KSQ: Top-k Similarity Query on Uncertain Trajectories

Chunyang Ma, Hua Lu, *Member*, *IEEE*, Lidan Shou, and Gang Chen

Abstract—Similarity search on spatiotemporal trajectories has a wide range of applications. Most of existing research focuses on certain trajectories. However, trajectories often are uncertain due to various factors, for example, hardware limitations and privacy concerns. In this paper, we introduce *p-distance*, a novel and adaptive measure that is able to quantify the dissimilarity between two uncertain trajectories. Based on this measure of dissimilarity, we define top-k similarity query (KSQ) on uncertain trajectories. A KSQ returns the k trajectories that are most similar to a given trajectory in terms of p-distance. To process such queries efficiently, we design *UTgrid* for indexing uncertain trajectories and develop query processing algorithms that make use of UTgrid for effective pruning. We conduct an extensive experimental study on both synthetic and real data sets. The results indicate that UTgrid is an effective indexing method for similarity search on uncertain trajectories. Our query processing using UTgrid dramatically improves the query performance and scales well in terms of query time and I/O.

Index Terms—Uncertain trajectories, spatiotemporal similarity, top-k

1 Introduction

Large amounts of trajectory data have been and are continuously being collected. For example, the wide use of the GPS service has yielded huge volumes of tracking data for vehicles and moving individuals. An interesting problem on such trajectory data is the *similarity search*. For example, many websites allow people to upload their GPS locations and share their journeys through web communities [4], [15]. Based on such shared journeys, social network websites can recommend to their users new friends who have the most similar journeys and, thus, probably similar interests in life.

Ideally, a trajectory is represented as a sequence of locations each being associated with a corresponding time stamp. This, however, is a simplification that does not take into account the inherent uncertainties in such trajectories. For example, a position reported in a GPS signal usually implies a location point with an error range rather than an exact geolocation. Moreover, as location privacy has become increasingly a concern, many locations are blurred (e.g., to cloaking regions like circles) when they are made public. We in this paper represent a trajectory of a moving object as a sequence of reported locations at corresponding time stamps and a probability distribution function. The probability distribution function represents the probability distribution of possible locations of the trajectory at a given time instant. In particular, the probability distribution function can be discrete or continuous.

A key issue in the context of similarity search on uncertain trajectories is an appropriate distance function to measure the (dis)similarity between two uncertain trajectories. Particularly, an effective similarity metric should be able to conduct measurements in terms of different probability distributions, taking into account spatial distances, temporal intervals, as well as relevant probabilities.

The majority of existing notions such as dynamic time warping (DTW), longest common subsequences (LCSS), and so on, are only focused on certain time-series analysis. In addition, these notions only focus on values of a sequence of discrete time stamps and do not work for in-between time intervals that are often considered in the context of trajectories.

An alternative and intuitive way is to measure the expected euclidean distance (ED) between two uncertain trajectories. Given a query trajectory Q and a query time interval I, a similarity search retrieves the trajectory with the smallest expected ED on I with respect to Q .

However, the expected ED is not a reliable indicator of similarity in the uncertain context [34]. It is very sensitive to outliers. We show this by an example in Fig. 1. There are three trajectories X, Y, and Z in the database, each involving two time instants. At each time instant, each trajectory is captured as two possible locations. For example, X has possible values $X_1[1]$ and $X_1[2]$ at time t_1, with probability 0.99 and 0.01, respectively. For the sake of easy presentation, we use a certain query trajectory Q that has only one possible location at each time instant (Q_1 and Q_2 for t_1 and t_2, respectively). Intuitively, X is the most similar trajectory to Q because at both time instants X is almost sure (with 99 percent probability) to be the nearest trajectory to Q. However, the expected euclidean distance between X and Q can be very large and particularly larger than that between Y (or Z) and Q, because the 1 percent

- *C. Ma, L. Shou, and G. Chen are with the Department of Computer Science, College of Compute Science, Zhejiang University, No. 38 Zheda Road, Hangzhou 310027, China. E-mail: {mcy, should, cg}@zju.edu.cn.*
- *H. Lu is with the Department of Computer Science, Aalborg University, Selma Lagerlofs Vej 300, DK-9220 Aalborg East, Denmark. E-mail: luhua@cs.aau.dk.*

Manuscript received 12 Sept. 2011; revised 25 Feb. 2012; accepted 24 July 2012; published online 2 Aug. 2012.
Recommended for acceptance by X. Zhou.
For information on obtaining reprints of this article, please send e-mail to: tkde@computer.org, and reference IEEECS Log Number TKDE-2011-09-0549.
Digital Object Identifier no. 10.1109/TKDE.2012.152.

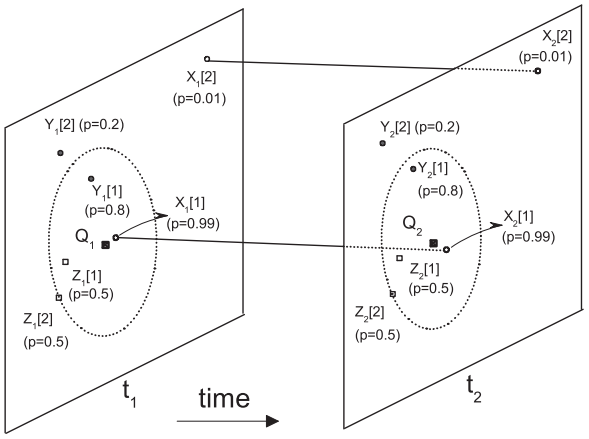


Fig. 1. An example of similarity search.

probability of X can be sufficiently far away from Q. In such a case, Y or Z will be returned as the most similar trajectory to Q if the expected ED is used as the similarity measure, which makes little sense compared to the straightforward answer X.

