamic attributes indexing were discussed in [15, 18]. A spatial index for trajectories has been developed in [11, 12]. In [9, 14], 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 [6]. Song et al. [17] 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 allows 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. To the best of our knowledge, with the exception of [21] which addresses probabilistic range queries for objects moving in a 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.

References

[1] T. Abdessalem, J. Moreira, and C. Ribeiro. Movement query operations for spatio-temporel databases. In *Proc. 17èmes Journées Bases de Données Avancées*, Agadir, Maroc, Oct 2001.

[2] P. Agarwal, L. Arge, and J. Erickson. Indexing moving points. In *Proc. of the 19th Conf. on Principles of Database Systems (PODS)*, 2000.

[3] R. Cheng, S. Prabhakar, and D. V. Kalashnikov. Querying imprecise data in moving object environments. In *Proceedings of the International Conference on Data Engineering (ICDE'03)*, 2003.

[4] M. Hadjieleftheriou, G. Kollios, V. J. Tsotras, and D. Gunopulos. Efficient indexing of spatiotemporal objects. In *Proc. of 8th Intl. Conf. on Extending Database Technology*, pages 251–268. 2002.

[5] D. V. Kalashnikov, S. Prabhakar, S. Hambrusch, and W. Aref. Efficient evaluation of continuous range queries on moving objects. In *DEXA 2002, Proc. of the 13th International Conference and Workshop on Database and Expert Systems Applications*, Aix en Provence, France, September 2–6 2002.

[6] G. Kollios, D. Gunopulos, and V. J. Tsotras. Nearest neighbour queries in a mobile environment. In *Proc. of Spatio-Temporal Database Management*, 1999.

[7] G. Kollios, D. Gunopulos, and V. J. Tsotras. On indexing mobile objects. In *Proc. of the 19th Conf. on Principles of Database Systems (PODS)*, 1999.

[8] I. Lazaridis, K. Porkaew, and S. Mehrotra. Dynamic queries over mobile objects. In *Proc. of 8th Intl. Conf. on Extending Database Technology*, pages 269–286, 2002.

[9] A. Papadopoulos and Y. Manolopoulos. Performance of nearest neighbor queries in r-trees. In *ICDT 1997*, pages 394–408, 1997.

[10] D. Pfoser and C. S. Jensen. Capturing the uncertainty of moving-objects representations. In *Proc. of the SSDBM Conference*, pages 123–132, 1999.

[11] D. Pfoser and C. S. Jensen. Querying the trajectories of on-line mobile objects. In *MobiDE 2001*, pages 66–73, 2001.

[12] D. Pfoser, C. S. Jensen, and Y. Theodoridis. Novel approaches in query processing for moving object trajectories. In *VLDB 2000, Proceedings of 26th International Conference on Very Large Data Bases*, pages 395–406, 2000.

[13] 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 Transactions on Computers*, 51(10):1124–1140, October 2002.

[14] N. Roussopoulos, S. Kelley, and F. Vincent. Nearest neighbor queries. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pages 71–79, San Jose, CA, 1995.

[15] P. A. Sistla, O. Wolfson, S. Chamberlain, and S. Dao. Modeling and querying moving objects. In *ICDE*, pages 422–432, 1997.

[16] P. A. Sistla, O. Wolfson, S. Chamberlain, and S. Dao. Querying the uncertain position of moving objects. In *Temporal Databases: Research and Practice*, number 1399, pages 310–337. 1998.

[17] Z. Song and N. Roussopoulos. *k*-nearest neighbor search for moving query point. In *Proc. of Sym. on Spatial and Temporal Databases*, pages 79–96, 2001.

[18] J. Tayeb, Ö. Ulusoy, and O. Wolfson. A quadtree-based dynamic attribute indexing method. *The Computer Journal*, 41(3):185–200, 1998.

[19] Y. Theodoridis, T. Sellis, A. N. Papadopoulos, and Y. Manolopoulos. Specifications for efficient indexing in spatiotemporal databases. In *Proc. of the SSDBM Conference*, 1999.

[20] G. Trajcevski, O. Wolfson, F. Zhang, and S. Chamberlain. The geometry of uncertainty in moving object databases. In *EDBT, 8th International Conference on Extending Database Technology*, pages 233–250. Springer, March 2002.

[21] O. Wolfson, P. A. Sistla, S. Chamberlain, and Y. Yesha. Updating and querying databases that track mobile units. *Distributed and Parallel Databases*, 7(3):257–387, 1999.

