Probabilistic Nearest Neighbor Queries on Uncertain Moving Object Trajectories

Johannes Niedermayer*, Andreas Züfle*, Tobias Emrich*,
Matthias Renz*, Nikos Mamoulis°, Lei Chen+, Hans-Peter Kriegel*
*Institute for Informatics, Ludwig-Maximilians-Universität München
{niedermayer,zuefle,emrich,kriegel,renz}@dbs.ifi.lmu.de

°Department of Computer Science, University of Hong Kong
nikos@cs.hku.hk

+Department of Computer Science and Engineering, Hong Kong University of Science and Technology
leichen@cse.ust.hk

ABSTRACT

Nearest neighbor (NN) queries in trajectory databases have received significant attention in the past, due to their applications in spatio-temporal data analysis. More recent work has considered the realistic case where the trajectories are uncertain; however, only simple uncertainty models have been proposed, which do not allow for accurate probabilistic search. In this paper, we fill this gap by addressing probabilistic nearest neighbor queries in databases with uncertain trajectories modeled by stochastic processes, specifically the Markov chain model. We study three nearest neighbor query semantics that take as input a query state or trajectory q and a time interval, and theoretically evaluate their runtime complexity. Furthermore we propose a sampling approach which uses Bayesian inference to guarantee that sampled trajectories conform to the observation data stored in the database. This sampling approach can be used in Monte-Carlo based approximation solutions. We include an extensive experimental study to support our theoretical results.

1. INTRODUCTION

With the wide availability of satellite, RFID, GPS, and sensor technologies, spatio-temporal data can be collected in a massive scale. The efficient management of such data is of great interest in a plethora of application domains: from structural and environmental monitoring and weather forecasting, through disaster/rescue management and remediation, to Geographic Information Systems (GIS) and traffic control and information systems. In most current research however, each acquired trajectory, i.e., the function of a spatio-temporal object that maps each point in time to a position in space, is assumed to be known entirely without any uncertainty. However, the physical limitations of the sensing devices or limitations of the data collection process introduce sources of uncertainty.

Specifically, it is usually not possible to *continuously* capture the position of an object for each point of time. In an indoor tracking environment where the movement of a person is captured using static RFID sensors, the position of the people in-between two successive tracking events is not available ([1]). The same holds for geo-social network (GSN) applications, where users have recently been enabled to publicly share trajectories, such as bike routes[1], tourist routes[2] and GPS trajectories[3]. In many applications, the frequency of data collection is often decreased to save resources such as battery power and wireless network traffic. Examples of trajectories with a relatively low frequency can be found on Bikely. Furthermore, traditional check-in data of GSN users often shows a frequency high enough to allow inference of a user's position in between discrete check-ins. Furthermore, incomplete (location, time) data is also collected in mobile object tracking applications. For example, in the T-Drive dataset ([3]) which consists of GPS-logs of taxis in Beijing, the time between two successive GPS measurements ranges from two seconds up to several minutes. In the GeoLife dataset ([4]), GPS observations of mobile users are logged frequently, usually every 1-5 seconds per point, while some observations still have a lower sampling rate. All these datasets create a common challenge of interpolating the position of a user in-between discrete observations. In-between these observations the exact values are not explicitly stored in the database and are thus uncertain from the database perspective.

In this work, we consider a database $\mathcal{D}$ of uncertain moving object trajectories, where for each trajectory there is a set of observations for only some of the history timestamps. Thus, the entire trajectory of an object is described by a time-dependent random variable, i.e., a stochastic process. Given a reference state or trajectory q and a time interval T, we define *probabilistic* nearest-neighbor (PNN) query semantics, which are extensions of nearest neighbor queries in trajectory databases [5, 6, 7, 8]. Specifically, a P∃NNQ (P∀NNQ) query retrieves all objects in $\mathcal{D}$, which have sufficiently high probability to be the NN of q at one time (at the entire set of times) in T; a probabilistic *continuous* NN (PCNNQ) query finds for each object $o \in \mathcal{D}$ the time subsets T_i of T, wherein o has high enough probability to be the NN of q at the entire set of times in T_i. Note that to the best of our knowledge this is the first approach that tackles the PNN query problem correctly in consideration of possible worlds semantics.

PNN queries find several applications in analyzing historical trajectory data. For example, consider a geo-social network where users can publish their current spatial position at any time by so-called check-ins. For a historical event, users might want to find

This work is licensed under the Creative Commons Attribution-NonCommercial-NoDerivs 3.0 Unported License. To view a copy of this license, visit http://creativecommons.org/licenses/by-nc-nd/3.0/. Obtain permission prior to any use beyond those covered by the license. Contact copyright holder by emailing info@vldb.org. Articles from this volume were invited to present their results at the 40th International Conference on Very Large Data Bases, September 1st - 5th 2014, Hangzhou, China.
Proceedings of the VLDB Endowment, Vol. 7, No. 3
Copyright 2013 VLDB Endowment 2150-8097/13/11.

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

their nearest friends during this event, e.g. to share pictures and experiences. As another application example, consider GPS-tracked taxi cars as given in the T-Drive dataset [3] where PNN queries can be used for analysis tasks like the assessment of taxi-client assignment procedures or for search tasks like searching for taxi drivers that might have observed a certain event like a car accident or a criminal activity such as a bank robbery. The taxi drivers that have been closest to the certain event location during the time the event might happened are potential witnesses. Note that this example application is used as our running application throughout this paper.

The main contributions of our work are as follows:

- A thorough theoretical complexity analysis for variants of probabilistic NN query problems.
- A sampling-based approximate solution for all PNN problems which is based on Bayesian inference.
- Thorough experimental evaluation of the proposed concepts on real and synthetic data.

The rest of the paper is structured as follows. Section 2 reviews related work. Section 3 provides a formal problem definiton. A complexity analysis, approximate solutions and pruning techniques of the proposed query semantics are provided in Sections 4-6. An extensive experimental evaluation of the proposed techniques is presented in Section 7. Section 8 briefly discusses general kNN queries. Section 9 concludes this work.

2. RELATED WORK

Within the last decade, a considerable amount of research effort has been put into query processing in trajectory databases (e.g. [8, 9, 10, 11, 6]). In these works, the trajectories have been assumed to be certain, by employing linear [8] or more complex [9] types of interpolation to supplement sparse observational data. However, employing linear interpolation between consecutive observations might create impossible patterns of movement, such as cars travelling through lakes or similar impossible-to-cross terrain. Furthermore, treating the data as uncertain and answering probabilistic queries over them offers better insights[4].

Uncertain Trajectory Modeling. Several models of uncertainty paired with appropriate query evaluation techniques have been proposed for moving object trajectories (e.g. [12, 13, 14, 15]). Many of these techniques aim at providing conservative bounds for the positions of uncertain objects. This can be achieved by employing geometric objects such as cylinders [13, 14] or beads [16] as trajectory approximations. While such approaches allow to answer queries such as "is it possible for object o to intersect a query window q", they are not able to assign probabilities to these events conforming to possible worlds semantics.

Other approaches use independent probability density functions (pdf) at each point of time to model the uncertain positions of an object [17, 14, 12]. However, as shown in [15], this may produce wrong results (not in accordance with possible world semantics) for queries referring to a time interval because they ignore the temporal dependence between consecutive object positions in time. To capture such dependencies, recent approaches model the uncertain movement of objects based on stochastic processes. In particular, in [18, 15, 1, 19], trajectories are modeled based on Markov chains. This approach permits correct consideration of possible world semantics in the trajectory domain.

Nearest Neighbor Queries in Trajectory Databases. In the context of certain trajectory databases there is not a common definition of nearest neighbor queries, but rather a set of different interpretations. In [5], given a query trajectory (or spatial point) q and a time interval T, a NN query returns either the trajectory from the database which is closest to q during T or for each $t \in T$ the trajectory which is closest to q. The latter problem has also been addressed in [6]. Similarly, in [20], all trajectories which are nearest neighbors to q for at least one point of time t are computed.

Other approaches consider *continuous* nearest neighbor (CNN) semantics, definition of this query varies between publications [7, 21, 8]. CNN have also been addressed for objects with uncertain velocity and direction in [22]; the solutions proposed only find possible results, but not result probabilities. Solutions for road network data were also proposed for the case where the velocities of objects are unknown [23]. Furthermore, [14, 24] extended the problem of continuous kNN queries (on historical search) to an uncertain setting, serving as important preliminary work, however, based on a model which is not capable to return answers according to possible world semantics.

3. PROBLEM DEFINITION

A spatio-temporal database $\mathcal{D}$ stores triples (o_i, *time*, *location*), where o_i is a unique object identifier, *time* $\in \mathcal{T}$ is a point in time and *location* $\in \mathcal{S}$ is a position in space. Semantically, each such triple corresponds to an *observation* that object o_i has been seen at some *location* at some *time*. In $\mathcal{D}$, an object o_i can be described by 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*.

In this work, we assume a discrete time domain $\mathcal{T} = \{0, \ldots, n\}$. Thus, a trajectory becomes a sequence, i.e., a function on a discrete and ordinal scaled domain. Furthermore, we assume a discrete state space of possible locations (*states*): $\mathcal{S} = \{s_1, ..., s_{|\mathcal{S}|}\} \subset \mathbb{R}^d$, i.e., we use a finite alphabet of possible locations in a d-dimensional space. The way of discretizing space is application-dependent: for example, in traffic applications we may use road crossings, in indoor tracking applications we may use the positions of RFID trackers and rooms, and for free-space movement we may use a simple grid for discretization.

3.1 Uncertain Trajectory Model

Let $\mathcal{D}$ be a database containing the trajectories of $|\mathcal{D}|$ uncertain moving objects $\{o_1, ..., o_{|\mathcal{D}|}\}$. For each object o in $\mathcal{D}$ we store a set of observations $\Theta^o = \{\langle t_1^o, \theta_1^o\rangle, \langle t_2^o, \theta_2^o\rangle, \ldots, \langle t_{|\Theta^o|}^o, \theta_{|\Theta^o|}^o\rangle\}$ where $t_i^o \in \mathcal{T}$ denotes the time and $\theta_i^o \in \mathcal{S}$ the location of observation Θ_i^o. W.l.o.g. let $t_1^o < t_2^o < \ldots < t_{|\Theta^o|}^o$. Note that the location of an observation is assumed to be certain, while the location of an object between two observations is uncertain.