A more reasonable approach is to combine both spatial distances and probabilities to measure the dissimilarity between two uncertain trajectories. To the best of our knowledge, the only such technique is the time-range-specified probabilistic similarity query proposed for data streams [33] (also known as cloaked range query [18] for cloaked time series). Requiring two user-specified parameters, a distance threshold r and a probability threshold τ, such a query returns the answers with a probability higher than τ to be within a total distance r to the given trajectory. The query results, however, are very sensitive to the two parameters. Small changes to these threshold values can lead to highly different query results in terms of both content and size.

Refer to Fig. 1 again. Trajectories in the database have multiple possible distance values, each with a corresponding probability. Specifically, X has four possible distance values $|Q_1, X_1[1]| + |Q_2, X_2[1]|, |Q_1, X_1[1]| + |Q_2, X_2[2]|, |Q_1, X_1[2]| + |Q_2, X_2[1]|$, and $|Q_1, X_1[2]| + |Q_2, X_2[2]|$, with probabilities 0.9801, 0.0099, 0.0099, and 0.0001, respectively. Similarly, Y and Z both have four possible distances. If a user specifies an r less than $|Q_1, X_1[1]| + |Q_2, X_2[1]|$, no trajectories will be returned as answer. If a user specifies $r = |Q_1, Y_1[2]| + |Q_2, Y_2[2]|$ and a τ larger than 0.9801, only Y and Z will be returned. Specifying these query parameters to get the desirable results requires deep understanding of the uncertain trajectory data set. This, unfortunately, is often beyond a user's capacity in many application scenarios.

This paper proposes a novel and effective metric, *p-distance*, to measure the dissimilarity between two uncertain trajectories. Given a query trajectory Q, the p-distance between an uncertain trajectory in the database and Q is determined by computing the quantity of all other trajectories in the database that may be nearer to Q during the query time interval. The p-distance gives a ranking on uncertain trajectories based on the total order of distances between trajectories and the query, which renders it adaptive to different data sets.

Based on the p-distance metric, we define a top-k similarity query (KSQ) to retrieve exactly k trajectories with the smallest p-distances with respect to the query trajectory. KSQs give users considerable convenience in that users do not need to specify any ranking functions in querying. To facilitate processing KSQs, we propose a novel index structure called *UTgrid* to manage all uncertain trajectories. The results of an extensive experimental study show that UTgrid-based KSQ algorithms are efficient and scalable.

Our contributions in this paper are the following:

- We introduce p-distance, a novel and adaptive metric to measure the dissimilarity between two uncertain trajectories.
- We propose a p-distance-based top-k similarity query (KSQ) that retrieves the k most similar trajectories to a query trajectory in a very easy-to-use manner.
- We design a novel structure UTgrid to index large amount of uncertain trajectories.
- We develop detailed algorithms for processing KSQs by exploiting the UTgrid.
- We do an extensive experimental study on both synthetic and real data sets to evaluate our proposals.

The rest of the paper is organized as follows: Section 2 reviews the related work. Section 3 presents the problem statement. Section 4 describes our indexing technique UTgrid. Section 5 details the algorithms for processing KSQs. Section 6 reports on experiments of our proposals. Section 7 concludes the paper.

2 Related Work

In this section, we briefly review works of querying certain time series and trajectories, querying uncertain time series and trajectories, and top-k queries on probabilistic databases.

2.1 Certain Time Series and Trajectories

Similarity search has attracted considerable research efforts in the context of time series. Various distance notions have been proposed, such as DTW [26], LCSS [32], and edit distance on real sequence [8]. However, they all address only certain time series with discrete time stamps and do not work for continuous time intervals. ED-based spatio-temporal trajectory similarity [13], [1] falls short in an uncertain setting. It becomes unreliable in identifying similarity when uncertain data set contains outliers [34]. In contrast, the p-distance proposed in this paper is a holistic approach that takes into account all trajectories in a database. This makes p-distance more adaptive to the characteristics of the trajectory set of interest and, thus, avoiding query result distortion caused by data outliers.

Many index structures have been proposed for indexing certain trajectory data [21], [6], [20], [24], [25], [7], [12], [5]. Some works treat every 2D trajectory as a 3D polyline and construct R-tree to index 3D line segments accordingly [24], [25]. In some proposals, trajectories are split spatially into subtrajectories based on space partitioning methods such as grid and Quad-tree, and then grouped into a collection of "spatial partitions" [7], [12], [5]. Within each spatial partition, the temporal information of relevant subtrajectories is indexed by a sparse index like 1D R-tree. However, all these index structures focus on certain trajectories.

2.2 Uncertain Time Series and Trajectories

Pfoser and Jensen [23] employ a 2D elliptical area to capture the possible locations of a moving object between two consecutive sampling positions. Zhang et al. [35] propose an inference method for uncertain moving object to predict their future locations. In this paper, we focus on similarity queries on historical trajectories in which object locations can follow various distributions.

Spatiotemporal range queries on uncertain trajectories are addressed in both unconstrained space [31], [10] and road networks [37]. In contrast, we consider similarity queries on uncertain trajectories in this paper.

Trajcevski et al. [30] study continuous probabilistic nearest neighbor queries on uncertain trajectories with respect to an uncertain query trajectory Q. This query returns all trajectories that could be one of the k uncertain trajectories with the highest probability of being closest to Q at any time instant within the query time interval and identifies time points where the order of the k nearest trajectories changes, whereas our KSQ returns trajectories with the smallest averaging "distance" to Q over the whole query time interval. In addition, [30] requires the sizes of uncertain zones, and pdfs of all trajectories in the data set are the same, which is too restricted in real applications.

Pelekis et al. [22] study the issue of clustering uncertain trajectories. They transform trajectories into multidimensional points based on a grid of equal rectangular cells. We also employ grid partitioning techniques; however, we use the grid as an index for fast spatial access and elimination.

