Reverse-Nearest Neighbor Queries on Uncertain Moving Object Trajectories

Tobias Emrich[1], Hans-Peter Kriegel[1], Nikos Mamoulis[2], Johannes Niedermayer[1], Matthias Renz[1], and Andreas Züfle[1]

[1] Institute for Informatics, Ludwig-Maximilians-Universität München, Germany
{emrich,kriegel,niedermayer,renz,zuefle}@dbs.ifi.lmu.de
[2] Department of Computer Science, University of Hong Kong
nikos@cs.hku.hk

Abstract. Reverse nearest neighbor (RNN) queries in spatial and spatio-temporal databases have received significant attention in the database research community over the last decade. A reverse nearest neighbor (RNN) query finds the objects having a given query object as its nearest neighbor. RNN queries find applications in data mining, marketing analysis, and decision making. Most previous research on RNN queries over trajectory databases assume that the data are certain. In realistic scenarios, however, trajectories are inherently uncertain due to measurement errors or time-discretized sampling. In this paper, we study RNN queries in databases of uncertain trajectories. We propose two types of RNN queries based on a well established model for uncertain spatial temporal data based on stochastic processes, namely the Markov model. To the best of our knowledge our work is the first to consider RNN queries on uncertain trajectory databases in accordance with the possible worlds semantics. We include an extensive experimental evaluation on both real and synthetic data sets to verify our theoretical results.

1 Introduction

The widespread use of smartphones and other mobile or stationary devices equipped with RFID, GPS and related sensing capabilities made the collection and analysis of spatio-temporal data at a very large scale possible. A wide range of applications benefit from analyzing such data, such as environmental monitoring, weather forecasting, rescue management, Geographic Information Systems, and traffic monitoring. In the past, however, research focused mostly on *certain* trajectory data, assuming that the position of a moving object is known precisely at each point in time without any uncertainty. In reality, though, due to physical limitations of sensing devices, discretization errors, and missing measurements, trajectory data have different degrees of *uncertainty*: GPS and RFID measurements introduce uncertainty in the position of an uncertain object. Furthermore, as RFID sensors are usually set up at a certain position, an RFID-based location tracker will only be activated if an object passes near its sensor. Between two consecutive sensor measurements the position of the object remains unknown. This problem of incomplete observations in time is a general problem of trajectory databases, and does not only appear in RFID applications, but also in geo-social networks where users can share trajectories[3], as decreasing the frequency of data collection can increase both storage efficiency and the battery lifetime of GPS devices. The problem of missing measurements also appears in well-known datasets published for research purposes such as T-Drive [28] and GeoLife [29]. As a consequence, it is important to find solutions for deducting the unknown and therefore uncertain positions of objects in-between discrete observations. The most straightforward solution for deducting a position between consecutive measurements would be linear interpolation. However, linear interpolation can cause impossible trajectories, such as a bike driving through a lake. Other solutions such as computing the shortest path between consecutive locations produce valid results, but do not provide probabilities for quantifying the quality of the result.

In this paper, we consider a historical database $\mathcal{D}$ of uncertain moving object trajectories. Each of the stored uncertain trajectories consists of a set of observations given at a some (but not all) timesteps in the past. An intuitive way to model such data is by describing it as a time-dependent random variable, i.e., a stochastic process. In this research, we model uncertain objects by a first-order Markov chain. It has been shown recently that even a first-order Markov chain, if augmented with additional observations, can lead to quite accurate results [12]. We address the problem of performing Reverse Nearest Neighbor (RNN) queries on such data. Given a query q, a reverse nearest neighbor query returns the objects in the database having q as one of its nearest neighbors. This query has been extensively studied on certain data [9, 15, 17]. Recent research has focused on RNN queries in uncertain spatial [3, 10] data. Xu et al were the first to address RNN queries on uncertain spatio-temporal data and showed how to answer an "interval reverse nearest neighbor query" [24]. This kind of query has many applications, for example in collaboration recommendation applications. However, as we will see later, the solution presented in [24] does not consider possible worlds semantics (PWS). In this work we fill this research gap by proposing algorithms to answer reverse nearest neighbour queries according to PWS.

The contributions of this work can be summarized as follows:

- We introduce two query definitions for the reverse nearest neighbor problem on uncertain trajectory data, namely the $P\exists RNNQ(q, \mathcal{D}, T, \tau)$ and $P\forall RNNQ(q, \mathcal{D}, T, \tau)$ query. The queries are consistent with existing definitions of nearest neighbor and window queries on this data.
- We demonstrate solutions to answer the queries we defined efficiently and, most importantly, according to possible worlds semantics.
- We provide an extensive experimental evaluation of the proposed methods both on synthetic and real world datasets.

This paper is organized as follows: In Section 2 we review related work on RNN queries and uncertain spatio-temporal data modeling. Section 3 provides a formal problem definition. Section 4 introduces algorithms for the queries proposed in Section 3. An extensive experimental evaluation follows in Section 5. Section 6 concludes this paper.

[3] http://www.bikely.com/, http://www.everytrail.com/, http://www.gpsxchange.com, http://www.gpsshare.com/

2 Related Work

Probabilistic Reverse-Nearest Neighbour Queries. Reverse (k)-Nearest Neighbor queries, initially proposed by Korn et al. [9] on certain data have been studied extensively in the past [15, 25, 17, 1, 19]. Many of the early solutions for RkNN queries rely on costly precomputations [9, 25, 1] and augment index structures such as R-trees or M-trees by additional information in order to speed up query evaluation. Follow-up techniques, such as TPL [17], aim at avoiding costly preprocessing at the cost of a more expensive query evaluation stage; moreover, they do not depend on specialized index structures. Recently, probabilistic reverse nearest neighbor queries have gained significant attention [10, 3, 2]. The solution proposed by Chen et al. [10] aims at processing PRNN queries on uncertain objects represented by continuous probability density functions (PDFs). In contrast, Cheema et al. [3] provided solutions for the discrete case. In the context of probabilistic reverse nearest neighbor queries, two challenges have to be addressed in order to speed up query evaluation. On the one hand, the I/O-cost has to be minimized; on the other hand, the solution has to be computationally efficient.