According to [15], we can interpret the location of an uncertain moving object o at time t as a realization of a random variable $o(t)$. Given a time interval $[t_s, t_e]$, the sequence of uncertain locations of an object is a family of correlated random variables, i.e., a stochastic process. This definition allows us to assess the probability of a possible trajectory, i.e., the realization of the corresponding stochastic process. In this work we follow the approaches from [15, 25, 19] and employ the first-order Markov chain model as a specific instance of a stochastic process. The state space of the model is the spatial domain $\mathcal{S}$. State transitions are defined over the time domain $\mathcal{T}$. In addition, the Markov chain model is based on the assumption that the position $o(t+1)$ of an uncertain object o at time $t+1$ only depends on the position $o(t)$ of o at time t. Clearly, this assumption is overly restrictive, as for example vehicles on a road network will never follow a first-order Markov chain. Such vehicles generally follow a best path (e.g. the shortest path or the path having the most beautiful landscape, etc.). Nevertheless, such a simplified model can, as we will see in our experimental evaluation, accurately model the set of possible trajectories that a vehicle

[4]http://infoblog.stanford.edu/2008/07/why-uncertainty-in-data-is-great-posted.html

may have taken between two discrete observations. Theoretically, this high accuracy can be explained by combining both observation information and the Markov model into a new model.

The probability $M^o_{ij}(t) := P(o(t+1) = s_j | o(t) = s_i)$ is the *transition probability* of a given object o from state s_i to state s_j at a given time t. Transition probabilities are stored in a matrix $M^o(t)$, called *transition matrix of object o at time t*. In general, every object o might have a different transition matrix, and the transition matrix of an object might vary over time. Further, let $\vec{s}^o(t) = (s_1, \ldots, s_{|\mathcal{S}|})^T$ be the distribution vector of a given single object o at time t, where $\vec{s}^o_i(t) = P(o(t) = s_i)$, i.e. each element of the vector describes o's probability of visiting the state s_i at time t. Without any further knowledge (from observations) the distribution vector $\vec{s}^o(t+1)$ can be inferred from $\vec{s}^o(t)$ by applying the following formula: $\vec{s}^o(t+1) = M^o(t)^T \cdot \vec{s}^o(t)$.

The traditional Markov model [15] uses forward probabilities only. In Section 5, we propose a Bayesian inference approach, to condition this *a-priori Markov chain* to an adapted *a-posteriori Markov chain* which also considers all observations of an object.

3.2 Nearest Neighbor Queries

In this work we consider three types of time-parameterized NN queries that take as input a certain reference state or trajectory q and a set of timesteps T. Note that q can be both a state or a trajectory, since a query state is simply a trivial query trajectory.

DEFINITION 1 (P∃NN QUERY). *A probabil. ∃ nearest neighbor query retrieves all objects $o \in \mathcal{D}$ which have a sufficiently high probability to be the nearest neighbor of q for at least one point of time $t \in T$, formally:*

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

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

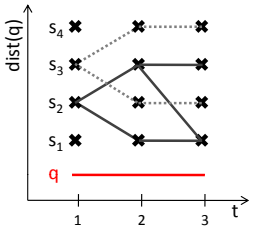
This definition is a extension of the spatio-temporal query proposed in [5] to the case of uncertainty. In the running taxi-tracking application mentioned in the introduction, the parameter T may correspond to the duration of a bank robbery, and q may correspond to the (constant) location of the bank, or the observed trajectory of the vehicle of the escaping robbers. In this application, a $P\exists NNQ(q, \mathcal{D}, T, \tau)$ query returns all taxis having a probability of at least τ of having been the closest cab at any time during the robbery, and thus, of possibly having observed something relevant.

In addition, we consider NN queries with the ∀ quantifier, which have also been proposed in [5] for crisp trajectory data.

DEFINITION 2 (P∀NN QUERY). *A probabil. ∀ nearest neighbor query retrieves all objects $o \in \mathcal{D}$ which have a sufficiently high probability ($P\forall NN$) to be the nearest neighbor of q for the entire set of timestamps T, formally:*

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

In the running taxi-tracking application $P\forall NNQ(q, \mathcal{D}, T, \tau)$ returns all taxis having a probability of at least τ of having been the closest cab during the whole robbery, and thus, of possibly having observed the whole crime scene. The main difference between Definition 1 and Definition 2 is that a $P\exists NNQ(q, \mathcal{D}, T, \tau)$ requires



object	trajectory	P(tr)
o_1 —	$tr_{1,1} = s_2, s_1, s_1$	0.5
o_1 —	$tr_{1,2} = s_2, s_3, s_1$	0.25
o_1 —	$tr_{1,3} = s_2, s_3, s_3$	0.25
o_2 ····	$tr_{2,1} = s_3, s_2, s_2$	0.5
o_2 ····	$tr_{2,2} = s_3, s_4, s_4$	0.5

Figure 1: Example uncertain trajectories

a candidate object to be the nearest-neighbor of q for at least one point of time in T to qualify as a result, while $P\forall NNQ(o, q, \mathcal{D}, T)$ requires a candidate object to remain the nearest-neighbor for the whole duration of T. In addition to these semantics for probabilistic nearest neighbor queries we now introduce a *continuous* query type which intuitively extends the spatio-temporal continuous nearest-neighbor query [21, 8] to apply on uncertain trajectories.

DEFINITION 3 (PCNN QUERY). *A probabilistic continuous nearest neighbor query retrieves all objects $o \in \mathcal{D}$ together with the set of timesets $\{T_i\}$ where in each T_i the object has a sufficiently high probability to be always the nearest neighbor of $q(t)$, formally:*

$$PCNNQ(q, \mathcal{D}, T, \tau) =$$
$$\{(o, T_i) : o \in \mathcal{D}, T_i \subseteq T, P\forall NN(o, q, \mathcal{D}, T_i) \geq \tau\}.$$

Analogously to the CNN query definition [21, 8], in order to reduce redundant answers it makes sense to redefine the PCNN Query where we focus on results that maximize $|T_i|$, formally:

$$PCNNQ(q, \mathcal{D}, T, \tau) =$$
$$\{(o, T_i) : o \in \mathcal{D}, T_i \subseteq T, P\forall NN(o, q, \mathcal{D}, T_i) \geq \tau$$
$$\wedge \forall T_j \supset T_i : P\forall NN(o, q, \mathcal{D}, T_j) < \tau\}.$$

Note that according to this definition result sets $T_i \subseteq T$ do not have to be connected. In the taxi-tracking application, a $PCNNQ$ allows to find the set of time intervals in T where a taxi has a sufficiently high probability of being a witness. Such results allow to find groups of taxi drivers having a high probability of having witnessed the same part of the crime scene, in order to synchronize the evidence of multiple witnesses. To summarize, we have defined three nearest-neighbor semantics for uncertain spatio-temporal data. All these semantics are inspired by corresponding nearest-neighbor semantics on certain trajectories, as defined in [5, 21, 8].

EXAMPLE 1. *To illustrate the three query types, consider the scenario shown in Figure 1 consisting of a query trajectory and two uncertain database objects $\mathcal{D} = \{o_1, o_2\}$ in a discretized space and time domain. For simplicity, whenever an object has two alternatives for choosing a possible state transition, each transition is assumed to have a probability of 0.5. These probabilities define the Markov chains of o_1 and o_2. Thus o_1 has three possible trajectories and o_2 has two possible trajectories, the probabilities of which are also shown in Figure 1. Using possible worlds semantics, any PNN query can naively be computed by considering all six possible combinations $(tr_{1,i}, tr_{2,j})$, $i \in \{1, 2, 3\}$, $j \in \{1, 2\}$, called possible worlds, of possible trajectories of objects o_1 and o_2. The total probability of all possible worlds where o_2 is closer to q than o_1 at any time, by definition, equals the probability $P\exists NN(o_2, q, \mathcal{D}, \{1, 2, 3\})$. For this example these possible worlds are $(tr_{1,2}, tr_{2,1})$ and $(tr_{1,3}, tr_{2,1})$. Assuming object independence, $P(tr_{1,i}, tr_{2,j})$ of a possible world is given by the*

product $P(tr_{1,i}) \cdot P(tr_{2,j})$ *yielding* $P\exists NN(o_2, q, \mathcal{D}, \{1,2,3\}) = P(tr_{1,2}) \cdot P(tr_{2,1}) + P(tr_{1,3}) \cdot P(tr_{2,1}) = 0.25 \cdot 0.5 + 0.25 \cdot 0.5 = 0.25$. *Accordingly the probability* $P\forall NN(o_1, q, \mathcal{D}, \{1,2,3\}) = 0.75$ *can be computed by the sum of the probabilities* $P(tr_{1,1}, tr_{2,1})$, $P(tr_{1,1}, tr_{2,2})$, $P(tr_{1,2}, tr_{2,2})$ *and* $P(tr_{1,3}, tr_{2,2})$ *of worlds where* o_1 *is always closer to* q *than* o_2. *A* $PCNNQ(q, \mathcal{D}, \{1,2,3\}, 0.1)$ *will return the object* o_1 *together with the interval* $\{1,2,3\}$ *and* o_2 *together with the interval* $\{2,3\}$, *as in these intervals, the respective objects have a probability of at least* 0.1 *to be closest to* q.

In this example, exact probabilities are computed by explicit consideration of all possible worlds. However, since the number of possible trajectories grows exponentially large in the number of time transitions, and the total number of possible worlds is furthermore exponential in the number of objects, the challenge of this work is to find a more efficient approach to compute the same nearest-neighbor probabilities without enumeration of all possible worlds.

3.3 Query Evaluation Framework, Roadmap

An intuitive way to evaluate a PNN query is to compute for *every* $o \in \mathcal{D}$ the probability P∃NN or P∀NN. However, to speed up query evaluation, in Section 6, we show that it is possible to prune some objects from consideration using an index over $\mathcal{D}$. Then, for each remaining object o, we have to compute a probability (i.e., P∃NN or P∀NN) and compare it to the threshold τ. In Section 4, we show that computing the P∃NN query and P∀NN query is prohibitively expensive. To solve this problem, in Section 5, we present a general sampling-based approximate but efficient solution to solve all types of PNN queries. As discussed in this section, P∃NN and P∀NN can be approximated by Monte-Carlo simulation: for each object $o' \in \mathcal{D}$ a trajectory is generated which conforms to both the Markov chain model $M^{o'}$ and the observations $\Theta^{o'}$ and all these trajectories are used to model a possible world. By performing the NN query in all these possible worlds and averaging the results, we are able to derive an approximate result probability.

4. THEORETICAL ANALYSIS

This section theoretically studies the runtime complexity of the P∃NNQ, P∀NNQ and PCNNQ queries.

4.1 The P∃NN Query

In a P∃NNQ query, for any candidate object $o \in \mathcal{D}$, Definition 1 requires the probability $P\exists NN(o, q, \mathcal{D}, T)$. However, the following lemma shows that this probability is hard to compute.