Probabilistic similarity queries have been studied on both cloaked time series [18] and uncertain data streams [33]. Such a query requires two parameters to be specified by a query user. As pointed out in Section 1, such queries demand too much user knowledge to guarantee desirable results and cannot control the output size, while our KSQ is very easy to use as it requires no parameters from a user and exactly k answers are returned. Aßfalg et al. [2] assume that the uncertainty of a time series is represented by a set of sample observations at each time slot. The limitation of this work is that not all application domains provide multiple sample points for each time slot.

2.3 Top-k Queries on Probabilistic Databases

A variety of top-k definitions have been proposed for probabilistic databases, such as the Global-Topk query [36], the PT-k query [16], the U-Topk query, and the U-kRanks query [27]. However, these top-k definitions do not satisfy the desired properties that evaluate a top-k definition [11]. The expected ranks satisfy the desired properties [11], and therefore, in this paper, we adopt the advantage of the expected ranks to rank objects (trajectories) based on an order naturally among them. Although the expected ranks tends to give very high priority to a tuple with a high occurrence probability even if it has a low score [17], our p-distance avoids the problem because all objects (trajectories) have equal occurrence probabilities in our setting. In addition, Li et al. [17] have proposed parameterized functions to generalize and approximate existing ranking functions for probabilistic databases. However, it is not clear how such generalization and approximation can be applied to uncertain trajectories.

TABLE 1
Symbols

Symbol	Meaning
$\mathscr{D}_p$	The p-distance.
I	The number of intervals in each dimension i.
$P_{X,c_j}(t)$	The probability function of X in a grid cell c_j.
$APF_{c_j}(t)$	The aggregated probability function of a cell c_j.
U	The total number of unit time intervals.

3 Problem Statement

In this section, we present our problem statement. We formalize the model of uncertain trajectories in Section 3.1 and give the definitions of our similarity notions and queries in Section 3.2. Frequently used symbols throughout this paper are summarized in Table 1.

3.1 Data Model

Without loss of generality, in this paper, we adopt a commonly used data model for uncertain trajectories, which was first introduced by Trajcevski et al. [31]. In particular, to make the representation concise, we set our discussion in a 2D space. Nevertheless, the definitions and techniques in this paper apply to higher dimensional spaces. We first give the definition of a certain trajectory.

Definition 1 (Certain trajectory). *A certain trajectory O is represented as a sequence of points $\{(o_1, t_1), (o_2, t_2), \ldots, (o_n, t_n)\}(t_1 < t_2 < \cdots < t_n)$, where o_i is the location of O at sample time t_i and the location of O in a time between t_i and $t_{i+1}(1 \leq i < n)$ is obtained by a linear interpolation between o_i and o_{i+1}.*

A certain trajectory defines the location of an object as an implicit function of time. Here, O is at point o_i at time t_i and during each segment $[t_i, t_{i+1}]$, O moves along a straight line from o_i and o_{i+1} at a constant speed [31]. Based on the definition of certain trajectories, we have the definition of an uncertain trajectory as follows:

Definition 2 (Uncertain trajectory). *An uncertain trajectory X is a combination of a certain trajectory O and a probability distribution function.*[1]

Based on Definition 2, at each time instant, the expected location of X is the location of O, and the probability that X is located at a particular point x in the data space follows X's corresponding probability distribution function.

Note that the probability distribution function of an uncertain trajectory X can be discrete or continuous. In particular, when X is associated with a discrete probability distribution, we use a probability mass function (pmf) to represent its probability distribution. Let $X.m$ denote the number of possible locations of X. At each time instant $\hat{t}$, we have $\sum_{j=1}^{X.m} X.\mathrm{pmf}(X_{\hat{t}}[j]) = 1$, where $X_{\hat{t}}[j]$ is the jth possible location of X at $\hat{t}$.

Given a trajectory X with a discrete probability distribution, we assume a one-to-one mapping between the possible points in two consecutive sample times. In particular, each possible point $X_{t_i}[j]$ at time t_i corresponds to a possible

1. More precisely speaking, we mean a series of isomorphic probability distribution functions each of which is associated to a sample time.

point $X_{t_{i+1}}[j]$ at the next sample time t_{i+1} if t_i is not the last sample time, as well as a possible point $X_{t_{i-1}}[j]$ at the previous sample time t_{i-1} if t_i is not the first. In addition, to obtain possible points of X at an arbitrary time between two consecutive sample times, we only allow linear interpolations to be done by using two corresponding possible points at these two points. For example, linear interpolation can be done for any time between t_i and t_{i+1} by using $X_{t_i}[j]$ and $X_{t_{i+1}}[j]$, but it is not allowed by using $X_{t_i}[j]$ and $X_{t_{i+1}}[j']$, where $j \neq j'$.

We further explain this with the example in Fig. 1. Suppose the lifespan of Q is $[t_1, t_2]$, where $t_1 = 1$ and $t_2 = 2$. Then, $X_1[1]$ and $X_1[2]$ at time t_1 are corresponding to $X_2[1]$ and $X_2[2]$ at time t_2, respectively. For each time instant $\hat{t}$ between t_1 and t_2, to obtain the first possible location of X at $\hat{t}$, we find the corresponding possible locations at times t_1 and t_2, which are $X_1[1](p = 0.99)$ and $X_2[1](p = 0.99)$, and obtain $X_{\hat{t}}[1]$ by a linear interpolation between $X_1[1]$ and $X_2[1]$. $X_{\hat{t}}[1]$ is associated with the same probability as $X_1[1]$ and $X_2[1]$, which is 0.99. Likewise, we can get $X_{\hat{t}}[2]$ with a probability 0.01.

When a trajectory X has a continuous probability distribution, we use a probability density function (pdf) to represent its probability distribution, and at each time instant, we have $\int X.pdf(x)dx = 1$. The aforementioned assumption of correspondence, as well as linear interpolation, between consecutive sample times also applies to the continuous case. In particular, the correspondence can be captured by giving the corresponding parameters to $X.pdf$ at the two consecutive sample times, respectively.