Uncertain Spatio-Temporal Data. Query processing in trajectory databases has received significant interest over the last ten years. (see for example [18, 16, 26, 23, 8]). Initially, trajectories have been assumed to be certain, by employing linear [18] or more complex [16] types of interpolation to handle missing measurements. These interpolation techniques, however can lead to impossible patterns of movement as, for example, a car might be assumed to drive through a lake. Other solutions such as computing the shortest path between consecutive locations can produce valid results, but do not provide probabilities for quantifying the result quality. As a result, a variety of uncertainty models and query evaluation techniques has been developed for moving object trajectories (e.g.[11, 22, 21, 7]).

A possible way to approach uncertain data is by providing conservative bounds for the positions of uncertain objects. These conservative bounds (such as cylinders [22, 21] or beads [20]) approximate trajectories and can answer queries such as "give me all objects that *could have* (or *definitely have*) the query as a nearest neighbor". However they cannot provide probabilities conforming to possible worlds semantics.

Another class of algorithms employ independent probability density functions (pdf) at each point of time. This way of modeling the uncertain positions of an object [4, 21, 11], can produce wrong results if a query considers more than a single point in time as shown in [7, 12], as temporal dependencies between consecutive object positions in time are ignored. A solution to this problem is modeling uncertain trajectories by stochastic processes.

In [13, 7, 14, 24], trajectories are modeled by Markov chains. Although the Markov chain model is still a *model* and can therefore only provide an approximate view of the world, it allows to answer queries according to possible worlds semantics, significantly increasing the quality of results. Recently, [12] addressed the problem of nearest neighbor queries based on the Markov model. Our work builds upon the results from this paper.

Regarding reverse nearest-neighbor processing using the Markov model, to the best of our knowledge, there exists only one work so far which addresses interval reverse nearest neighbor queries [24]. The approach basically computes for each point of time in the query interval separately the probability for each object $o \in \mathcal{D}$ to be the RNN to the query object. Then for each object o, the number of times where o has the highest probability to be RNN is counted. The object with the highest count is returned. Upon investigation, this approach has certain drawbacks. First, the proposed algorithm is not in accordance with possible worlds semantics, since successive points of time are considered independently (a discussion on the outcome of this treatment can be found in Section 3.2). Second, the paper does not show how to incorporate additional observations (besides the first appearance of an object).

3 Problem Definition

In this paper, following [7, 12], we define a spatio-temporal database $\mathcal{D}$ as a database storing triples (o_i, *time*, *location*), with o_i being a unique object identifier, *time* $\in \mathcal{T}$ a point in time and *location* $\in \mathcal{S}$ a position in space. Each of these triples describes an observation of object o_i at a given *time* at a given location *location*, e.g. a GPS measurement. Based on this definition an object o_i can be seen as a function $o_i(t) : \mathcal{T} \rightarrow \mathcal{S}$ that maps each point in time to a location in space; this function is called *trajectory*.

Following the related literature we assume a discrete time domain $\mathcal{T} = \{0, \ldots, n\}$. Furthermore, we assume a discrete state space of possible locations (*states*): $\mathcal{S} = \{s_1, ..., s_{|\mathcal{S}|}\} \subset \mathbb{R}^d$. Both of these assumptions are necessary to model uncertain trajectories by Markov chains. An object stored in the database can only be located in one of these states at each point in time. The semantics of such a state is application dependent; e.g., in a road network, the state space contains road crossings.

3.1 Uncertain Trajectory Model

The uncertain trajectory model used in this paper has been recently investigated (e.g., by [7, 12]) in the context of window queries and nearest neighbor queries. In the following, we recap this model. Let $\mathcal{D}$ be a database containing the trajectories of $|\mathcal{D}|$ uncertain moving objects $\{o_1, ..., o_{|\mathcal{D}|}\}$. An object $o \in \mathcal{D}$ is represented by a set of observations $\Theta^o = \{\langle t_1^o, \theta_1^o \rangle, \langle t_2^o, \theta_2^o \rangle, \ldots, \langle t_{|\Theta^o|}^o, \theta_{|\Theta^o|}^o \rangle\}$ with $t_i^o \in \mathcal{T}$ being the timestamp and $\theta_i^o \in \mathcal{S}$ the state (i.e. location) of observation Θ_i^o. Let $t_1^o < t_2^o < \ldots < t_{|\Theta^o|}^o$. This model assumes observations to be certain, however between two certain observations the location of an object is unknown and therefore uncertain. To model this uncertainty we can interpret the uncertain object o as a stochastic process [7]. With this interpretation, the location of an uncertain object o at time t becomes a realization the random variable $o(t)$. Considering a time interval $[t_s, t_e]$, results in a sequence of uncertain locations of an object, i.e. a stochastic process. With this definition we can compute the probability of a given trajectory.

In this paper, following [7, 6, 24], we investigate query evaluation on a first-order Markov model. The advantage of the first-order Markov model is its simplicity. By employing a Markov model, the position $o(t+1)$ of object o at time $t+1$ only depends on the location of o at time t, i.e. $o(t)$. Therefore, transitions between consecutive points in time can be easily realized by matrix multiplication. However note that by modeling uncertain objects by a Markov chain, the motion of these objects basically degenerates to a random walk, clearly not a realistic motion pattern of objects in real life. The motion of cars for example is better described by shortest paths than by a random walk. Fortunately, as showed in [12], by incorporating a second source of information into the model, namely observations of an object, the Markov chain can be used to accurately describe the uncertainty area of an uncertain object.