LEMMA 1. *The computation of* $P\exists NN(o, q, \mathcal{D}, T)$ *is NP-hard.*

PROOF. $P\exists NN(o, q, \mathcal{D}, T)$ is equal to $1 - P(\neg\exists t \in T, \forall o' \in \mathcal{D} \setminus o : d(q(t), o(t)) \leq d(q(t), o'(t)))$. We will show that deciding if there exists a possible world for which the expression:

$$\neg\exists t \in T, \forall o' \in \mathcal{D} \setminus o : d(q(t), o(t)) \leq d(q(t), o'(t)) \quad (1)$$

is satisfied is an NP-hard problem. (Note that this is a much easier problem than computing the actual probability.) Specifically, we will reduce the well-known NP-hard k-SAT problem to the problem of deciding on the existence of a possible world for which Expression 1 holds.

For this purpose, we provide a mapping to convert a boolean formula in conjunctive normal form to a Markov chain modeling the decision problem of Expression 1 in polynomial time. Thus, if the decision problem could be computed in PTIME, then k-SAT could also be solved in PTIME, which would only be possible if P=NP. A k-SAT expression E is based on a set of boolean variables $X = \{x_1, x_2, \ldots, x_n\}$. The *literal* l_i of a variable x_i is either x_i or $\neg x_i$ and a *clause* $c = \bigvee_{x_i \in \mathbb{C}} l_i$ is a disjunction of literals where $\mathbb{C} \subseteq X$ and $|\mathbb{C}| < k$. Then E is defined as a conjunction of clauses: $E = c_1 \wedge c_2 \wedge \ldots \wedge c_m$.

For our mapping, we will consider a simplified version of the P∃NN problem, specifically (1) q is a certain point, (2) o is a certain point and (3) the state space $\mathcal{S}$ of possible locations only includes 4 states. As illustrated in Figure 2, compared to o, states s_1 and s_2 are closer to q and states s_3 and s_4 are further from q.[5] Therefore, if an uncertain object is at states s_1 or s_2 then o is not the NN of q.

In our mapping, each variable $x_i \in X$ is equivalent to one uncertain object $o'_i \in \mathcal{D} \setminus o$. Furthermore each disjunctive clause c_j is interpreted as an event happening at time $t = j$, i.e., the event c_1 happens at time $t = 1$, c_2 happens at time $t = 2$ etc. Each clause c_j can be seen as a disjunctive event that at least one object o'_i at time $t = j$ is closer to q than o (in this case, c_j is $true$). Therefore, the conjunction of all these events, i.e. expression $E = \bigwedge_{1 \leq j \leq m} c_j$, becomes true if the set of variables is chosen in a way that at each point in time, compared to o, at least one object is closer to q; this directly represents Expression 1. However, in k-SAT, not every variable x_i (corresponding to o'_i) is contained in each term c_j which does not correspond to our setting, since an uncertain object has to be *somewhere* at each point in time. To solve this problem, we extend each clause c_j, such that each variable x_i is contained in c_j, without varying the semantics of c_j. Let us assume that x_i is not contained in c_j. Then $c'_j = c_j \vee false = c_j \vee (x_i \wedge \neg x_i)$. This means that we can assume that object o'_i is definitely not closer to q than o at time t.

Let l_i^j be the literal of variable x_i in clause c_j. Based on the above discussion, we are able to construct for each object o'_i two possible trajectories (worlds). The first one, based on the assumption that x_i is true, transitions between states s_2 (if l_i^j = true) and s_4 (if l_i^j = false). The second one, based on the assumption that x_i is set to false, transitions between states s_1 (if l_i^j = true) and s_3 (if l_i^j = false). Since these two trajectories can never be in the same state it is straightforward to construct a time-inhomogeneous Markov chain $M^o(t)$ for each object o'_i and each timestamp j.

After the Markov chains for each uncertain object o'_i in $\mathcal{D}$ have been determined, we would just have to traverse them and compute the probability $P\exists NN(o, q, \mathcal{D}, T)$. If this probability is < 1, there would exist a solution to the corresponding k-SAT formula. However it is not possible to achieve this efficiently in the general case as long as $P \neq NP$. Therefore computing $P\exists NN$ in subexponential time is impossible. □

Example: Consider a set of boolean variables $X = \{x_1, \ldots, x_4\}$ and the following formula:

$$E = (\neg x_1 \vee x_2 \vee x_3) \wedge (x_2 \vee \neg x_3 \vee x_4) \wedge (x_1 \vee \neg x_2)$$

Therefore, we have

$c_1 = (\neg x_1 \vee x_2 \vee x_3)$, $c_2 = (x_2 \vee \neg x_3 \vee x_4)$ and $c_3 = (x_1 \vee \neg x_2)$

By employing the mapping discussed above, we get the four inhomogeneous Markov chains illustrated in Figure 2. For instance, under the condition that x_1 is set to $true$, the value of the literal $\neg x_1$ is false at $t = 1$ (in clause c_1) such that o'_1 starts in the state s_4. On the other hand, if x_1 is set to $false$, then o'_1 starts in the state s_1.

[5]The states of o and q are omitted for the sake of simplicity.

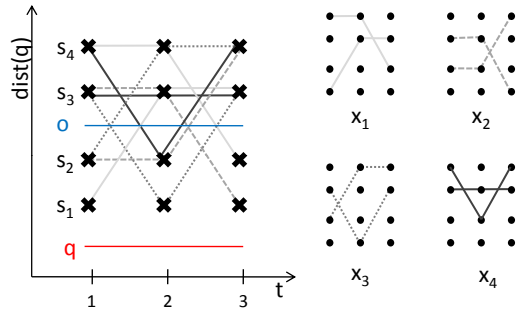


Figure 2: An example instance of our mapping of the 3-SAT problem to a set of Markov chains.

In the second clause c_2, since $x_1 \notin \mathbb{C}_2$, the position of o_1' must not affect the result. Therefore, for both cases $x_1 = false$ and $x_1 = true$, o_1' must be behind o. In the last clause c_3, if $x_1 = true$ the object moves to state s_2. On the other hand, if $x_1 = false$, the object moves to state s_3.

4.2 The P∀NN Query

Theoretically showing the complexity of the P∀NNQ is more difficult than the analysis of the P∃NN query; the actual complexity of this type remains unknown. In the following we provide insights why P∀NN probabilities can not be computed efficiently. Let $o \prec_q^T o_a$ denote the random predicate that is true iff object o is closer to q than object $o_a \in \mathcal{D}$ during the query time $T = [t_{start}, t_{end}]$, i.e. $\forall t \in T : d(o(t), q(t)) \leq d(o_a(t), q(t))$. If $o \prec_q^T o_a$ holds, we say that o *dominates* o_a with respect to q during T. If the parameters q and T are clear from the context, we simply say that o dominates o_a. Again we start our analysis by considering the single object probability $P\forall NN(o, q, \{o_a\}, T)$. This probability can be computed correctly and efficiently.

LEMMA 2. *The probability $P(o \prec_q^T o_a)$ that o dominates o_a can be computed in PTIME.*

PROOF. We here provide just a basic proof sketch. In a nutshell, the idea of this proof is to treat objects o and o_a as a single joint random variable having the joint transition matrix $J(t)$ defined on the state space $S \times S$. Starting at $t = t_{start}$, time transitions of $J(t)$ are performed iteratively. In each iteration, any entry of $J(t)$ corresponding to a possible world where o does not dominate o_a are set to zero. At time t_{end}, the total probability of remaining worlds in $J(t_{end})$ equals the probability that o dominates o_a over the whole duration of T. □

Since this probability can be computed efficiently, we now address how to compute $P\forall NN(o, q, \mathcal{D}, T)$ for a whole database efficiently, given that we can adapt the model of o to the domination relation $o \prec_q^T o_a$.

LEMMA 3. *Given that a model $(M_{i,j}^{post.}(t)) = P(o(t+1) = s_i | o(t) = s_j, o \prec_q^T o_a)$ can be computed in PTIME, the probability $P\forall NN(o, q, \mathcal{D}, T)$ can be computed in PTIME.*

PROOF. Again, a formal proof of this theorem can be found in the extended version of this paper [26]. Here we will provide the main idea of the proof. By definition of predicate $(o \prec_q^T o_a)$, we can rewrite the probability that o is the nearest neighbor of q during T as the following conjunctive formula: $P\forall NN(o, q, \mathcal{D}, T) =$

$$P(\forall o_a \in \mathcal{D} : o \prec_q^T o_a) = P(\bigwedge_{o_a \in \mathcal{D}} o \prec_q^T o_a). \quad (2)$$

Clearly, Equation 2 follows from the fact that o is the NN of q if and only if o is closer to q than *all* other objects in $\mathcal{D}$ during time T. Using the chain rule of probability, which iteratively uses the rule $P(A \wedge B) = P(A) \cdot P(B|A)$ for conditional probabilities, we obtain

$$P(\bigwedge_{o_a \in \mathcal{D}} o \prec_q^T o_a) = \prod_{1 \leq a \leq \mathcal{D}} P(o \prec_q^T o_a | \bigwedge_{j<a} o \prec_q^T o_j). \quad (3)$$

The first factor $P(o \prec_q^T o_1)$ is given by Lemma 2. Now, by employing the precondition of Lemma 3, we obtain an adapted model of o given $o \prec_q^T o_1$. These two steps start the induction. Now, given an adapted model of o given $o \prec_q^T o_1, ..., o \prec_q^T o_k$, we can reapply Lemma 2 to compute $P(o \prec_q^T o_{k+1} | o \prec_q^T o_1, ..., o \prec_q^T o_k)$ that the model of o, which has been adapted to dominate $o_1, ..., o_k$, dominates o_{k+1}. Next, we apply the precondition of Lemma 3 to take the current model of o, which has already been adapted to $o \prec_q^T o_1, ..., o \prec_q^T o_k$, and further adapt this model to $o \prec_q^T o_{k+1}$. Since both Lemma 2 and the precondition of Lemma 3 can be computed in PTIME, $P(\bigwedge_{o_a \in \mathcal{D}} o \prec_q^T o_i)$, which requires to apply Lemma 2 and the precondition of Lemma 3 exactly $|\mathcal{D}|$ times, Lemma 3 can be computed in PTIME. □

We still have to condition the a-priori model $M_{k,i}^{prior}(t-1) = P(o(t) = s_i | o(t-1) = s_k)$ of an object o to the event that o dominates another object $o_a \in \mathcal{D}$, yielding model $M_{k,i}^{post.}(t-1) = P(o(t) = s_i | o(t-1) = s_k, o \prec_q^T o_a)$. The problem here is, as we will see, that the adapted model does not fulfil the Markov property, resulting in either exponential runtime or incorrect solutions.