In the sequel, we use terms object and trajectory interchangeably.

3.2 The p-Distance and Top-k Similarity Query

Before giving the definition of the similarity query, we introduce a novel and adaptive metric p-distance for measuring the dissimilarity between two uncertain trajectories.

To make the presentation clear and concise, we proceed with considering Q as a certain trajectory. Generalizations of the respective definitions for the uncertain query trajectories are detailed elsewhere [19].

We first introduce the instant p-distance of an arbitrary point x at one time instant $\hat{t}$. Particularly, the instant p-distance of x measures the distance (dissimilarity) between x and Q by the quantity of other trajectories in the database that are nearer to Q than x. The more x is similar (near) to Q, the less trajectories are nearer to Q than x. Based on the intuition, we define the instant p-distance of x at time $\hat{t}$ as the aggregation of probabilities of all other trajectories being nearer to Q than x. Let $P(Q, x \succ Y, \hat{t})$ denote the probability of a trajectory Y being nearer to Q than x at time $\hat{t}$; we have the following definition:

Definition 3 (Instant p-distance of a point). *Given a trajectory database DB and a query trajectory Q, the instant p-distance of a point x at time $\hat{t}$, termed as $\mathcal{D}_p(Q, x, \hat{t})$, is*

$$\mathcal{D}_p(Q, x, \hat{t}) = \sum_{\forall Y \in DB} P(Q, x \succ Y, \hat{t}). \quad (1)$$

In (1), when Y is associated with a continuous probability distribution function pdf,

$$P(Q, x \succ Y, \hat{t}) = \int \theta(Q, x, y) \cdot Y.pdf(y)dy, \quad (2)$$

where

$$\theta(Q, x, y) = \begin{cases} 1 & \text{if } |x, \hat{q}| > |y, \hat{q}|; \\ 0 & \text{otherwise.} \end{cases} \quad (3)$$

When Y is associated with a discrete probability distribution function pmf,

$$P(Q, x \succ Y, \hat{t}) = \sum_{j=1}^{Y.m} \theta(Q, x, Y_{\hat{t}}[j]) \cdot Y.\mathrm{pmf}(Y_{\hat{t}}[j]), \quad (4)$$

where $\theta(Q, x, Y_{\hat{t}}[j])$ is the same as defined in (3).

Note that in (3), $\hat{q}$ is the location of Q at time $\hat{t}$ and $|a, b|$ denotes the ED between two points a and b. Next, we have the definition of the instant p-distance of a trajectory X, which is the expectation of the instant p-distance of each possible location x of X.

Definition 4 (Instant p-distance of a trajectory). *Given a trajectory database DB and a query trajectory Q, the instant p-distance of a trajectory $X \in DB$ at time $\hat{t}$ is termed as $\mathcal{D}_p(Q, X, \hat{t})$, where when X is associated with a continuous probability distribution function pdf,*

$$\mathcal{D}_p(Q, X, \hat{t}) = \int \mathcal{D}_p(Q, x, \hat{t}) \cdot X.pdf(x)dx, \quad (5)$$

when X is associated with a discrete probability distribution function pmf,

$$\mathcal{D}_p(Q, X, \hat{t}) = \sum_{j=1}^{X.m} \mathcal{D}_p(Q, X_{\hat{t}}[j], \hat{t}) \cdot X.\mathrm{pmf}(X_{\hat{t}}[j]). \quad (6)$$

It can be proved that when trajectories are all associated with discrete probability distribution functions, at each time instant, the instant p-distance of a trajectory X is the expected rank of X under the possible world semantics, where the ranking value is the ED. Details can be found elsewhere [19].

However, the instant p-distance is different from the expected ranks in two aspects. First, expected rank is proposed in probabilistic databases, where the probability distribution of data tuples is naturally represented discretely, while the instant p-distance is suitable for objects with both discrete or continuous probability distributions. Second, the ranking key (score) of expected ranks is prefixed before ranking, while the ranking key of instant p-distance of each object needs to be computed according to the specific query object during query processing.

Furthermore, the instant p-distance is different from the absolute metric [9] that directly measures the ED between the query object q and one possible location x of an uncertain object. Namely, the instant p-distance is a relative metric. It measures the distance between the query object q and one possible location x of an uncertain object as the quantity of other objects that are nearer to q than x.

Next, we give the definition of interval p-distance, which is the expectation of the instant p-distance of X during Q's entire lifespan. Without the loss of generality, we assume that the instant p-distance of X at each time instant has the same weight on the interval p-distance of X.

Definition 5 (Interval p-distance). *Given a query trajectory Q, the interval p-distance of X is denoted as $\mathcal{D}_p(Q,X)|_{t_s}^{t_e}$, where $[t_s,t_e]$ is Q's entire lifespan. Specifically,*

$$\mathcal{D}_p(Q,X)|_{t_s}^{t_e} = \frac{1}{t_e - t_s} \cdot \int_{t_s}^{t_e} \mathcal{D}_p(Q,X,t)dt \tag{7}$$

The interval p-distance of X reflects the quantity of all nearer trajectories during Q's lifespan. If a trajectory X is more similar to Q, there are less trajectories nearer to Q than X during $[t_s,t_e]$, which leads to smaller interval p-distance of X.

For the sake of simple presentation, we omit the parameter Q in $\mathcal{D}_p(Q,x,\hat{t}), \mathcal{D}_p(Q,X,\hat{t})$, and $\mathcal{D}_p(Q,X)$, and refer to them as $\mathcal{D}_p(x,\hat{t}), \mathcal{D}_p(X,\hat{t})$, and $\mathcal{D}_p(X)$, respectively, in the rest of the paper.