Now, let the state space of the Markov chain be given as the spatial domain $\mathcal{S}$, i.e. points in Euclidean space. The *transition probability* $M^o_{ij}(t) := P(o(t+1) = s_j | o(t) = s_i)$ denotes the probability of object o moving from state s_i to s_j at time t. These transition probabilities can be stored in a matrix $M^o(t)$, i.e. the *transition matrix* of o at time t. The transition matrix of an object might change with time, and different objects might have different transition matrices. The first property is useful to model varying motion patterns of moving objects at different times of a day, a month or a year: birds move to the south in autumn and to the north during springtime. Each of these patterns could be described by a different transition matrix. The second property is useful to model different classes of objects such as busses and taxis.

Let $\boldsymbol{s}^o(t) = (s_1, \ldots, s_{|\mathcal{S}|})^T$ be the probability distribution vector of object o at time t, with $\boldsymbol{s}^o_i(t) = P(o(t) = s_i)$. An entry $\boldsymbol{s}^o_i(t)$ of the vector describes the probability of o entering s_i at time t. The state vector $\boldsymbol{s}^o(t+1)$ can be computed from $\boldsymbol{s}^o(t)$ as follows: $\boldsymbol{s}^o(t+1) = M^o(t)^T \cdot \boldsymbol{s}^o(t)$ Note that simple matrix multiplications can only be employed in the absence of observations. In the presence of observations, transition matrices must be adapted, see [12]. Finally note that we assume different objects to be *mutually independent*.

3.2 Probabilistic Reverse Nearest Neighbor Queries

In the following we define two types of probabilistic time-parameterized RNN queries. The queries conceptually follow the definitions of time-parameterized nearest neighbor and window queries in [12, 7]. We assume that the RNN query takes as input a set of timestamps T and either a single state or a (certain) query trajectory q. Still, our definitions and solutions can be trivially extended to consider RNN queries where the input states are uncertain.

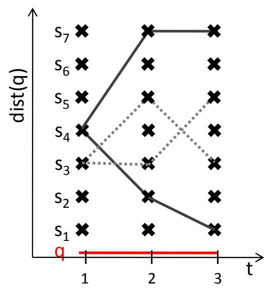
Definition 1 (P∃RNN Query). *A probabilistic $\exists$ reverse nearest neighbor query retrieves all objects $o \in \mathcal{D}$ having a sufficiently high probability to be the reverse nearest neighbor of q for at least one point of time $t \in T$, formally:*

$$P\exists RNNQ(q, \mathcal{D}, T, \tau) = \{o \in \mathcal{D} : P\exists RNN(o, q, \mathcal{D}, T) \geq \tau\}$$

where $P\exists RNN(o, q, \mathcal{D}, T) = P(\exists t \in T : \forall o' \in \mathcal{D} \setminus o : d(o(t), q(t)) \leq d(o(t), o'(t)))$

and $d(x, y)$ is a distance function defined on spatial points, typically the Euclidean distance.

This query returns all objects from the database having a probability greater τ to have q as their probabilistic $\exists$ nearest neighbor [12]. In addition to this $\exists$ query, we consider RNN queries with the $\forall$ quantifier:



object	trajectory	P(tr)
o_1 —	$tr_{1,1} = s_4, s_2, s_1$	0.4
o_1 —	$tr_{1,2} = s_4, s_7, s_7$	0.6
o_2 ····	$tr_{2,1} = s_3, s_3, s_5$	0.2
o_2 ····	$tr_{2,2} = s_3, s_5, s_3$	0.8

Fig. 1. Example database of uncertain trajectories

Definition 2 (P∀RNN Query). *A probabilistic ∀ reverse nearest neighbor query retrieves all objects $o \in \mathcal{D}$ having a sufficiently high probability ($P\forall RNN$) to be the reverse nearest neighbor of q for the entire set of timestamps T, formally:*

$$P\forall RNNQ(q, \mathcal{D}, T, \tau) = \{o \in \mathcal{D} : P\forall RNN(o, q, \mathcal{D}, T) \geq \tau\}$$

where $P\forall RNN(o, q, \mathcal{D}, T) = P(\forall t \in T : \forall o' \in \mathcal{D} \setminus o : d(o(t), q(t)) \leq d(o(t), o'(t)))$

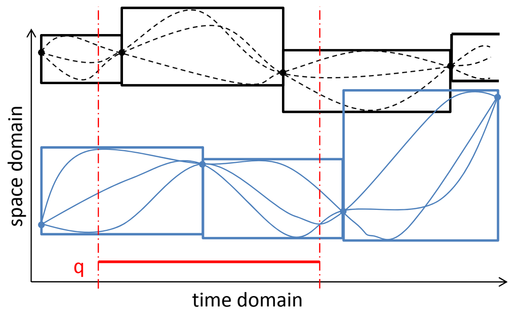
The above definition returns all objects from the database which have a probability greater τ to have q as their probabilistic ∀ nearest neighbor [12].

Example 1. To illustrate the differences between the proposed queries consider the example in Figure 1. Here for simplicity the query is not moving at all over time and the two objects o_1 and o_2 each have 2 possible trajectories. o_1 follows the lower trajectory ($tr_{1,1}$) with a probability of 0.4 and the upper trajectory ($tr_{1,2}$) with a probability of 0.6. o_2 follows trajectory $tr_{2,1}$ with a probability of 0.2 and trajectory $tr_{2,2}$ with a probability of 0.8. For query object q and the query interval $T = [2, 3]$, we can compute the probability for each object to be probabilistic reverse nearest neighbor of q. Specifically for o_1 the probability $P\exists RNN(o_1, q, \mathcal{D}, T) = 0.4$ since whenever o_1 follows $tr_{1,1}$ then at least at $t = 3$, o_1 is RNN of q. The probability for $P\forall RNN(o_1, q, \mathcal{D}, T)$ in contrast is 0.32 since it has to hold that o_1 follows $tr_{1,1}$ (this event has a probability of 0.4) and o_2 has to follow $tr_{2,2}$ (this event has a probability of 0.8). Since both events are mutually independent we can just multiply the probabilities to obtain the final result probability. Regarding object o_2 we can find no possible world (combination of possible trajectories of the two objects) where o_2 is always ($T = [2, 3]$) RNN, thus $P\forall RNN(o_2, q, \mathcal{D}, T) = 0$. However $P\exists RNN(o_2, q, \mathcal{D}, T) = 0.6$ since whenever o_1 follows $tr_{1,2}$ then o_2 is RNN either at $t = 2$ or at $t = 3$.