Appendix A: Objects with Zero Uncertainty

An object is said to have *zero uncertainty* at time t if it has no uncertainty at time t, i.e., $U_i(t)$ is simply the last recorded location of O_i. Zero uncertainty occurs in objects where their locations do not change with time e.g., infrastructures. In this appendix we discuss how to change our probabilistic query solutions for these kind of objects.

A.1 Evaluation of PRQ for Zero Uncertainty Objects

Step 3(a) of the PRQ algorithm in Section 3.1 evaluates the overlapping area of $U_i(t_0)$ and R. If $U_i(t_0)$ is a point, then it always has zero overlapping area with R. To handle this special case, we present the modified PRQ algorithm in Figure 19.

```
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) if $U_i(t_0)$ is a point then
        i. if $R$ contains $U_i(t_0)$ then $X \leftarrow (O_i, 1)$
        ii. continue Step 3
    (b) $A \leftarrow U_i(t_0) \cap R$
    (c) 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 $X$
```

Figure 19: Modified PRQ algorithm for Handling Zero Uncertainty.

The only change that we make to the original algorithm is the addition of Step 3(a), where we handle $U_i(t_0)$ separately when it is a zero uncertainty region. If $U_i(t_0)$ is inside R, we add $(O_i, 1)$ to X. Otherwise, p_i must be 0 and we do not include O_i into X.

A.2 Evaluation of PNNQ for Zero Uncertainty Objects

Recall that $P_i(r)$ is the probability that $U_i(t_0)$ lies in $C_q(r)$. When $U_i(t_0)$ is a point, $C_q(r)$ either contains $U_i(t_0)$ or does not contain $U_i(t_0)$, and we have:

$$P_i(r) = \begin{cases} 0 & r < d(q, U_i(t_0)) \\ 1 & otherwise \end{cases}$$

Since $P_i(r)$ becomes a step function, its derivative, $pr_i(r)$, is undefined at $r = d(q, U_i(t_0))$. Therefore, we cannot use $pr_i(r)$ in Step 4 of the evaluation phase of the PNNQ algorithm. The PNNQ algorithm that also handles zero uncertainty is shown in Figure 20.

Observe that Step 4(a) is inserted to the original PNNQ algorithm to find the probability that an object with zero uncertainty is the nearest neighbor of q. To determine this probability, Step 4(a) evaluates the probability that all other points are farther to q than $U_i(t_0)$, i.e.,

$$p_i = \prod_{j=1 \wedge j \neq i}^{|S|} (1 - Pr_j(d(q, U_i(t_0))))$$

There are situations when two or more objects have zero uncertainty, and their locations coincide. If this happens, they share the same probability of being the nearest neighbor of q. This is catered by a counter called *samept* in the above algorithm, where it counts the number of objects that have their point uncertainty regions coincide. The final probability is thus equal to $p_i/samept$.

Appendix B: Query Point Inside an Uncertainty Region

In Section 4, we derive $P_i(r)$ and $pr_i(r)$ for both line-segment and free-moving uncertainty, assuming that the query point q is outside $U_i(t_0)$. We will now derive $P_i(r)$ and its derivative $pr_i(r)$ for the case when q is inside $U_i(t_0)$, which can be a line segment or a circle.

B.1 Query Point Inside a Line-Segment Uncertainty Region

When q lies on $U_i(t_0)$, $n_i = 0$, as shown in Figure 21(a). $P_i(r)$ has the following characteristics:

- When $r \leq d_{in}$, the length of the line-segment uncertainty intersected by $C_q(r)$ is simply $2r$ as shown in Figure 21(b). Thus $P_i(r) = \frac{2r}{d(x_{in}, x_{il})}$.

- When $d_{in} < r < d_{il}$, as illustrated by Figure 21(c), $P_i(r) = \frac{r+d_{in}}{d(x_{in},x_{il})}$.
- 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.