We describe how to compute the interval p-distance using the example in Fig. 1. Suppose the lifespan of Q is $[t_1,t_2]$, where $t_1 = 1$ and $t_2 = 2$. First, we compute the instant p-distance of X at time t_1. Since $X_1[1]$ is nearer to Q than possible locations of both Y and Z and $X_1[2]$ is always farther to Q than Y and Z, we have $\mathcal{D}_p(X_1[1],t_1) = P(X_1[1] \succ Y,t_1) + P(X_1[1] \succ Z,t_1) = 0$ and $\mathcal{D}_p(X_1[2],t_1) = 2$. As a result, we have $\mathcal{D}_p(X,t_1) = 0 + 2 * 0.01 = 0.02$. The instant p-distance of X at time t_2 can be computed likewise. For each time instant $\hat{t}$ between t_1 and t_2, we obtain the possible locations of X by linear interpolation between possible locations of t_1 and t_2 based on Definition 2, and then compute the instant p-distance $\mathcal{D}_p(X,\hat{t})$ in the same way as computing $\mathcal{D}_p(X,t_1)$.

Thereafter, based on (7), we can compute the interval p-distance of X. For the example in Fig. 1, we have $\mathcal{D}_p(X)|_{t_1}^{t_2} = 0.02$. Similarly, we can compute $\mathcal{D}_p(Y)|_{t_1}^{t_2} = 1.39$ and $\mathcal{D}_p(Z)|_{t_1}^{t_2} = 1.59$.

Note that the interval p-distance of X is much smaller than the other two trajectories. This indicates that X has very high probability to be nearer to Q than others during Q's lifespan. Therefore, X will be returned as the most similar trajectory to Q. Unlike using expected ED directly, the outlier values of X ($X_1[2]$ and $X_2[2]$) have very limited impact on its p-distance. No matter how far $X_1[2]$ and $X_2[2]$ are away from Q, X is still retrieved because most "portion" of X is quite similar to Q.

In the rest of this paper, we use p-distance to imply interval p-distance when the context is clear. The p-distance of trajectories is only based on the order of trajectories themselves, and therefore, it can well capture the characteristics of a data set. Also, no complicated specifications are required from users, which can ease users' work greatly. It is worth mentioning that, like many other total ranking methods, p-distance may have ties in ranking. Increasing the precision of the distance calculations can reduce such ties. Now, we give the definition of top-k spatiotemporal similarity query based on the p-distance.

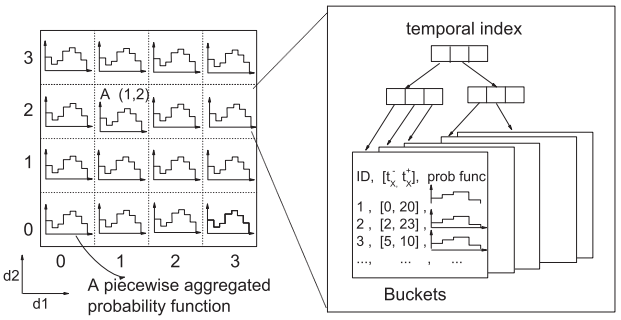


Fig. 2. UTgrid structure.

Definition 6 (Top-k Similarity Query, KSQ). *Given a trajectory database DB ($|DB| \geq k$), a query trajectory Q with lifespan $[t_s,t_e]$, and an integer k, a KSQ retrieves a set $S \subseteq DB$ that consists of k uncertain trajectories, such that $\forall X \in S$ and $\forall Y \in DB - S$, $\mathcal{D}_p(X)|_{t_s}^{t_e} \leq \mathcal{D}_p(Y)|_{t_s}^{t_e}$.*

The KSQ has advantages that other similarity queries on uncertain time series/data streams mentioned in Section 2 do not have. KSQs give users considerable convenience that they do not need to specify any ranking functions for the trajectories. In addition, when users specify the size of required answers k, exact k trajectories will be returned. KSQs satisfy the desired properties of top-k definitions summarized elsewhere [11].

4 The UTgrid Index

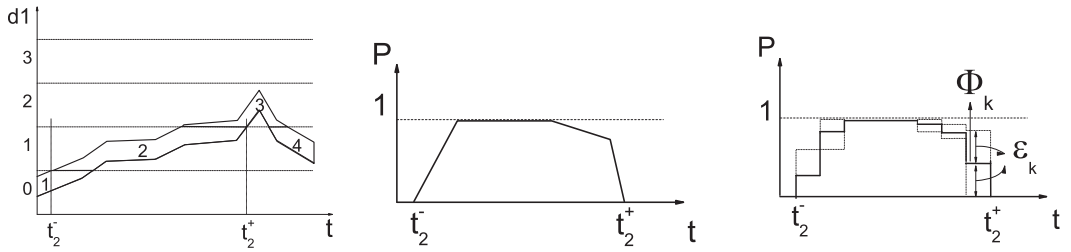
In this section, we present *UTgrid*, a practical index for spatiotemporal *u*ncertain *t*rajectories based on *grid* partitioning. As the query lifespan $[t_s,t_e]$ is usually long, we expect the spatial dimensions to be much more discriminative than the temporal dimension. Therefore, UTgrid first partitions the spatial dimensions (the space domain) into a set of nonoverlapping cells for spatial comparison, and then creates a temporal index within each spatial partition (cell) to help temporal elimination. Particularly, UTgrid effectively divides each trajectory into several pieces and puts each piece into a corresponding cell.

Each cell is organized as follows: A cell maintains the pointer to the bucket(s) that contains all trajectory pieces in it. A cell also has a sparse 1D R-tree as a temporal index for trajectory pieces in it. In addition, a cell has an aggregated probability function that is derived from all trajectory pieces in it. An overall example of UTgrid is shown in Fig. 2. We proceed to give details of UTgrid.