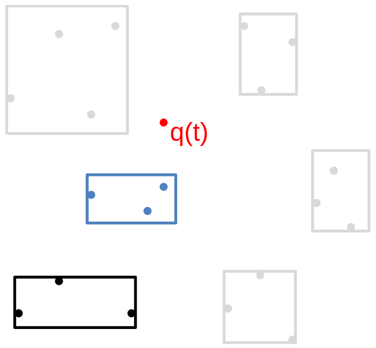
An important observation is that it is not possible to compute these probabilities by just considering the snapshot RNN probabilities for each query time stamp individually. For example the probability for o_2 to be RNN at time $t = 2$ is 0.12 (the possible world where objects follow $tr_{1,2}$ and $tr_{2,1}$) and the probability at $t = 3$ is 0.48 (the possible world where objects follow $tr_{1,2}$ and $tr_{2,2}$). However these events are not mutually independent and thus we cannot just multiply the probabilities to obtain the probability that object o_2 is RNN at both points of time (note that the true probability of this event is $P\forall RNN(o_2, q, \mathcal{D}, T) = 0$).

4 PRNN Query Processing

To process the two RNN query types defined in Section 3, we proceed as follows. First, we perform a temporal and spatial filtering to quickly find candidates in the database and exclude as many objects as possible from further processing. In the second step we perform a verification of the remaining candidates to obtain the final result. Although different solutions to this problem are possible, we decided to describe an algorithm that splits the query involving several timesteps into a series of queries involving only a single point in time during the pruning phase. The interesting point in this algorithm is that it shows that spatial pruning *does not introduce errors when disregarding temporal correlations*. However, disregarding temporal correlation during the probability computation phase *does introduce errors*.



(a) Uncertatin trajectories in 1D space



(b) space (2D) at one point of time t

Fig. 2. Spatio-temporal filtering (only leaf nodes are shown)

4.1 Temporal and Spatial Filtering

In the following we assume that the uncertain trajectory database $\mathcal{D}$ is indexed by an appropriate data structure, like the UST tree [6].[4] For the UST tree, the set of possible (location,time)-tuples between two observations of the same object is conservatively approximated by a minimum bounding rectangle (MBR) (cf Figure 2(a)). All these rectangles are then used as an input for an R*-Tree. For simplicity we only rely on these MBRs and do not consider the more complicated probabilistic approximations of each object.

The pseudo code of the spatio-temporal filter is illustrated in Algorithm 1. The main idea is to (i) perform *candidates search* for each timestamp t in the query interval T separately (cf. Figure 2(a)) and (ii) for each candidate find the set of objects which are needed for the verification step (influence objects S_{ifl}^{cnd}). For each t we consider the R-Tree I_{DB}^{t} which results by intersecting the time-slice t with the R-Tree I_{DB} (cf Figure 2(b)). This can be done efficiently during query processing, by just ignoring pages of

[4] Note that the techniques for pruning objects do not rely on this index and thus can also be applied in scenarios where there is no index present.

Algorithm 1 Spatio-Temporal Filter for the P∀RNN query

```
Require: $q$,$T$, $I_{DB}$
 1: $\forall t \in T : S^t_{cnd} = \emptyset$
 2: $\forall t \in T : S^{cnd,t}_{ifl} = \emptyset$
 3: for each $t \in T$ do
 4:     init min-heap $H$ ordered by minimum distance to $q$
 5:     insert root entry of $I^t_{DB}$ into $H$
 6:     $S_{prn} = \emptyset$
 7:     while $H$ is not empty do
 8:         de-heap an entry $e$ from $H$
 9:         if $\exists e_2 \in H \cup S_{prn} \cup S^t_{cnd} : Dom(e_2, q, e)$ then
10:             $S_{prn} = S_{prn} \cup \{e\}$
11:         else if $e$ is directory entry then
12:             for each child $ch$ in $e$ do
13:                 insert $ch$ in $H$
14:             end for
15:         else if $e$ is leaf entry then
16:             $S^t_{cnd} = S^t_{cnd} \cup \{e\}$
17:         end if
18:     end while
19:     for each $cnd \in S^t_{cnd}$ do
20:         if $\exists le : Dom(le, q, e)$ then
21:             break;
22:         end if
23:         $S^{cnd,t}_{ifl} = \{le : \neg Dom(le, q, e)\} \wedge \neg Dom(q, le, e)\}$
24:     end for
25: end for
26: $S_{ref} = \bigcap_{t \in T} S^t_{cnd}$
27: for each $cnd \in S_{ref}$ do
28:     $S^{cnd}_{ifl} = \bigcup_{t \in T} S^{cnd,t}_{ifl}$
29: end for
30: return $\forall cnd \in S_{ref} : (cnd, S^{cnd}_{ifl})$
```

the index, that do not intersect with the value of t in the temporal domain. For each time t a reverse nearest neighbor candidate search [2] is performed.

Therefore, a heap H is initialized, which organizes its entries by their minimum distance to the query object q. H initially only contains the root node of I^t_{DB}. Additionally we initialize two empty sets. S_{cnd} contains all RNN candidates which are found during query processing and S_{prn} contains objects (leaf entries) or entries which have been verified not to contain candidates. Then, as long as there are entries in H, a best-first traversal of I^t_{DB} is performed. For each entry e, which is de-heaped from H, the algorithm checks whether e can be pruned (i.e., it cannot contain potential candidates) by another object or entry e_2 which has already been seen during processing. This is the case if e_2 dominates q w.r.t. e, i.e. e_2 is definitely closer to e than q which implies that e cannot be RNN of q. To verify spatial domination we adapt the technique proposed in [5], as shown in the following lemma.