To compute $P(o(t) = s_i | o(t-1) = s_k, o \prec_q^T o_a), s_i, s_k \in \mathcal{S}$, the idea is to treat the positions of o and o_a as a single joint stochastic process, having possible alternatives in $\mathcal{S}^2$. Then, the joint a-priori transition matrix $M_{o \times o_a}(t)$ is conditioned to the event $o \prec_q^T o_a$ following the forward-backward paradigm similar to the forward-backward approach used for sampling in Section 5. As a result, we get a joint probability matrix $M_{o \times o_a}(t-1) =$

$$P(o(t) = s_i, o_a(t) = s_j | o(t-1) = s_k, o_a(t-1) = s_l, o \prec_q^T o_a)$$

Finally, in order to keep the complexity of this algorithm sub-exponential, we have to reduce this joint transition matrix to an adapted transition matrix $M_{k,i}^{post.}(t-1) = P(o(t) = s_i | o(t-1) = s_k, o \prec_q^T o_a)$ of object o. By applying the law of conditional probability, it can be shown that:

$$M_{k,i}^{post.}(t-1) = \sum_{s_j} \sum_{s_l} P(o_a(t-1) = s_l | o(t-1) = s_k, o \prec_q^T o_a) *$$

$$P(o(t) = s_i, o_a(t) = s_j | o(t-1) = s_k, o_a(t-1) = s_l, o \prec_q^T o_a)$$

Assuming that the Markov property still holds, we should get the same results for

$$P(o(t) = s_i | o(t-1) = s_k, ..., o(t-n) = s_{t-n}, o \prec_q^T o_a) =$$

$$\sum_{s_j} \sum_{s_l} P(o_a(t-1) = s_l | o(t-1) = s_k, ..., o(t-n) = s_{t-n}, o \prec_q^T o_a) *$$

$$P(o(t) = s_i, o_a(t) = s_j | o(t-1) = s_k, o_a(t-1) = s_l, o \prec_q^T o_a)$$

which is clearly not equivalent, i.e. the Markov property does not hold on the reduced transition matrices and hence the algorithm has exponential complexity. Therefore, in Section 5, we will propose to use sampling to compute $P\forall NN(o, q, \mathcal{D}, T)$ probabilities efficiently.

4.3 The PCNN Query

The traditional CNN query [21, 8], retrieves the nearest neighbor of every point on a given query trajectory in a time interval T. This basic definition usually returns $m << |T|$ time intervals together having the same nearest neighbor. The main issue when considering uncertain trajectories and extending the query definition is the possibly large number of results due to highly overlapping and alternating result intervals. In particular, considering Definition 3, a PCNN result may have an exponential number of elements when τ becomes small. This is because in the worst case each $T_i \subseteq T$ can be associated with an object o for which the probability $P\forall NN(o, q, \mathcal{D}, T_i) \geq \tau$, i.e., 2^T different T_i's occur in the result set.

To alleviate (but not solve) this issue, in the following we propose a technique based on Apriori pattern mining to return the subsets of T that have a probability greater than τ. This algorithm requires to compute a $P\forall NN$ probability in each validation step; we assume that this is achieved by employing the sampling approach proposed in Section 5. Since each subset of T may have a probability greater than τ (especially when τ is chosen too small), a worst-case of $O(2^n)$ validations may have to be performed.

Algorithm. Algorithm 1 shows how to compute, for a query trajectory q, a time interval T, a probability threshold τ, and an uncertain trajectory $o \in \mathcal{D}$ all $T_i \subseteq T$ for which o is the nearest neighbor to q at all timestamps in T_i with probability of at least τ, and the corresponding probabilities.

Algorithm 1 $PC_\tau NN(q, o, \mathcal{D}, \mathrm{T}, \tau)$

1: $L_1 = \{(\{t\}, P) | t \in T \wedge P = P\forall NN(o, q, \mathcal{D} \setminus \{o\}, \{t\}) \geq \tau\}$
2: **for** $k = 2; L_{k-1} \neq \emptyset; k++$ **do**
3: $\quad X^k = \{T_k \subseteq T | |T_k| = k \wedge \forall T'_{k-1} \subset T_k \exists (T'_{k-1}, P) \in L_{k-1}\}$
4: $\quad L_k = \{(T_k, P) | T_k \in X^k \wedge P = P\forall NN(o, q, \mathcal{D} \setminus \{o\}, T_k) \geq \tau\}$
5: **end for**
6: **return** $\bigcup_k L_k$

We take advantage of the anti-monotonicity property that for a T_i to qualify as a result of the PCNNQ query, all proper subsets of T_i must also satisfy this query. In other words if o is the P$\forall$NN of q in T_i with probability at least τ, then for all $T_j \subset T_i$ o must be the P$\forall$NN of q in T_j with probability at least τ. Exploiting this property, we adapt the Apriori pattern-mining approach from [27] to solve the problem as follows. We start by computing the probabilities of all single points of time to be query results (line 1). Then, we iteratively consider the set X^k of all timestamp sets with k points of time by extending timestamp sets T_{k-1} with an additional point of time $t \in T \setminus T_{k-1}$, such that all $T'_{k-1} \subset T_k$ have qualified at the previous iteration, i.e., we have $P\forall NN(o, q, \mathcal{D} \setminus \{o\}, T'_{k-1})) \geq \tau$ (line 3).

The P$\forall$NN probability is monotonically decreasing with the number of points in time considered, i.e., P$\forall$NN$(o, q, \mathcal{D} \setminus \{o\}, T_k) \geq$ P$\forall$NN$(q, \mathcal{D}, T_{k+1})$ where $T_k \subset T_{k+1}$. Therefore we do not have to further consider the set of points of time T_k that do not qualify for the next iterations during the iterative construction of sets of time points. Based on the sets of timesteps T_k constructed in each iteration we compute $P\forall NN(o, q, \mathcal{D} \setminus \{o\}, T_k)$ to build the set of results of length k (line 4) that are finally collected and reported as result in line 6. The basic algorithm can be sped up by employing the property that given P$\forall$NN$(o, q, \mathcal{D} \setminus \{o\}, T_1) = 1$ the probability of P$\forall$NN$(o, q, \mathcal{D} \setminus \{o\}, T_1 \cup T_2) =$ P$\forall$NN$(o, q, \mathcal{D} \setminus \{o\}, T_2)$.

Based on Algorithm 1 it is possible to define a straightforward algorithm for processing PCNNQ queries (by considering each object o' from the database). Again this approach can be improved by the use of an appropriate index-structure (cf. Section 6).

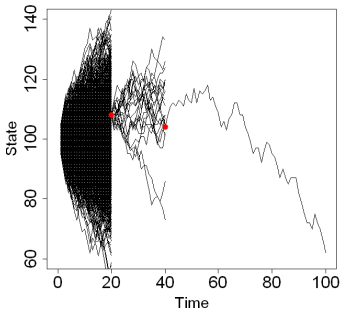


Figure 3: Traditional MC-Sampling.

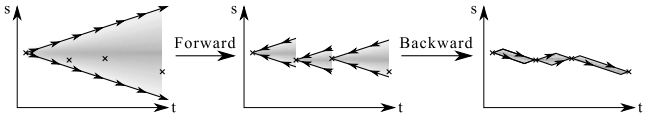


Figure 4: An overview over our forward-backward-algorithm.

5. SAMPLING POSSIBLE TRAJECTORIES

Based on the discussion in the previous sections, it is clear that answering probabilistic queries over uncertain trajectory databases has high run-time cost. Therefore, like previous work [28], we now study sampling-based approximate solutions to improve query efficiency.

5.1 Traditional Sampling

To sample possible trajectories of an object, a traditional Monte-Carlo approach would start by taking the first observation of the object, and then perform forward transitions using the a-priori transition matrix. This approach however, cannot directly account for additional observations for latter timestamps. Figure 3 illustrates a total of 1000 samples drawn in a one-dimensional space. Starting at the first observation time $t = 0$, transitions are performed using the a-priori Markov chain. At the second observation at time $t = 20$, the great majority of trajectories becomes inconsistent. Such impossible trajectories have to be dropped. At time $t = 40$, even more trajectories become invalid; After this observation, only one out of a thousand samples remains possible and useful.

Clearly, the number of trajectory generations required to obtain a single valid trajectory sample increases exponentially in the number of observations of an object, making this traditional Monte-Carlo approach inappropriate in obtaining a sufficient number of valid samples within acceptable time.

5.2 Efficient and Appropriate Sampling

To tackle the disadvantages of traditional sampling, we now introduce an optimized approach af drawing samples. On these samples, traditional NN algorithms for (certain) trajectories ([5, 6, 20, 7, 21, 8]) can be used to estimate NN probabilities.

In a nutshell, our approach starts with the initial observation θ^o_1 at time t^o_1, and performs transitions for object o using the a-priori Markov chain of o until the final observation $\theta^o_{|\Theta^o|}$ at time $t_{|\Theta^o|}$ is reached. During this *Forward*-run phase, Bayesian inference is used to construct a time-reversed Markov model $R^o(t)$ of o at time t given observations in the past, i.e., a model that describes the probability

$$R^o_{ij}(t) := P(o(t-1) = s_j | o(t) = s_i, \{\theta^o_i | t^o_i < t\})$$

of coming from a state s_j at time $t-1$, given being at state s_i at time t and the observations in the past. Then, in a second *Backward*-run phase, our approach traverses time backwards, from time $t_{|\Theta^o|}$

to t_1, by employing the time-reversed Markov model $R^o(t)$ constructed in the forward phase. Again, Bayesian inference is used to construct a new Markov model $F^o(t-1)$ that is further adapted to incorporate knowledge about observations in the future. This new Markov model contains the transition probabilities

$$F^o_{ij}(t-1) := P(o(t) = s_j | o(t-1) = s_i, \Theta^o). \quad (4)$$

for each point of time t, given all observations, i.e., in the past, the present and the future.

As an illustration, Figure 4(left) shows the initial model given by the a-priori Markov chain, using the first observation only. In this case, a large set of (*time*, *location*) pairs can be reached with a probability greater than zero. The adapted model after the forward phase (given by the a-priori Markov chain and all observations), depicted in Figure 4(center), significantly reduces the space of reachable (*time*, *location*) pairs and adapts respective probabilities. The main goal of the forward-phase is to construct the necessary data structures for efficient implementation of the backward-phase, i.e., $R^o(t)$. This task is not trivial, since the Markov property does not hold for the future, i.e., the past is *not* conditionally independent of the future given the present. Figure 4(right) shows the resulting model after the backward phase. In the following section, both phases are elaborated in detail.