The following is a summary of $P_i(r)$ and $pr_i(r)$ for this case:

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

B.2 Query Point Inside a Free-Moving Uncertainty Region

As illustrated in Figure 13(b), we have $n_i = 0$. Also, the overlapping area of $C_q(r)$ and $U_i(t_0)$ when $r \leq R_i - d_i$ is simply $C_q(r)$ itself. Therefore, when $r \leq R_i - d_i$, we have:

$$P_i(r) = \frac{\mathit{Area\ of}\, C_q(r)}{\mathit{Area\ of}\, U_i(t_0)} = \frac{\pi r^2}{\pi R_i^2} = \frac{r^2}{R_i^2}$$

When $r > R_i - d_i$, $C_q(r)$ is not totally contained inside $U_i(t_0)$. The overlapping area of $C_q(r)$ and $U_i(t_0)$ has the same form as Equation 7, so does $P_i(r)$.

The overall formula of $P_i(r)$ is:

$$P_i(r) = \begin{cases} \frac{r^2}{R_i^2} & r \leq R_i - d_i \\ \frac{r^2}{\pi R_i^2}(\theta - \frac{1}{2}\sin(2\theta)) + \frac{1}{\pi}(\alpha - \frac{1}{2}\sin(2\alpha)) & R_i - d_i < r \leq f_i \\ 1 & \text{otherwise} \end{cases}$$

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

Recall that $pr_i(r)$ is the derivative of $P_i(r)$:

$$pr_i(r) = \begin{cases} \frac{2r}{R^2} & r \leq R_i - d_i \\ \frac{2r}{\pi R_i^2}(\theta - \frac{1}{2}\sin(2\theta)) + \frac{r^2\theta'}{\pi R_i^2}(1-\cos(2\theta)) + \frac{\alpha'}{\pi}(1-\cos(2\alpha)) & R_i - d_i \leq r \leq f_i \\ 0 & \text{otherwise} \end{cases}$$

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

```
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) if $U_i(t_0)$ is a point then
           i. $p_i \leftarrow 1$
           ii. $samept \leftarrow 1$
           iii. for $j \leftarrow 1$ to $|S|$ do
                  A. if $j \neq i$ then
                       I. if $U_j(t_0) \neq U_i(t_0)$ then $p_i \leftarrow p_i \cdot (1 - P_j(d(q, U_i(t_0))))$
                       II. else $samept \leftarrow samept + 1$
           iv. $X \leftarrow X \cup (O_i, p_i/samept)$
           v. continue Step 4
     (b) $p_i \leftarrow 0$
     (c) 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. $p_i \leftarrow p_i + p$
     (d) $X \leftarrow X \cup (O_i, p_i)$
5. return $X$
```

Figure 20: Modified Evaluation Phase for Handling Zero Uncertainty.

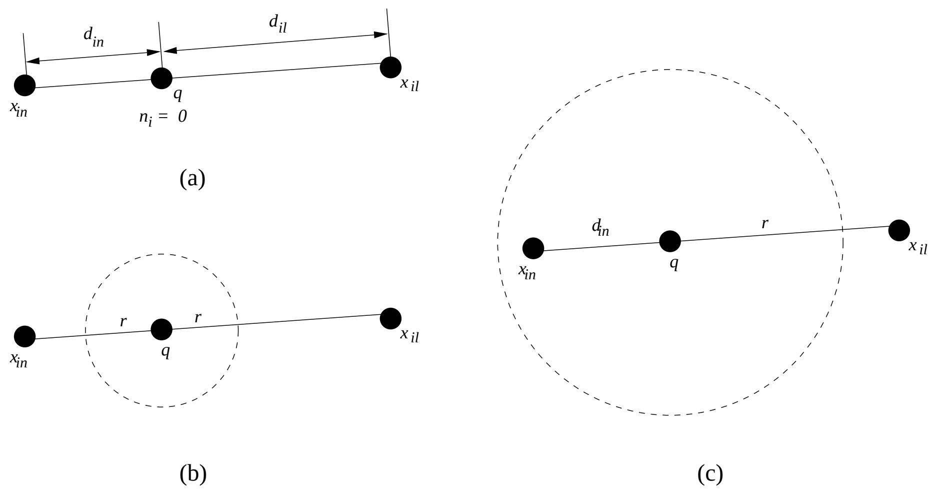


Figure 21: Line Moving Uncertainty for $n_i = 0$. (a) An example of this case. x_{in} is the end point of the line-segment uncertainty that yields a shorter distance to q than another end point x_{il}. (b) Portion of the line-segment uncertainty intersected by $C_q(r)$ with radius r, centered at q, such that $r \leq d_{in}$. (c) Portion of the line-segment uncertainty intersected by $C_q(r)$ with radius r, centered at q, such that $d_{in} < r < d_{il}$.