Lemma 1 (Spatial Domination [5]). *Let A, B, R be rectangular approximations then the relation*

$$Dom(A, B, R) = \forall a \in A, b \in B, r \in R : dist(a, r) < dist(b, r)$$

can efficiently be checked by the following term

$$Dom(A, B, R) = \sum_{i=1}^{d} \max_{b_i \in \{B_i^{min}, B_i^{max}\}} (\mathit{MaxDist}(A_i, b_i)^2 - \mathit{MinDist}(Q_i, b_i)^2) < 0$$

where X_i ($X \in \{A, B, Q\}$) denotes the projection interval of the rectangular region of X on the i^{th} dimension, X_i^{min} (X_i^{max}) denotes the lower (upper) bound of the interval X_i, and MaxDist(I, p) (MinDist(I, p)) denotes the maximal (minimal) distance between a one-dimensional interval I and a one-dimensional point p.

An entry which is pruned by this technique is moved to the S_{prn} set. If an entry cannot be pruned it is either moved to the candidate set if it is a leaf entry or put into the heap H for further processing.

After the index traversal, for each candidate it can be checked if the candidate is pruned by another object. If the other object it is definitely closer to the candidate than the query, the candidate object can be discarded (see line 21). To find the set of objects that could possibly prune a candidate, we can again use the domination relation (see line 23). An object (leaf entry le) is necessary for the verification step if it might be closer to the candidate than the query, which is reflected by the statement in this line. A more detailed description of this step can be found in [2].

After performing this process for each timestamp t we have to merge the results for each point of time to obtain the final result. In the case of a P∀RNN we intersect the *candidate* sets for each point in time. The only difference for the P∃RNN query is that we have to unify the results in this step.

These *influencing* objects of each candidate have to be unified for each time $t \in T$ to obtain the final set of influencing objects. The algorithm ultimately returns a list of candidate objects together with their sets of influencing objects.

4.2 Verification

The objective of the verification step is to compute, for each candidate c, the probability $P\exists RNN(c, q, \mathcal{D}, T)$ ($P\forall RNN(c, q, \mathcal{D}, T)$) and compare this probability with the probability threshold τ. An interesting observation is, that we were able to prune objects based on the consideration of each point $t \in T$ separately, however as shown in Section 3.2 it is not possible to obtain the final probability value by just considering the single time probabilities. Thus our approach relies on sampling of possible trajectories for each candidate and the corresponding influence objects. For this step, we can utilize the techniques from [12]. On each sample (which then consists of certain trajectories) we then are able to efficiently evaluate the query predicate. Repeating this step often enough we are able to approximate the true probability of $P\exists RNN(c, q, \mathcal{D}, T)$

($P\forall RNN(c, q, \mathcal{D}, T)$) by the percentage of samples where the query predicate was satisfied. The algorithm for the verification of the P∀RNN query is given in Algorithm 2. Note, that it is possible to early terminate sampling of influence objects, when we find a time t where the candidate is not closer to q than to the object trajectory just sampled. For the P∃RNN query we can also implement this early termination whenever we can verify the above for each point of time.

Algorithm 2 Verification for the P∀RNN query

```
Require: $q$,$T$, $cnd$, $S_{ifl}^{cnd}$, num_samples
 1: num_satisfied = 0
 2: for $0 \leq i \leq num\_samples$ do
 3:     num_satisfied = num_satisfied + 1
 4:     $cnds = sampleTrajectory(cnd)$
 5:     for all $o \in S_{ifl}^{cnd}$ do
 6:         $os = sampleTrajectory(o)$
 7:         if $\exists t \in T : dist(cnds(t), os(t)) < dist(cnds(t), q(t))$ then
 8:             num_satisfied = num_satisfied - 1
 9:             break
10:         end if
11:     end for
12: end for
13: return num_satisfied/num_samples
```

5 Experiments

In our experimental evaluation, we focus on testing the efficiency of our algorithm for P∀RNNQ and P∃RNNQ, by measuring (i) the number of candidates and pruners remaining after pruning irrelevant objects based on their spatio-temporal MBRs, and (ii) the runtime of the refinement procedure, i.e. sampling. Similar to [12], we split the sampling procedure into adapting the transition matrices (building the trajecory sampler) of the candidate objects and the actual sampling process, as the adaption of transition matrices can be done as a preprocessing step. All experiments have been conducted in the UST framework that has also been used in [12]. The experimental evaluation has been conducted on a desktop computer with Intel i7-870 CPU at 2.93 GHz and 8GB of RAM.

5.1 Setup

Our experiments are based on both artificial and real data sets. In the artificial dataset states are drawn from a uniform distribution. The real data set uses an OpenStreetMap graph of the city of Beijing to derive the underlying state space. Both of the datasets have also been used in [12], however here we recap their generation for the sake of completeness.