Note that the Baum-Welch algorithm for hidden Markov models (HMMs) is similar to the proposed algorithm. This algorithm aims at estimating *time-invariant* transition matrices and emission probabilities of a hidden Markov model. In contrast, we assume this underlying model to be given, however we aim at adapting it by computing *time-variant* transition matrices. Despite these differences, the above algorithm could also be proven by showing that our model is a special case of a HMM and deducting the algorithm from the Baum-Welch algorithm [30]. Related to our algorithm is also the Forward-Backward-Algorithm for HMMs that aims at computing the *state distribution* of an HMM for each point in time. In contrast, we aim at computing transition matrices for each point in time, given a set of observations.

5.2.1 Forward-Phase

First note that, for the algorithm to work, it is necessary that observations are non-contradicting. To obtain the backward transition matrix $R^o(t)$, we can apply the theorem of Bayes as follows:

$$R^o(t)_{ij} := P(o(t-1) = s_j | o(t) = s_i) = \quad (5)$$

$$\frac{P(o(t) = s_i | o(t-1) = s_j) \cdot P(o(t-1) = s_j)}{P(o(t) = s_i)}$$

Computing $R^o(t)_{ij}$ is based on the a-priori Markov chain only, and does not consider any information provided by observations. To incorporate knowledge about past observations into $R^o(t)_{ij}$, let $past^o(t) := \{\theta^o_i | t^o_i < t\}$ denote the set of observations temporally preceding t. Also, let $prev^o(t) := argmax_{\Theta^o_i \in past^o(t)} t^o_i$ denote the most recent observation of o at time t. Given all past observations, Equation 5 becomes conditioned as follows:

LEMMA 4.

$$R^o(t)_{ij} := P(o(t-1) = s_j | o(t) = s_i, past^o(t)) = \quad (6)$$

$$\frac{P(o(t) = s_i | o(t-1) = s_j, past^o(t)) \cdot P(o(t-1) = s_j | past^o(t))}{P(o(t) = s_i | past^o(t))}$$

PROOF. Equation 6 uses the conditional theorem of Bayes $P(A|B,C) = \frac{P(B|A,C) \cdot P(A|C)}{P(B|C)}$, the correctness of which is shown in the extended version of this paper ([26]). □

The conditional probability $P(o(t) = s_i | o(t-1) = s_j, past^o(t))$ can be rewritten as $P(o(t) = s_i | o(t-1) = s_j)$, exploiting the Markov property and the assumption of non-contradiction between model and observation.

By exploiting the Markov property, the priors $P(o(t-1) = s_j | past^o(t))$ and $P(o(t) = s_i | past^o(t))$ can both be rewritten as $P(o(t-1) = s_j | prev^o(t))$ and $P(o(t) = s_i | prev^o(t))$ respectively as long as o was not observed at time t; i.e., given the position at some time t, the position at a time $t^+ > t$ is conditionally independent of the position at any time $t^- < t$. If o was observed at time t, this probabilty is trivially given by the observation. Thus, Equation 6 can be rewritten as $R^o(t)_{ij} =$

$$\frac{P(o(t) = s_i | o(t-1) = s_j) \cdot P(o(t-1) = s_j | prev^o(t))}{P(o(t) = s_i | prev^o(t))} \quad (7)$$

The probability $P(o(t) = s_i | o(t-1) = s_j)$ is given directly by the definition of the a-priori Markov chain $M^o(t)$ of o. Both priors $P(o(t-1) = s_j | prev^o(t))$ and $P(o(t) = s_i | prev^o(t))$ can be computed by performing time transitions from observation $prev^o(t)$, also using the a-priori Markov chain $M^o(t)$. For each element $r_{ij} \in R^o_{ij}(t)$, and each point of time $t \in [t_1, t_{|\Theta^o|}]$, these priors can be computed in a single run, iteratively performing transitions from t_1 to $t_{|\Theta^o|}$. During this run, all backward probabilities $P(o(t-1) = s_j | o(t) = s_i, past^o(t))$ are computed using Equation 7 and memorized in the inhomogeneous matrix $R^o(t)$. During any iteration of the forward algorithm, where a new observation $present^o(t) := \Theta^o_t \in \Theta^o$ is reached, the information of this observation has to be incorporated into the model. This is done trivially, by setting $P(o(t) = s_i | past^o(t), present^o(t))$ to one if s_i is the state θ observed by $present^o(t)$ and to zero otherwise.

5.2.2 Backward Phase

During the backward phase, we traverse time backwards using the reverse transition matrix $R^o(t)$, to propagate information about future observations back to past points of time, as depicted in Figure 4(c). During this traversal, we again obtain a time reversed matrix $F^o(t)$, describing state transitions between adjacent points of time, given observations in the future. Due to this second reversal of time, matrix $F^o(t)$ also contains adapted transition probabilities in the forward direction of time. Thus, matrix $F^o(t)$ represents a Markov model which corresponds to the desired a-posteriori model: It contains the probabilities of performing a state transition between state s_i and s_j at time t to time $t+1$, incorporating knowledge of observations in both the past and the future. In contrast, the a-priori Markov model $M^o(t)$ only considers past observations. We now discuss the details of this phase.

LEMMA 5. *The following reverse Markov property holds for each element R^o_{ij} of R^o:*

$$P(o(t) = s_j | o(t+1) = s_i, o(t+2) = s_{t+2}, ..., o(t+k) = s_{t+k}) =$$

$$P(o(t) = s_j | o(t+1) = s_i) \quad (8)$$

This reverse markov property allows us to traverse the time domain backward equivalent to a forward traversal. As an initial state for the backward phase, we use the state vector corresponding to the final observation $\Theta^o_{|\Theta^o|}$ at time $t^o_{|\Theta^o|}$ at state $\theta^o_{|\Theta^o|}$. This way, we take the final observation as given, making any further probabilities that are being computed conditioned to this observation. At each point of time $t \in [t_{|\Theta^o|}, t_1]$ and each state $s_i \in S$, we compute the probability that o is located at state s_i at time t *given* (conditioned to the event) that the observations $future^o(t) := \{\theta^o_i | t^o_i > t)\}$ at times later than t are made. Let $next^o(t) = argmin_{\Theta^o_i \in future^o(t)}(t^o_i)$

denote the soonest observation of o after time t. To obtain $F^o(t)$, we once again exploit the theorem of Bayes:

$$F^o_{ij}(t) := P(o(t+1) = s_j | o(t) = s_i, \Theta^o) =$$

$$\frac{P(o(t) = s_i | o(t+1) = s_j, \Theta^o) \cdot P(o(t+1) = s_j | \Theta^o)}{P(o(t) = s_i | \Theta^o)} \quad (9)$$

By exploiting the reverse Markov property (c.f. Equation 8), we can rewrite $P(o(t) = s_i | o(t+1) = s_j, \Theta^o) = P(o(t) = s_i | o(t+1) = s_j, past(t+1))$ which is given by matrix $R^o(t)$. Both priors $P(o(t+1) = s_j | \Theta^o)$ and $P(o(t) = s_i | \Theta^o)$ can be computed in the following way inductively. Given that we compute $P(o(t) = s_i | past(t), present(t))$ during the forward phase, the last transition of the forward phase yields $P(o(t_{end}) = s_i | \Theta^o)$. The remaining probabilities $P(o(t_k) = s_i | \Theta^o)$ can be computed by employing the Markov transitions in backward direction with matrix $R(t)$.

5.2.3 Sampling Process

Algorithm 2 AdaptTransitionMatrices(o)

1: {Forward-Phase}
2: $\vec{s}^o(t^o_1) = \theta^o_1$
3: **for** $t = t^o_1 + 1; t \leq t^o_{|\Theta^o|}; t$++ **do**
4: $X'(t) = M^o(t-1)^T \cdot diag(\vec{s}^o(t-1))$
5: $\forall i \in \{1 \ldots |\mathcal{S}|\} : \vec{s}^o(t)_i = \sum_{j=1}^{|S|} X'_{ij}(t)$
6: $\forall i,j \in \{1 \ldots |\mathcal{S}|\} : R^o(t)_{ij} = \frac{X'_{ij}(t)}{\vec{s}^o(t)_i}$
7: **if** $t \in \Theta^o$ **then**
8: $\vec{s}^o(t) = \theta^o_t$ {Incorporate observation}
9: **end if**
10: **end for**
11: {Backward-Phase}
12: **for** $t = t^o_{|\Theta^o|} - 1; t \geq t^o_1; t$-- **do**
13: $X'(t) = R^o(t+1)^T \cdot diag(\vec{s}^o(t+1))$
14: $\forall i \in \{1 \ldots |\mathcal{S}|\} : \vec{s}^o(t)_i = \sum_{j=1}^{|S|} X'_{ij}(t)$
15: $\forall i,j \in \{1 \ldots |\mathcal{S}|\} : F^o(t)_{ij} = \frac{X'_{ij}(t)}{\vec{s}^o(t)_i}$
16: **end for**
17: **return** F^o

Algorithm 2 summarizes the construction of the transition model for a given object o. In the forward phase, the new distribution vector $\vec{s}^o(t)$ of o at time t and backward probability matrix $R^o(t)$ at time t can be efficiently derived from the temporary matrix $X'(t)$, computed in Line 4. The equation is equivalent to a simple transition at time t, except that the state vector is converted to a diagonal matrix first. This trick allows to obtain a matrix describing the joint distribution of the position of o at time $t-1$ and t. Formally, each entry $X'(t)_{i,j}$ corresponds to the probability $P(o(t-1) = s_j \wedge o(t) = s_i | past^o(t))$ which is equivalent to the *numerator* of Equation 6.[6] To obtain the denominator of Eq. 6 we first compute the row-wise sum of $X'(t)$ in Line 5. The resulting vector directly corresponds to $\vec{s}^o(t)$, since for any matrix A and vector x it holds that $A \cdot x = rowsum(A \cdot diag(x))$. By employing this rowsum operation, only one matrix multiplication is required for computing $R^o(t)$ and $\vec{s}^o(t)$.

Next, the elements of the temporary matrix $X'(t)$ and the elements of $\vec{s}^o(t)$ are normalized in Equation 6, as shown in Line 6 of the algorithm.

[6]The proof for this transformation $P(A \cap B|C) = P(A|C) \cdot P(B|A,C)$ can be derived analogously to Lemma 4.

Finally, possible observations at time t are integrated in Line 8. In Lines 12 to 15, the same procedure is followed in time-reversed direction, using the backward transition matrix $R^o(t)$ to compute the a-posteriori matrix $F^o(t)$.

The overall complexity of this algorithm is $O(|T| \cdot |\mathcal{S}|^2)$. The initial matrix multiplication requires $|\mathcal{S}|^2$ multiplications. While the complexity of a matrix multiplication is in $O(|\mathcal{S}|^3)$, the multiplication of a matrix with a diagonal matrix, i.e., $M^T \cdot s$ can be rewritten as $M^T_i \cdot s_{ii}$, which is actually a multiplication of a vector with a scalar, resulting in an overall complexity of $O(|\mathcal{S}|^2)$. Rediagonalization needs $|\mathcal{S}|^2$ additions as well, such as re-normalizing the transition matrix, yielding $3 \cdot |T| \cdot |\mathcal{S}|^2$ for the forward phase. The backward phase has the same complexity as the forward phase, leading to an overall complexity of $O(|T| \cdot |\mathcal{S}|^2)$.