4.1 Components in UTgrid

We first divide each spatial dimension into I intervals. As a result, a 2D data space is partitioned into I^2 nonoverlapping cells. We number the intervals in each dimension i with an identifier cid_i ($0 \leq cid_i < I$). Thus, each cell is identified by a pair of numbers (cid_1, cid_2), corresponding to the two dimensions, respectively. For example, cell A in Fig. 2 is identified as $(1,2)$.

Trajectory pieces. When a trajectory X is to be divided with respect to UTgrid, if X overlaps with several cells, it is split at the boundaries of the cells and divided into several



(a) Pieces of a trajectory (b) The original $P_{X,1}(t)$ (c) The piecewise $P_{X,1}(t)$

Fig. 3. An example of trajectory pieces.

pieces. A 1D example is shown in Fig. 3a. Suppose we divide dimension 1 into four cells (intervals). The boundaries of the four cells partition the trajectory into four pieces, and each piece is completely within one cell. Each trajectory piece in a cell c_j consists of three parts: the ID of X, a time span ($[t_X^-, t_X^+]$), and the probability of X being in c_j, represented as a function of time (denoted as $P_{X,c_j}(t)$).

The time span is the maximum time interval during which X intersects c_j continuously. It is possible that several pieces of a trajectory are within the same cell, like pieces 2 and 4 in Fig. 3a. In such a case, we still treat them as different pieces, because their time spans are not connected.

The probability function $P_{X,c_j}(t)$ indicates the probability of X being in the cell c_j at each time instant. The time domain of $P_{X,c_j}(t)$ is $[t_X^-, t_X^+]$. For example, the probability function $P_{X,1}(t)$ of piece 2 in cell 1 in Fig. 3a is shown in Fig. 3b. Sometimes, the probability functions may take a complex set of segmented functions to represent, which is tedious to present and hard to process. Therefore, we represent them as follows: We divide the whole time domain (the time span of all trajectories in the database) into U unit time intervals. For every unit time interval k intersecting $[t_X^-, t_X^+]$, we calculate a probability Φ_k and an error bound ε_k, where Φ_k is the average probability of X being in c_j during the intersection of the kth unit time interval and $[t_X^-, t_X^+]$, and ε_k is the maximum value of the absolute deviation between Φ_k and the exact probability value. Then, $P_{X,c_j}(t)$ is represented as a sequence of pairs of Φ_k and ε_k, $P_{X,c_j}(t) = \{(\Phi_{k_\vdash}, \varepsilon_{k_\vdash}), \ldots, (\Phi_{k_\dashv}, \varepsilon_{k_\dashv})\}$, where $k_\vdash$ and $k_\dashv$ are the first and last unit time intervals intersecting $[t_X^-, t_X^+]$. Fig. 3c shows the $P_{X,1}(t)$ represented in such a way.

Buckets and temporal indexes. When one bucket does not suffice to store all trajectory pieces inside a cell c_j, we store trajectory pieces in several buckets. Pieces with close time spans are clustered together in the same bucket. Each bucket is also corresponding to a time span that is the minimum time interval covering the time spans of all pieces in it. Then, we employ a 1D R-tree as a temporal index to index the time spans of buckets of cell c_j. Note that when the number of buckets is small enough, only one node suffices to store the time spans of all buckets. Then, the R-tree actually degenerates into a single root node.

Aggregated probability function. The aggregated probability function of a cell c_j is denoted as $APF_{c_j}(t)$. Given two cells c_j and c'_j, if $c_j.cid_i \geq c'_j.cid_i$ in each dimension i, we say that c_j *dominates* c'_j. Then, $APF_{c_j}(t)$ is the aggregation of the probability functions of all pieces in all cells dominated by c_j. Referring to Fig. 2, cell $A(1,2)$ dominates cells $(0,0), (0,1), (0,2), (1,0), (1,1)$, and $(1,2)$ (a cell naturally dominates itself). Then, $APF_{A(1,2)}(t)$ is the aggregation of the probability functions of pieces in all these six cells. Note that $APF_{c_j}(t)$ is also represented piecewise as a sequence of probabilities and error bounds. The time domain of $APF_{c_j}(t)$ equals the whole time domain. Specifically, $APF_{c_j}(t) = \{(\Phi_0, \varepsilon_0), \ldots, (\Phi_{U-1}, \varepsilon_{U-1})\}$.

Note that the temporal indexes and all buckets of UTgrid cells are stored on disk, while the boundaries, piecewise aggregated probability functions, and pointers to the temporal indexes of UTgrid cells are stored in memory. We will further investigate the memory cost of UTgrid in Section 6.

4.2 Constructing UTgrid

When an $I \times I$ grid is initialized, each cell has a pointer to the root of an empty 1D R-tree with only one empty bucket, and an $APF_{c_j}(t)$ that is initially set to 0 for all U unit time intervals.

To insert a trajectory X, we find the cells that intersect with X. For each such cell c_j, we do the following steps. First, we compute the pieces of X that are inside c_j. In particular, we compute the time interval $[t_X^-, t_X^+]$ that X intersects c_j and the piecewise probability function $P_{X,c_j}(t)$. Second, we insert the piece into one of the buckets of c_j in the form of $(X, [t_X^-, t_X^+] P_{X,c_j}(t))$ and update the temporal index of c_j. Finally, we aggregate $P_{X,c_j}(t)$ on the aggregated probability functions of all cells that dominate c_j. In particular, we aggregate $P_{X,c_j}(t)$ onto $APF_{c_{dom}}(t)$ of a cell c_{dom} according to $APF_{c_{dom}}(t).\Phi_k = APF_{c_{dom}}(t).\Phi_k + P_{X,c_j}(t).\Phi_k$ and $APF_{c_{dom}}(t).\varepsilon_k = APF_{c_{dom}}(t).\varepsilon_k + P_{X,c_j}(t).\varepsilon_k$ for all $k \in [k_\vdash, k_\dashv]$.