Artificial Data. Artificial data was generated in four steps. *First* a state space was generated by randomly drawing N points from the $[0, 1]^2$ space. These points represent

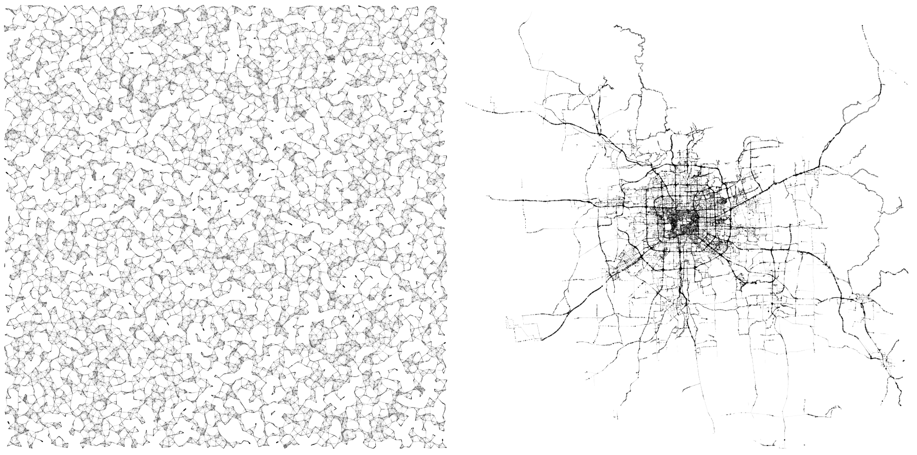

(a) State Space (Synthetic Data) (b) State Space (Real Data)

Fig. 3. Examples of the models used for synthetic and real data. Black lines denote transition probabilities. Thicker lines denote higher probabilities, thinner lines lower probabilities. The synthetic model consists of 10k states.

the states of the underlying Markov chain. *Second* we built a graph from these neighboring states by connecting neighboring points. Given a reference point p, we selected all points within in a radius of $r = \sqrt{\frac{b}{N*\pi}}$ from p, connecting each of the resulting points with p. Here b denotes the average branching factor of the resulting graph structure. *Third* we weighted each of the graph's edges based on the distance between the corresponding states: The transition probability between two states was set indirectly proportional to the states' distance relative to all outgoing distances to other states. The resulting transition matrix is independent of a specific object and therefore provides a general model for all objects in the database. An example of such a model can be found in 3(a). Finally and *fourth* we had to generate uncertain trajectories from this data for each object o in the database. Each of these uncertain trajectories has a length of 100 timesteps. To generate the uncertain object, we first sampled a random sequence of states from the state space and connected two consecutive states by their shortest path. The resulting paths contain straight segments (shortest paths), modelling the directed motion of objects, and random changes in direction that can appear e.g. when objects move from one destination (e.g. home, work, or disco for humans) to another destination. The resulting trajectories serve as a certain baseline and are made uncertain by considering only a subset of the trajectory points: every l^{th} node of the trajectory, $l = i * v$, $v \in [0, 1]$ is used as an observation of the corresponding uncertain object. Here, i models the time difference between observations and v is a lag factor, making the object slower. With, for example, $v = 0.8$, the object moves only with 80% of its maximum speed. We randomly distributed the resulting uncertain object over the time horizon of the database (defaulting 1000 timesteps). The objects can then be indexed by a UST tree [6]. For our experiments we determined candidates based on MBR filtering, i.e. the MBR over all states that can be reached by an object between two consecutive observations. For our experiments we concentrate on the case of the query q beeing given as a query state.

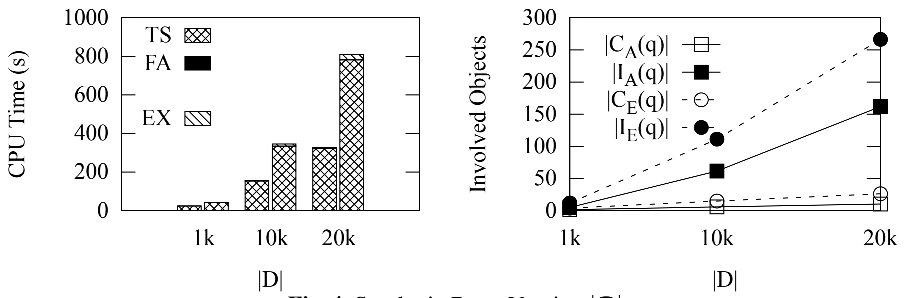


Fig. 4. Synthetic Data, Varying $|\mathcal{D}|$.

Real Data. As a real dataset we used taxi trajectories in the city of Beijing [27], equvivalent to [12]. After an initial cleaning phase, the taxi trajectories were map-matched to a street graph taken from OpenStreetMap. Then, from the resulting map-matched trajectories, a general model, i.e. the Markov transition matrix and the underlying state space was generated with a time interval of 10s between two consecutive tics. The resulting model is visualized in Figure 3(b). In the model, transition probabilities denote turning probabilities of taxis at street crossings and states model the street crossings. Due to the sparsity of taxi trajectories only a subset of the real street crossings was hit by a taxi. Therefore the resulting network consisting of only 68902 states is smaller than the actual street network. The uncertain object trajectories were taken directly from the trajectory data. Again, uncertain objects have a lifetime of 100 tics, and uncertain trajectories were generated by taking each 8-th measurement as an observation. For a more detailed description of the dataset we refer to [27], for a detailed description of the model generation from the real dataset we refer to [12].

5.2 Evaluation: P∀RNNQ and P∃RNNQ

The default setting for our performance analysis is as follows: We set the number of states to $N = |\mathcal{S}| = 100$k, the database size (number of objects) to $|\mathcal{D}| = 10k$, the average branching factor (synthetic data) to $b = 8$, probability threshold $\tau = 0$. The length of the query interval was set to $|T| = 10$.

In each experiment, the left plot shows a stacked histogram, visualizing the cost for building the trajectory sampler (TS) and the actual cost for sampling (EX for the $P\exists RNN$ query, FA for the $P\forall RNN$ query). The right plot first visualizes the number of candidates ($C_x(q)$) for the $P\exists RNN$ ($x = E$) and $P\forall RNN$ ($x = A$) query, i.e. the number of objects for which the nearest neighbor has to be computed. Second it visualizes the number of pruners or influence objects ($I_x(q)$), i.e. the number of objects that can prune candidates. Clearly the number of candidates and pruners is different for $P\forall RNN$ and $P\exists RNN$ queries: for the $P\exists RNN$ query objects not totally overlapping the query interval can be candidates, increasing the number of candidates. For the $P\forall RNN$ query, candidates can be definitely pruned if at least at one point of time another object prunes the candidate object.