Once the transition matrices $F^o(t)$ for each point of time t have been computed, the actual sampling process is simple: For each object o, each sampling iteration starts at the initial position θ^o_1 at time t^o_1. Then, random transitions are performed, using $F^o(t)$ until the final observation of o is reached. Doing this for each object $o \in \mathcal{D}$, yields a (certain) trajectory database, on which exact NN-queries can be answered using previous work. Since the event that an object o is a $\forall$NN ($\exists$NN) of q is a binomial distributed random variable, we can use methods from statistics, such as the Hoeffding's inequality ([29]) to give a bound of the estimation error, for a given number of samples.

6. SPATIAL PRUNING

Pruning objects in probabilistic NN search can be achieved by employing appropriate index structures available for querying uncertain spatio-temporal data. In this work, we use the *UST-tree* [25]. In this section, we briefly summarize the index and show how it can be employed to efficiently prune irrelevant database objects, identify result candidates, and find influence objects that might affect the $\forall$NN probability of a candidate object.

The UST-Tree. Given an uncertain spatio-temporal object o, the main idea of the UST-tree is to conservatively approximate the set of possible (*location*, *time*) pairs that o could have possibly visited, given its observations Θ^o. In a first approximation step, these (*location*, *time*) pairs, as well as the possible (*location*, *time*) pairs defined by Θ^o_i and Θ^o_{i+1} are minimally bounded by rectangles. Such a rectangle, for observations Θ^o_i and Θ^o_{i+1} is defined by the time interval $[t^o_i, t^o_{i+1}]$, as well as the minimal and maximal longitude and latitude values of all reachable states.

EXAMPLE 2. *Consider Figure 5, where four objects objects A, B, C and D are given by three observations at time 0, 5 and 10. For each object, the set of possible states in the corresponding time intervals $[0,5]$ and $[5,10]$ is approximated by two minimum bounding rectangles. For illustration, the set of possible states at each point of time is also depicted by dashed rectangles.*

The UST-tree indexes the resulting rectangles using an R^*-tree ([31]). We now discuss how such an index structure can be used for the evaluation of P$\forall$NNQ and P$\exists$NNQ queries.

Pruning candidates of P$\forall$NNQ queries. For a P$\forall$NNQ query, an object must have a non-zero probability of being the closest object to q, for all timestamps in the query interval. As a consequence, to find candidate objects for the P$\forall$NNQ query, we have to consider for all objects $o \in \mathcal{D}$ whether for each $t \in q.T$ there does not exist an object $o' \in \mathcal{D}$ such that $d_{min}(o(t), q(t)) > d_{max}(o'(t), q(t))$. Here, $d_{min}(o(t), q(t))$ ($d_{max}(o(t), q(t))$) denotes the minimum (maximum) distance between the possible states of $o(t)$ and $q(t)$. Thus, the set of candidates $C_\forall(q)$ of a P$\forall$NNQ is defined as $C_\forall(q) =$

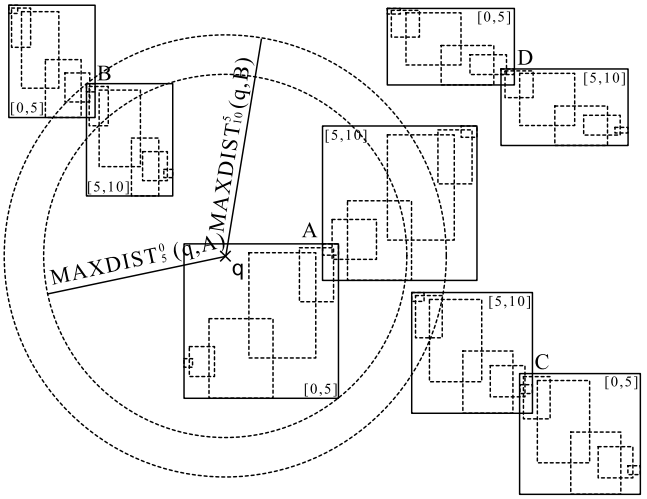


Figure 5: Spatio-Temporal Pruning Example.

$$\{o \in \mathcal{D} | \forall t \in q.T : d_{min}(o(t), q(t)) \leq min_{o' \in \mathcal{D}} d_{max}(o'(t), q(t))\}$$

Applying spatial pruning on the leaf level of the UST-tree, we have to apply the d_{min} and d_{max} distance computations on the minimum bounding rectangles on the leaf level in consideration of the time intervals associated with these leaf entries. In our example, given the query point q with $q.T = [2, 8]$, only object A is a candidate, since $d_{min}(q(t), A(t)) \leq d_{max}(q(t), o(t))$ for all $o \in \mathcal{D}$ in the time intervals [0,5] and [5,10], both together covering $q.T$. Objects B, C and D can be safely pruned.

It is important to note that pruned objects, i.e., objects not contained in $C_\forall(q)$ may still affect the ∀NN probability of other objects and even may prune other objects. For example, though object B is not a candidate, it affects the ∀NN probability of all other objects and contributes to prune possible worlds of object A, because $d_{max}(q(t), A(t)) > d_{min}(q(t), B(t))\ \forall t \in [5, 10]$. All objects having at at least one timestamp $t \in q.T$ a non-zero probability being the NN of q may influence the ∀NN probability of other objects. Since we need these objects for the verification step of both the exact and the sampling algorithms, we have to maintain them in an additional list $I_\forall(q) =$

$$\{o \in \mathcal{D} | \exists t \in T : d_{min}(o(t), q(t)) \leq min_{o' \in \mathcal{D}} d_{max}(o'(t), q(t))\}$$

To perform spatial pruning at the non-leaf level of the UST-tree, we can analogously apply d_{min} and d_{max} on the MBRs of the non-leaf level.

Pruning for the P∃NNQ query. Pruning for the P∃NNQ query is very similar to that for the P∀NNQ query. However, we have to consider that an object being the nearest neighbor for a single point in time is already a valid query result. Therefore, no distinction is made between *candidate*s and *influence object*s. Every pruner can be a valid result of the P∃NNQ query, such that each object with a d_{min} smaller than the pruning distance has to be refined. The remaining procedure of the P∃NNQ-algorithm is equivalent to P∀NNQ-pruning.

7. EXPERIMENTAL EVALUATION

Setup Our experimental evaluation focuses on the efficiency and effectiveness of P∀NNQ, P∃NNQ and PCNNQ queries. Due to the high runtime complexity of the exact solutions we will focus on the approximation techniques. We conducted a set of experiments to verify both the effectiveness and efficiency of the proposed solutions, using a desktop computer having an Intel i7-870 CPU at 2.93 GHz and 8GB of RAM. All algorithms were implemented in C++ and integrated into the UST framework. This framework and a video illustrating the datasets can be found on the project page[7].

Artificial Data. Artificial data for our experiments was created in three steps: state space generation, transition matrix construction and object creation. First, the data generator constructs a two-dimensional Euclidean *state space*, consisting of N states. Each of these states is drawn uniformly from the $[0, 1]^2$ square. In order to construct a transition matrix, we derive a graph by introducing edges between any point p and its neighbors having a distance less than $r = \sqrt{\frac{b}{n*\pi}}$ with b denoting the average branching factor of the underlying network. This parameter ensures that the degree of a node does not depend on the number of states in the network. Each edge in the resulting network represents a non-zero entry in the transition matrix. The transition probability of this entry is indirectly proportional to the distance between the two vertices.

To create observations of an object o, we sample a sequence of states and compute the shortest paths between them, modeling the motion of o during its whole lifetime (which we set to 100 steps by default). To add uncertainty to the resulting path, every l^{th} node, $l = i * v$, $v \in [0, 1]$, of this trajectory is used as an observed state. i denotes the time between consecutive observations and v denotes a lag parameter describing the extra time that o requires due to deviation from the shortest path; the smaller v, the more lag is introduced to o's motion. The resulting uncertain trajectories were distributed over the database time horizon (default: 1000 timestamps) and indexed by a UST-tree [25]. As a pruning step for query evaluation, we employed the UST-tree's MBR filtering approach described in Section 6. Our experiments concentrate on evaluating nearest neighbor queries given a certain query state. These states were uniformly drawn from the underlying state space.

Real Data. We also generated a data set from a set of GPS trajectories of taxis in the city of Beijing [32] using map matching. First, trajectories from the dataset below a given gps-frequency were filtered out since these trajectories are not fine-granular enough to provide useful information during the training step. The remaining trajectories were interpolated to obtain measurements with a frequency of 1Hz. These trajectories where then map matched to a reduced Beijing-graph obtained from OpenStreetMap (OSM). Due to the sparsity of data, we assume that a-priori, all objects utilize the same Markov model M. The time domain is discretized to one tic every 10 seconds. From the map matched trajectories, the transition matrix was extracted by aggregating the turning probabilities at crossroads. OSM-nodes with no hits in the underlying training data where filtered out. The state space was then formed by the remaining nodes of the OSM graph, all in all 68902 states. Certain trajectories of cars where taken directly from the map matched trajectories, but in order to ensure comparability to the artificial data have been capped at a length of 100 tics and distributed in the database horizon. The certain trajectories where then made uncertain by taking every l-th gps measurement as an observation; the discarded gps measurements serve as ground truth for effectiveness experiments. For the real data experiment varying the number of objects, we set $l = 8$.

7.1 Evaluation: P∀NNQ and P∃NNQ

For performance analysis, the sampling approach (Section 5) is divided into two phases. In the first phase the trajectory sampler

[7]http://www.dbs.ifi.lmu.de/cms/Publications/UncertainSpatioTemporal

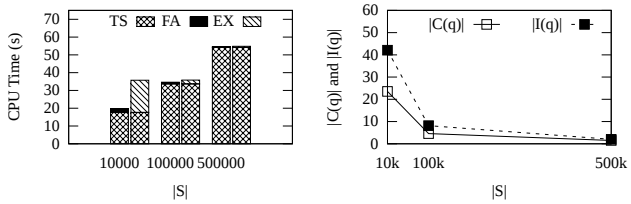


Figure 6: Varying the Number of States N

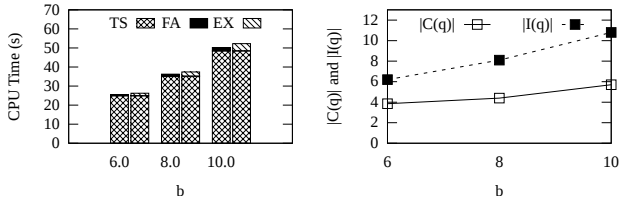


Figure 7: Varying the Branching Factor b

(*TS*) is initialized (the adapted transition matrices are computed according to Algorithm 2). This phase can be performed once and used for all queries. In the second phase, the actual sampling of 10k trajectories (per object) for the approximate P∀NNQ (FA) and P∃NNQ (EX) queries is performed.