5 Processing KSQs

In this section, we describe the method to process KSQs with support of UTgrid. Specially, we focus on processing KSQs with certain query trajectories here. Note that the algorithms proposed in this section can be extended to the case of uncertain query trajectories. Although the extension would not be significantly more complex, we leave it for future work due to the space constraint.

We adapt the "filter-refinement" strategy. Facilitated by UTgrid, we first eliminate unqualified objects and get a candidate list. Then, we compute the exact p-distance for objects in the candidate list to get the final answer. Algorithm 1 illustrates the overall procedure of processing a KSQ with query trajectory Q and an integer value k.

Algorithm 1: KSQ(Query trajectory Q, Integer k)

1 Create an empty heap H;
2 Create an empty candidate trajectory list L_{cand};
3 Create an empty list L_{center_cells};
4 $S_c \leftarrow$ a set of cells containing Q during Q's time span.;
5 **for** *every cell* c_l^c *in* S_c **do**
6 $\quad [t_l^-, t_l^+] \leftarrow$ the time interval that Q is inside c_l^c;
7 $\quad L_{center\_cells} \leftarrow L_{center\_cells} \bigcup \{(c_l^c, [t_l^-, t_l^+])\}$;
8 $\quad$ PushHeap(Q, c_l^c, $[t_l^-, t_l^+]$, H);
9 $k_{up} \leftarrow \infty$;
10 **while** H *is not empty and* $top(H).key \leq k_{up}$ **do**
11 $\quad \langle c_j, key, [t_l^-, t_l^+], PD_{min}(t), PD_{max}(t)\rangle \leftarrow$ Deheap(H);
12 $\quad S_c \leftarrow ChooseNextCell(L_{center\_cells}, c_j, [t_l^-, t_l^+])$;
13 $\quad$ **for** *every cell* c_j' *in* S_c **do**
14 $\quad\quad$ PushHeap(Q, c_j', $[t_l^-, t_l^+]$, H);
15 $\quad$ UpdateLcand($(c_j, key, [t_l^-, t_l^+], PD_{min}(t), PD_{max}(t))$, L_{cand});
16 **for** *every entry* $(X, AggP, LB_{cur}, UB_{cur})$ *in* L_{cand} **do**
17 $\quad LB_{est} \leftarrow LB_{cur} + \frac{top(H).key \cdot (t_e - t_s - AggP)}{t_e - t_s}$;
18 $\quad LB_{cur} \leftarrow LB_{est}$;
19 $\quad$ **if** $LB_{est} > k_{up}$ **then**
20 $\quad\quad$ Mark X as "pruned";
21 Refine the candidate in L_{cand} by computing relevant exact p-distances;
22 Return the k trajectories with the smallest p-distances;

In query processing, we use a min-heap H that maintains entries in the form of $(c, key, [t_l^-, t_l^+], PD_{min}(t), PD_{max}(t))$, where c is a UTgrid cell, $[t_l^-, t_l^+]$ is the valid time interval of the entry and $PD_{min}(t)$ and $PD_{max}(t)$ are functions of the minimum and maximum quantity of probabilities of other objects being nearer than c_j with respect to Q at each time instant during $[t_l^-, t_l^+]$. The valley value of function $PD_{min}(t)$ is used as the key of min-heap H to prioritize all entries in H.

We also maintain a candidate list L_{cand}. It stores all the trajectories that have been seen so far and may be in the query result. Each entry in L_{cand} is in the form of $(X, AggP, LB_{cur}, UB_{cur})$, where X is the ID of a candidate trajectory, $AggP$ is the total probabilities of X seen so far, LB_{cur} and UB_{cur} are the lower bound and upper bound of p-distance of X so far. Note that we may not see all pieces of X during query processing, and therefore, LB_{cur} and UB_{cur} in L_{cand} entries are not the final lower and upper bounds of X. However, we can estimate the final upper and lower bounds of p-distance based on LB_{cur} and UB_{cur}, as to be described in Section 5.3.

Let k_{up} denote the kth current smallest estimated final upper bound of p-distance of trajectories. The overall query procedure works as follows: We visit the UTgrid cells one by one, each with a valid time interval. Using min-heap H, we push cells in certain order to visit the cells with smallest possible p-distance first. The specific order of cell visit is described in Section 5.1. When pushing a cell c_j into H (by calling the function *PushHeap* in lines 8 and 14), we calculate its corresponding $PD_{min}(t), PD_{max}(t)$, and key in the way described in Section 5.2.

We use a while loop to visit cells in min-heap H (lines 10-15). In every while-loop iteration, we pop the top entry of H and process the cell c_j in it. In particular, we retrieve every relevant piece $(X, [t_X^-, t_X^+], P_{X,c_j}(t))$ in c_j's buckets and then, update the relevant entry $(X, AggP, LB_{cur}, UB_{cur})$ in L_{cand} if X is not marked as "pruned." We also estimate the final lower and upper bounds of X (LB_{est} and UB_{est}), and update k_{up}. If $LB_{est} > k_{up}$, X is marked as "pruned." This process is encapsulated in the function *UpdateLcand* (called in line 15), which will be further described in Section 5.3. The while loop stops when H is empty or $top(H).key > k_{up}$. The latter means that no unseen trajectories can have p-distance smaller than the k trajectories that have the smallest estimated upper bound of p-distance seen so far.

Subsequently, we update the LB_{est} value of each entry in L_{cand}, replace LB_{cur} with LB_{est}, and identify unqualified trajectories (lines 16-20). Finally, we refine the candidates in L_{cand} that are not marked as "pruned" in ascending order of LB_{est} and return the k most similar trajectories (lines 21-22). Note that it is unnecessary to compute the exact p-distance for all objects in L_{cand}. We keep updating k_{up} as we compute the exact p-distance, and if an object in L_{cand} is found to have LB_{est} higher than the current k_{up}, it will be eliminated without consideration.