Varying $|\mathcal{D}|$. Let us first analyze the impact of the database size (see Figure 4), i.e. the number of uncertain objects on the runtime of a probabilistic reverse nearest neighbor query. First of all note that the number of candidates and influence objects increases if the database gets large. This is the case because with more objects, the

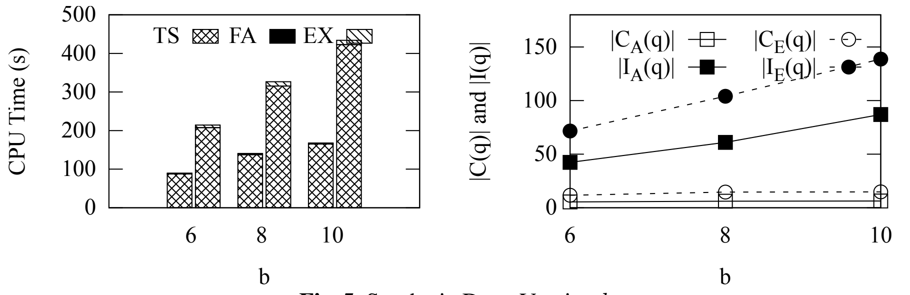


Fig. 5. Synthetic Data, Varying b.

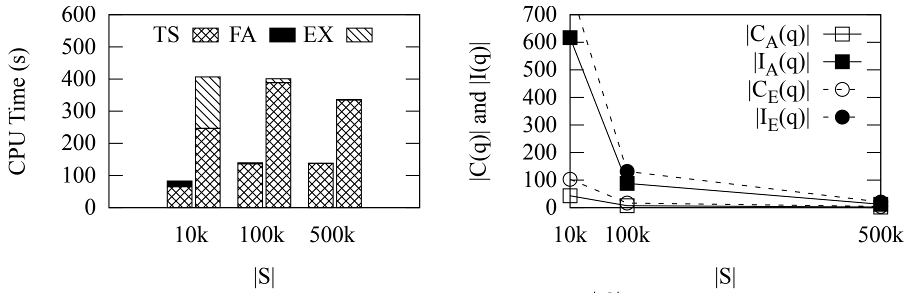


Fig. 6. Synthetic Data, Varying $|\mathcal{S}|$.

density of objects increases and therefore more objects become possible results of the RNN query. Also, clearly, the number of influence objects increases due to the higher degree of intersection of MBRs. Second the the probability computation of candidate objects during refinement is mostly determined by the computation of the adapted transition matrices, i.e. building the trajectory sampler. This is actually good news, as this step can also be performed offline and the resulting transition matrices can be stored on disk. Last note that evaluating the $P\exists RNN$ query during the actual sampling process (EX and FA for $P\exists RNN$ and $P\forall RNN$ respectively) is also more expensive than the $P\forall RNN$ query, as more samples have to be drawn for each candidate object: possible worlds of $P\forall RNN$-candidates can be pruned if a single object at a single point in time prunes the candidate. For the $P\exists RNN$ query, all points in time have to be pruned which is much less probable. Additionally, more candidates than for the $P\forall RNN$ have to be evaluated, also increasing the complexity of the $P\exists RNN$ query.

Varying b. The effect of varying the branching factor b (see Figure 5) is similar but not as severe as varying the number of objects. Increasing the branching factor increases the number of states that can be reached during a single transition. In our setting, this also increases the uncertainty area of an uncertain object, making pruning less effective, and therefore increasing the number of objects that have to be considered during refinement. As a result, the computational complexity of the refinement phase increases with increasing branching factor.

Varying $|\mathcal{S}|$. Increasing the number of states in the network (see Figure 6)while keeping the branching factor b constant shows an opposite effect on the number of candidates: as the number of states increases, objects become more sparsely distributed such that pruning becomes more effective. Note that for small networks especially the sampling process of the $P\exists RNN$ query becomes very expensive. This effect dimin-

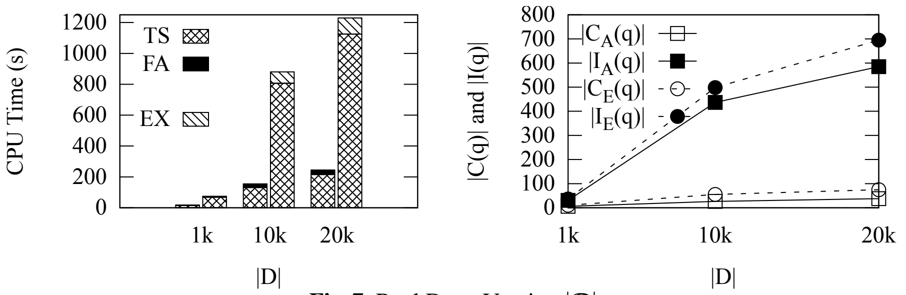


Fig. 7. Real Data, Varying $|\mathcal{D}|$.

ishes with increasing size of the network. Although less objects are involved, building the adapted transition matrices becomes more expensive, as for example matrix operations become more expensive with larger state spaces.

Real Dataset. Last but not least we show results of the $P\forall RNN$ and $P\exists RNN$ queries on the real dataset (see Figure 7). We decided to vary the number of objects as the number of states and the branching factor is inherently given by the underlying model. The results are similar to the synthetic data, however more than twice as many objects have to be considered during refinement. We explain this exemplarily by the non-uniform distribution of taxis in the network. A part of this performance difference can also be partially explained by the slightly smaller network consisting of about 70k states instead of 100k states in the default setting.