In our default setting during efficiency analysis on the artificial dataset we set the number of objects $|\mathcal{D}| = 10k$, the number of states $N = |\mathcal{S}| = 100$k, average branching factor of the synthetic graph $b = 8$, probability threshold $\tau = 0$ and the length of the query interval $|T| = 10$. These parameters lead to a total of $110k$ observations (11 per object) and $100k$ diamonds for the UST-index.

Varying N. In the first experiment (Figure 6) we investigate the effect of an increasing state space size N, while keeping a constant average branching factor of network nodes. This effect corresponds to expanding the underlying state space, e.g., from a single country to a whole continent. In Figure 6 (left) we can see that increasing N leads to a sublinear increase in the run-time of the sampling approaches. This effect can be mostly explained by two aspects. First, the size of the a-priori model increases linearly with N, since the number of non-zero elements of the sparse matrix M increases linearly with N. This leads to an increase of the time complexity of matrix operations, and therefore makes adapting transition matrices more costly. At the same time, the number of candidates $|C(t)|$ and influence objects $|I(t)|$ (see Section 6) decreases significantly as seen in Figure 6 (right) because the degree of intersection between objects decreases with a higher number of states, making pruning more effective, and therefore reducing the actual cost for sampling.

The runtime difference among sampling the P∀NNQ and P∃NNQ query diminishes with increasing N because the size of the result set of the P∀NNQ increases with N while P∃NNQ produces less results with increasing N. The P∃NNQ runtime is also higher than the P∀NNQ runtime because for the P∃NNQ query not only candidate objects are possible results, but also influence objects.

Varying b. Figure 7 evaluates the branching factor b, i.e., the average degree of each network node. As expected, Figure 7 (left) shows that an increasing branching factor yields a higher run-time of all approaches due to a higher number of non-zero values in vectors and matrices, making computations more costly. Furthermore, in our setting, a larger branching factor also increases the number of influence objects, as shown in Figure 7 (right).

Varying $|\mathcal{D}|$. The number of objects (Figure 8) leads to a decreasing performance as well. The more objects stored in a database with the same underlying motion model, the more candidates and influence objects are found during the filter step. This leads to an increasing number of probability calculations during refinement, and hence a higher query cost.

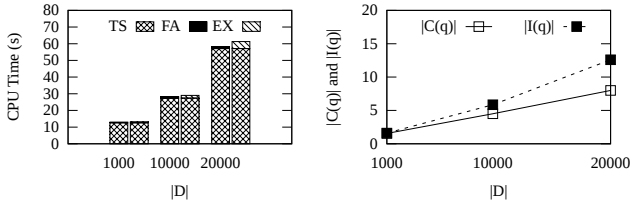


Figure 8: Varying the Number of Objects $|\mathcal{D}|$

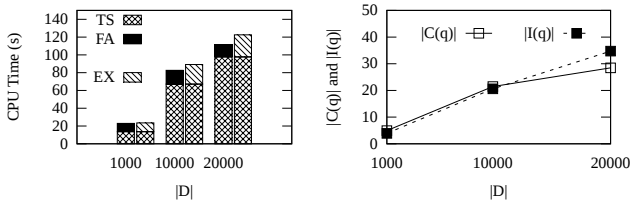


Figure 9: Realdata: Varying the Number of Objects

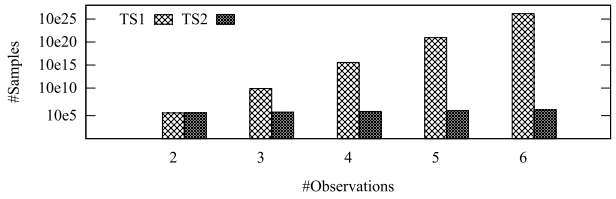
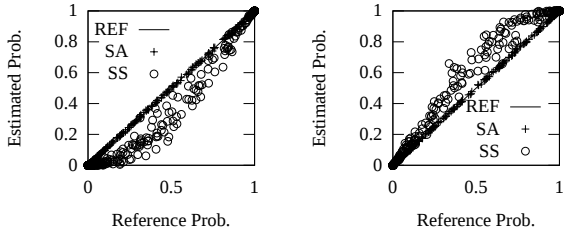


Figure 10: Efficiency of Sampling without Model Adaption.



(a) P∀NN (b) P∃NN

Figure 11: Effectiveness of Sampling, P∀NN and P∃NN

Real Dataset. We conducted additional experiments to evaluate P∀NNQ and P∃NNQ queries on the taxi dataset (Figure 9). The underlying state space consisting of 68902 states is a bit smaller than the default synthetic dataset. Based on this dataset, we ran an experiment varying the number of objects between 1000 and 20000. The smaller size of the state space leads to a higher objects density, leading to a larger number of candidates and influence objects than the corresponding experiment on the artificial dataset. Additionally, the non-uniform distribution of taxis in the city is more dense close to the city center, making queries in this area more costly due to the higher number of candidates and pruners. Further note that in the real dataset, the motion patterns of objects are more diverse than on the synthetic data. There are taxis standing still, and taxis moving quite fast. Standing taxis have a larger area of uncertainty between observations, such that these objects reduce the performance of query evaluation.

Sampling Efficiency. In the next experiment we evaluate the overhead of the traditional sampling approach (using the a-priori Markov model only) compared to the approach presented in Section 5 which uses the a-posteriori model again based on the artificial dataset. The first, traditional approach (TS1) discards any trajectory not visiting all observations. As discussed in Section 5.1, the expected number of attempts required to draw one sample that hits all observations increases exponentially in the number of observations. This increase is shown in Figure 10, where the expected number of samples is depicted with respect to the number

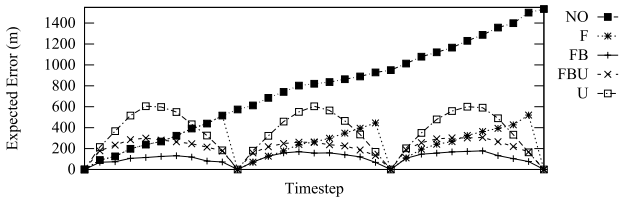


Figure 12: Realdata: Effectiveness of the Model Adaption

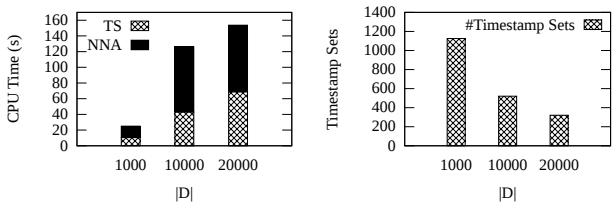


Figure 13: PCNN: Varying the Number of Objects

of observations. This approach can be improved, by segment-wise sampling between observations (TS2). Once the first observation is hit, the corresponding trajectory is memorized, and further samples from the current observation are drawn until the next observation is hit. The number of trajectories required to be drawn in order to obtain one possible trajectory, i.e., the trajectory hits all observations, is linear to the number of observations when using this approach. We note in Figure 10, that in either approach at least 100k samples are required even in the case of having only two observations. In contrast using the approach presented in Section 5, the number of trajectories that need to be sampled, in order to obtain a trajectory that hits all observations, is always *one*.

Sampling Precision and Effectiveness. Next, we evaluate the precision of our approximate P∀NNQ and P∃NNQ query and an aspect of a competitor approach [19]. The latter approach has been tailored for *reverse* NN queries, but can easily be adapted to NN query processing. Essentially, this approach performs a snapshot query $P\forall NNQ(q, \mathcal{D}, \{t\}, \tau)$ for each $t \in T$. $P\forall NN(o, q, \mathcal{D}, T)$ is estimated by $\prod_{t \in T} P\forall NN(o, q, \mathcal{D}, \{t\})$. $P\exists NN(o, q, \mathcal{D}, T)$ can be approximated by $1 - \prod_{t \in T}(1 - P\exists NN(o, q, \mathcal{D}, \{t\}))$. The scatterplot in Figure 11 (right) illustrates a set of P∀NN probabilties on synthetic data ($v = 0.2$, $|T| = 5$). For each experiment, we estimate probabilities by our sampling approach (SA) (Section 5) with (10^4) samples and by the adapted approach of [19] (SS). We approximated the exact approach (REF) by drawing a very high (10^6) number of samples.

We model each case as a (x,y) point, where x models the reference (REF) and y the estimated probability (SA or SS). For (REF) the results always lie on the diagonal identity function depicted by a straight line. Probabilities of SA are very close to the diagonal, showing that our sampling solution tightly approximates the results of the exact P∀NNQ query. Concerning the snapshot approach, a strong bias towards underestimating probabilities can be observed for the P∀NNQ query. The snapshot-based P∃NNQ-query overestimates the results. This bias is a result of treating points of time mutually independent. In reality, the position at time t must be in vicinity of the position at time $t-1$, due to maximum speed constraints. This positive correlation in space directly leads to a nearest neighbor correlation: If o is close to q at time $t-1$, then o is likely close to q at time t. And clearly, if o is more likely to be close to q at time t, then o is more likely to be the NN of q at time t. This correlation is ignored by snapshot approaches. It can be seen that the systematic error of [19] is quite significant.

The number of samples required to obtain an accurate approximation of the probability of a binomial distributed random event such as the event that o is the NN of q for each time $t \in T$ has been studied extensively in statistics [29]. Thus the required number of samples is not explicitly evaluated here.