6 Conclusion

In this paper we addressed the problem of probabilistic RNN queries on uncertain spatio-temporal data following the possible worlds semantics. We defined two queries, the $P\exists RNNQ(q, \mathcal{D}, T, \tau)$ and the $P\forall RNNQ(q, \mathcal{D}, T, \tau)$ query and proposed pruning techniques to exclude irrelevant objects from costly propability computations. We then used sampling to compute the actual $P\exists RNNQ(q, \mathcal{D}, T, \tau)$ and $P\forall RNNQ(q, \mathcal{D}, T, \tau)$ probabilities for the remaining candidate objects. During an extensive performance analysis on both synthetic and real data we empirically evaluated our theoretic results.

References

1. E. Achtert, C. Böhm, P. Kröger, P. Kunath, A. Pryakhin, and M. Renz. Efficient reverse k-nearest neighbor search in arbitrary metric spaces. In *Proc. SIGMOD*, 2006.
2. T. Bernecker, T. Emrich, H.-P. Kriegel, M. Renz, S. Zankl, and A. Züfle. Efficient probabilistic reverse nearest neighbor query processing on uncertain data. In *Proc. VLDB*, pages 669–680, 2011.
3. M. A. Cheema, X. Lin, W. Wang, W. Zhang, and J. Pei. Probabilistic reverse nearest neighbor queries on uncertain data. *IEEE Trans. Knowl. Data Eng.*, 22(4):550–564, 2010.
4. R. Cheng, D. Kalashnikov, and S. Prabhakar. Querying imprecise data in moving object environments. In *IEEE TKDE*, volume 16, pages 1112–1127, 2004.
5. T. Emrich, H.-P. Kriegel, P. Kröger, M. Renz, and A. Züfle. Boosting spatial pruning: On optimal pruning of mbrs. In *Proc. SIGMOD*, 2010.

6. T. Emrich, H.-P. Kriegel, N. Mamoulis, M. Renz, and A. Züfle. Indexing uncertain spatio-temporal data. pages 395–404, 2012.
7. T. Emrich, H.-P. Kriegel, N. Mamoulis, M. Renz, and A. Züfle. Querying uncertain spatio-temporal data. pages 354–365, 2012.
8. R. H. Güting, T. Behr, and J. Xu. Efficient *k*-nearest neighbor search on moving object trajectories. *VLDB J.*, 19(5):687–714, 2010.
9. F. Korn and S. Muthukrishnan. Influenced sets based on reverse nearest neighbor queries. In *Proc. SIGMOD*, 2000.
10. X. Lian and L. Chen. Efficient processing of probabilistic reverse nearest neighbor queries over uncertain data. *VLDB J.*, 18(3):787–808, 2009.
11. H. Mokhtar and J. Su. Universal trajectory queries for moving object databases. pages 133–144, 2004.
12. J. Niedermayer, A. Züfle, T. Emrich, M. Renz, N. Mamoulis, L. Chen, and H.-P. Kriegel. Probabilistic nearest neighbor queries on uncertain moving object trajectories. In *Proc. VLDB*, page (to appear), 2014.
13. S. Qiao, C. Tang, H. Jin, T. Long, S. Dai, Y. Ku, and M. Chau. Putmode: prediction of uncertain trajectories in moving objects databases. *Appl. Intell.*, 33(3):370–386, 2010.
14. C. Ré, J. Letchner, M. Balazinksa, and D. Suciu. Event queries on correlated probabilistic streams. In *Proc. SIGMOD*, pages 715–728, 2008.
15. I. Stanoi, D. Agrawal, and A. E. Abbadi. Reverse nearest neighbor queries for dynamic databases. In *Proc. DMKD*, 2000.
16. Y. Tao, C. Faloutsos, D. Papadias, and B. Liu. Prediction and indexing of moving objects with unknown motion patterns. In *Proc. SIGMOD*, pages 611–622, 2004.
17. Y. Tao, D. Papadias, and X. Lian. Reverse kNN search in arbitrary dimensionality. In *Proc. VLDB*, 2004.
18. Y. Tao, D. Papadias, and Q. Shen. Continuous nearest neighbor search. In *Proc. VLDB*, pages 287–298, 2002.
19. Y. Tao, M. L. Yiu, and N. Mamoulis. Reverse nearest neighbor search in metric spaces. *IEEE TKDE*, 18(9):1239–1252, 2006.
20. G. Trajcevski, A. N. Choudhary, O. Wolfson, L. Ye, and G. Li. Uncertain range queries for necklaces. pages 199–208, 2010.
21. G. Trajcevski, R. Tamassia, H. Ding, P. Scheuermann, and I. F. Cruz. Continuous probabilistic nearest-neighbor queries for uncertain trajectories. In *Proc. EDBT*, pages 874–885, 2009.
22. G. Trajcevski, O. Wolfson, K. Hinrichs, and S. Chamberlain. Managing uncertainty in moving objects databases. *ACM Trans. Database Syst.*, 29(3):463–507, 2004.
23. X. Xiong, M. F. Mokbel, and W. G. Aref. Sea-cnn: Scalable processing of continuous k-nearest neighbor queries in spatio-temporal databases. In *Proc. ICDE*, pages 643–654, 2005.
24. C. Xu, Y. Gu, L. Chen, J. Qiao, and G. Yu. Interval reverse nearest neighbor queries on uncertain data with markov correlations. 2013.
25. C. Yang and K.-I. Lin. An index structure for efficient reverse nearest neighbor queries. In *Proc. ICDE*, 2001.
26. X. Yu, K. Q. Pu, and N. Koudas. Monitoring k-nearest neighbor queries over moving objects. In *Proc. ICDE*, pages 631–642, 2005.
27. J. Yuan, Y. Zheng, X. Xie, and G. Sun. Driving with knowledge from the physical world. pages 316–324, 2011.
28. J. Yuan, Y. Zheng, C. Zhang, W. Xie, X. Xie, and Y. Huang. T-drive: Driving directions based on taxi trajectories. In *Proc. ACM GIS*, 2010.
29. Y. Zheng, Q. Li, Y. Chen, X. Xie, and W. Ma. Understanding mobility based on gps data. In *Proc. Ubicomp*, pages 312–312, 2008.