Effectiveness of the Forward-Backward Model. We tested the effectiveness of the forward-backward model adaption in comparison to other approaches on the real dataset with a time interval between observations of 100 seconds. Figure 12 shows the mean error of these approaches, computed during each point of time, evaluated over a time interval of 30 tics (5 minutes). The mean error has been computed in leave-one-out manner, i.e. trajectories for computing the error have not been used to train the model in order to avoid overfitting. The figure visualizes the error of the a-priori model (NO) considering only the first observation, the model adapted by the forward phase only (F) and the forward-backward-adapted a-priori-model (FB) from this paper. We further implemented two additional approaches. The uniform approach (U), a competitor corresponding to [13, 16], discards all probability information of FB and, due to a lack of better knowledge, assumes all reachable states at a given time to have a uniform probability. The difference to the cylinders and beads approximation models presented in [13, 16] is that these models use conservative approximations that may include some (time, state) pairs actually having a zero probability for an object to be located at. Thus, our U approach is at least as good as the cylinders and beads approximation models in terms of effectiveness, regardless of the approximation type used. The approach FBU is equivalent to FB, however turning probabilities in the transition matrix are equally distributed instead of learning the exact transition probabilities from the underlying map data. First note that the approach not incorporating any observations (NO), yields significant errors compared to the remaining approaches. Clearly, observations can reduce errors and uncertainty during query evaluation. The forward-only approach (F) reduces this error, however the error is still high especially directly before an observation. This problem is solved by the forward-backward approach from this paper (FB). Note that even if the Markov chain is assumed to be uniformly distributed (FBU), the results are still good, but worse than with the actual learned probabilities (FB). This is good news, as it shows that even a non-optimally learned Markov chain can lead to useful results, however with a slightly higher error. This good performance comes from the fact that with a uniform transition distribution the diamond-shaped space of possible time-state pairs still has high probabilities in the center of the diamond, since trajectories near the center of the bead will have a higher likelihood than trajectories close to the beads boundary. This stands in contrast to the uniform approach (U) that models all states at the diamonds border to have the same probabilities as the states in the diamonds center; explaining why U performs worse than FBU. To conclude, combining observations with a sufficiently accurate transition matrix can produce the most accurate results.

7.2 Continuous Queries

In our experimental evaluation on continuous queries we compare the runtime and the size of the (unprocessed) result set for various database sizes and values of the threshold τ (default $\tau = 0.5$) using artificial data. After query evaluation, this result set can be further condensed, e.g. by removing all smaller sets of timestamps that are already implicitly contained in a larger set of timestamps.

Increasing the number of objects stored in the database leads to an increase in the time needed to compute the a-posteriori Markov model (TS) for each object (cf. Figure 13 (left)). This result is equivalent to the result for P∀NNQ queries, since a-posteriori models have to be computed for either query semantics. However, the time required to obtain a sufficient number of samples (*SA*) is

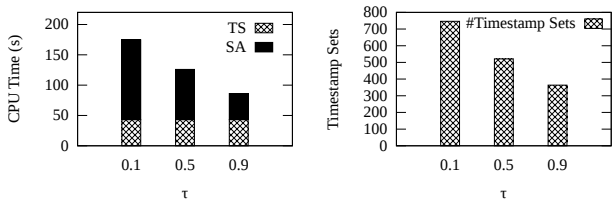


Figure 14: PCNN: Varying τ

much higher, since probabilities have to be estimated for a number of sets of time intervals, rather than for the single interval T. This increase in run-time is alleviated by the effect that the number of candidate time intervals obtained in the candidate time interval generation step of our Apriori-like algorithm decreases (Figure 13 (right)). This effect follows from the fact that more objects lead to more pruners, leading to smaller probabilities of time intervals, leading to fewer candidate time intervals. The results of varying τ can be found in Figure 14. Clearly an increasing probability threshold decreases the average size of the result (Figure 14 (right)). Consequently, the computational complexity of the query decreases as fewer candidates are generated. Figure 14(left) shows that the runtime of the sampling approach becomes very large for low values of τ, since samples have to be generated for each relevant candidate set. Similar to the Apriori-algorithm, the number of such candidates grows exponentially with T, if τ is small.

8. K-NEAREST-NEIGHBOR QUERIES

Computing P$\forall k$NN and P$\exists k$NN is NP-hard in k. Finally, the $C\forall k$NNQ query which is based on the $\forall k$NNQ query, is also NP-hard. The proof of this statement can be found in our technical report [26]. To answer P$\exists k$NNQ queries, P$\forall k$NNQ queries and PCkNNQ queries approximately in the case of $k > 1$, we can again utilize the model adaptation and sampling technique presented in Section 5. Therefore, possible worlds are sampled using the a-posteriori models of all objects, given their observations. On each such (certain) world an existing solution for kNN search on certain trajectories (e.g. [5, 6, 7, 8]) is applied. The results of these deterministic queries can again be used to estimate the distribution of the probabilistic result.

9. CONCLUSIONS

In this paper, we addressed the problem of answering NN queries in uncertain spatio-temporal databases. We proposed three different semantics of NN queries: P$\forall$NNQ queries, P$\exists$NNQ queries and PCNN queries. We have first analyzed the complexity of these queries, showing that computing all of them has high runtime complexity. These results provide insights about the complexity of NN search over uncertain data in general since the Markov chain model is one of the simplest models that consider temporal dependencies. More complex models are expected to be at least as hard. To mitigate the problems of computational complexity, we used a sampling-based approach based on Bayesian inference. For the PCNNQ query we proposed to reduce the cardinality of the result set by means of an Apriori pattern mining approach. To cope with large trajectory databases, we introduced a pruning strategy to speed-up PNN queries exploiting the UST tree, an index for uncertain trajectory data. The experimental evaluation shows that our adapted a-posteriori model allows to effectively and efficiently answer probabilistic NN queries despite the strong a-priori Markov assumption.

10. REFERENCES

[1] C. Ré, J. Letchner, M. Balazinksa, and D. Suciu, "Event queries on correlated probabilistic streams," in *Proc. SIGMOD*, 2008, pp. 715–728.

[2] E. Cho, S. A. Myers, and J. Leskovec., "Friendship and mobility: User movement in location-based social networks," in *Proc. KDD*, 2001.

[3] 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.

[4] Y. Zheng, Q. Li, Y. Chen, X. Xie, and W. Ma., "Understanding mobility based on gps data." in *Proc. Ubicomp*, 2008, pp. 312–312.

[5] E. Frentzos, K. Gratsias, N. Pelekis, and Y. Theodoridis, "Algorithms for nearest neighbor search on moving object trajectories," *Geoinformatica*, vol. 11, no. 2, pp. 159–193, 2007.

[6] R. H. Güting, T. Behr, and J. Xu, "Efficient *k*-nearest neighbor search on moving object trajectories," *VLDB J.*, vol. 19, no. 5, pp. 687–714, 2010.

[7] G. S. Iwerks, H. Samet, and K. Smith, "Continuous k-nearest neighbor queries for continuously moving points with updates," in *VLDB*, 2003, pp. 512–523.

[8] Y. Tao, D. Papadias, and Q. Shen, "Continuous nearest neighbor search," in *Proc. VLDB*, 2002, pp. 287–298.

[9] Y. Tao, C. Faloutsos, D. Papadias, and B. Liu, "Prediction and indexing of moving objects with unknown motion patterns," in *Proc. SIGMOD*, 2004, pp. 611–622.

[10] X. Yu, K. Q. Pu, and N. Koudas, "Monitoring k-nearest neighbor queries over moving objects," in *Proc. ICDE*, 2005, pp. 631–642.

[11] 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*, 2005, pp. 643–654.

[12] H. Mokhtar and J. Su, "Universal trajectory queries for moving object databases," in *Proc. MDM*, 2004, pp. 133–144.

[13] G. Trajcevski, O. Wolfson, K. Hinrichs, and S. Chamberlain, "Managing uncertainty in moving objects databases," *ACM Trans. Database Syst.*, vol. 29, no. 3, pp. 463–507, 2004.

[14] G. Trajcevski, R. Tamassia, H. Ding, P. Scheuermann, and I. F. Cruz, "Continuous probabilistic nearest-neighbor queries for uncertain trajectories," in *Proc. EDBT*, 2009, pp. 874–885.

[15] T. Emrich, H.-P. Kriegel, N. Mamoulis, M. Renz, and A. Züfle, "Querying uncertain spatio-temporal data," in *Proc. ICDE*, 2012, pp. 354–365.

[16] G. Trajcevski, A. N. Choudhary, O. Wolfson, L. Ye, and G. Li, "Uncertain range queries for necklaces," in *Proc. MDM*, 2010, pp. 199–208.

[17] R. Cheng, D. Kalashnikov, and S. Prabhakar, "Querying imprecise data in moving object environments," in *IEEE TKDE*, vol. 16, no. 9, 2004, pp. 1112–1127.

[18] 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.*, vol. 33, no. 3, pp. 370–386, 2010.

[19] C. Xu, Y. Gu, L. Chen, J. Qiao, and G. Yu, "Interval reverse nearest neighbor queries on uncertain data with markov correlations," in *Proc. ICDE*, 2013.

[20] G. Kollios, D. Gunopulos, and V. Tsotras, "Nearest neighbor queries in a mobile environment," in *Spatio-Temporal Database Management*. Springer, 1999, pp. 119–134.

[21] A. Prasad Sistla, O. Wolfson, S. Chamberlain, and S. Dao, "Modeling and querying moving objects," in *Proc. ICDE*. IEEE, 1997, pp. 422–432.

[22] Y.-K. Huang, S.-J. Liao, and C. Lee, "Efficient continuous k-nearest neighbor query processing over moving objects with uncertain speed and direction," in *Proc. SSDBM*, 2008, pp. 549–557.

[23] G. Li, Y. Li, L. Shu, and P. Fan, "Cknn query processing over moving objects with uncertain speeds in road networks," in *APWeb*, 2011, pp. 65–76.

[24] G. Trajcevski, R. Tamassia, I. F. Cruz, P. Scheuermann, D. Hartglass, and C. Zamierowski, "Ranking continuous nearest neighbors for uncertain trajectories," *VLDB J.*, vol. 20, no. 5, pp. 767–791, 2011.

[25] T. Emrich, H.-P. Kriegel, N. Mamoulis, M. Renz, and A. Züfle, "Indexing uncertain spatio-temporal data," in *Proc. CIKM*, 2012, pp. 395–404.

[26] 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 (technical report)," 2013, http://www.dbs.ifi.lmu.de/Publikationen/Papers/TR_PNN.pdf.

[27] R. Agrawal and R. Srikant, "Fast algorithms for mining association rules," in *Proc. VLDB*, 1994, pp. 487–499.

[28] R. Jampani, F. Xu, M. Wu, L. L. Perez, C. M. Jermaine, and P. J. Haas, "Mcdb: a monte carlo approach to managing uncertain data," in *Proc. SIGMOD*, 2008, pp. 687–700.

[29] W. Hoeffding, "Probability inequalities for sums of bounded random variables," *Journal of the American Statistical Association*, pp. 13–30, 1963.

[30] L. R. Welch, "Hidden markov models and the baum-welch algorithm," *IEEE Information Theory Society Newsletter*, vol. 53, no. 4, pp. 1,10–13, 2003.

[31] N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger, "The R*-Tree: An efficient and robust access method for points and rectangles," in *Proc. SIGMOD*, 1990.

[32] J. Yuan, Y. Zheng, X. Xie, and G. Sun, "Driving with knowledge from the physical world," in *Proc. KDD*, 2011, pp. 316–324.