5.1 Cell Visiting Order

To visit cells with smallest possible p-distance first, we push cells into H in ascending order of their minimum distance to Q, since the cells with smaller distance tend to have smaller p-distance. Note that Q may move everywhere during Q's time span, and therefore, the distance between a cell c_j and Q is continuously changing. A cell may be quite near to Q during a period of time but far away from Q in other periods. So, each time we visit a cell c_j, we only consider it in a valid time interval, during which the minimum distance between c_j and Q is relatively steady.

We divide query time $[t_s, t_e]$ based on the locations of query trajectory Q. In particular, Q is partitioned into T pieces by the boundaries of UTgrid cells such that each piece is completely contained in a particular cell. We call the corresponding T cells the *center cells* of Q. Accordingly, $[t_s, t_e]$ is partitioned into T time intervals each in the form of $[t_l^-, t_l^+]$ $(0 \leq l < T)$. In other words, Q is within a corresponding center cell c_l^c during a time interval $[t_l^-, t_l^+] \subseteq [t_s, t_e]$. When we visit a cell, we only consider its portion during one of the T valid time intervals. Note that a cell may be visited more than once, and each visit involves a different valid time interval. In Algorithm 1, we store the T valid time intervals together with the corresponding center cells in a list, L_{center_cells}, in the form of $(c_l^c, [t_l^-, t_l^+])$ (lines 6-9). Fig. 4 shows an example where Q crosses three cells $(3,2), (4,2), (4,1)$, and accordingly, we get $L_{center\_cells} = \{((3,2), [t_0^-, t_0^+]), ((4,2), [t_1^-, t_1^+]), ((4,1), [t_2^-, t_2^+])\}$.

First, we push center cells, each with its corresponding valid time interval, into H, since the minimum distances between center cells and Q are always 0 during their corresponding time intervals. Details of pushing cells into H, together with relevant information for query processing, are described in Section 5.2.

Subsequently, every time when we pop a cell (organized in an entry) out from H, we process the cell and then push relevant cells into H for further processing. Such cells are chosen as follows. Note that during a time interval $[t_l^-, t_l^+]$ involved in L_{center_cells}, if we treat the corresponding cell c_l^c as a center, all relevant cells are in a track with respect to c_l^c.

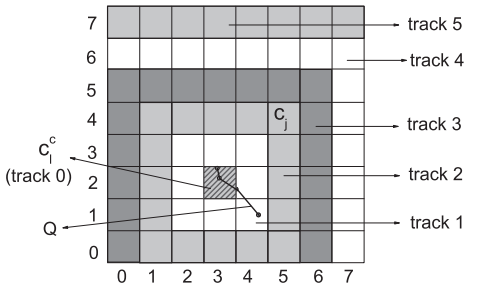


Fig. 4. The order of cell visiting.

Those cells in tracks nearer c_l^c tend to have smaller minimum distances than those in tracks further to c_l^c. We order the tracks in nondescending order of their minimum distance to c_l^c. Accordingly, we push several cells in the next track into H after a pop of H. Particularly, any time we pop a cell c_j in the nth track out of H, we push cells in the $(n+1)$th track that share boundaries with c_j into H. Note that if c_j is on the corner of the nth track, we also push the corner cell in the $(n+1)$th track near c_j into H. Follow the example in Fig. 4. Suppose we have just popped the cell $c_j(5,4)$ with a valid time interval $[t_0^-, t_0^+]$ out of H. To choose relevant cells to be pushed into H, we first find the center cell corresponding to $[t_0^-, t_0^+]$ in L_{center_cells}, which is $c_l^c(3,2)$. Then, centered at $c_l^c(3,2)$, the whole space is partitioned into five ordered tracks, each shadowed in alternate colors. Since c_j is on the corner of track 2, we choose cell $(6,4),(5,5)$, and $(6,5)$ from track 3 to push into H.

Choosing next relevant cells is encapsulated in a function *ChooseNextCell* (called in line 13 in Algorithm 1), which can be found elsewhere [19].

5.2 Pushing Cells into Heap H

When pushing a cell c_j, with a valid time interval $[t_l^-, t_l^+]$, into H, we first compute $PD_{min}(t)$ and $PD_{max}(t)$ of c_j, respectively. Then, we set key of c_j to the valley value of $PD_{min}(t)$ and push c_j into H in the form of $(c_j, key, [t_l^-, t_l^+], PD_{min}(t), PD_{max}(t))$. This procedure of pushing cells is encapsulated in a function $PushHeap$, which can be found elsewhere [19].

Next, we describe how to compute $PD_{min}(t)$ and $PD_{max}(t)$ of a cell c_j. Each $PD_{min}(t)$ is computed in two steps as shown in Algorithm 2. In the first step, we determine the set S_c of the cells that are always nearer to Q than c_j during $[t_l^-, t_l^+]$. Suppose that during $[t_l^-, t_l^+]$ the piece of Q consists of n segments. Since the segments are connected, we have totally $(n+1)$ breaking points. For each breaking point p_f $(0 \le f \le n)$, we compute its minimum distance to c_j, denoted as $dmin_f$. Then, we draw $(n+1)$ circles each with one breaking point as center and the corresponding $dmin_f$ as radius. Therefore, S_c is the set of all the cells that are completely contained in intersection of the circular regions bounded by these $(n+1)$ circles. An example is shown in Fig. 5, where during $[t_0^-, t_0^+]$ the piece of Q consists of two segments and three breaking points. To compute $PD_{min}(t)$ for cell $c_j(5,5)$, we draw three circles centered at the three breaking points and with the corresponding minimum distances as radii. S_c contains all the shadowed cells in this example. Details of how to find cells in the intersection of the $(n+1)$ circular regions to get set S_c are given elsewhere [19].

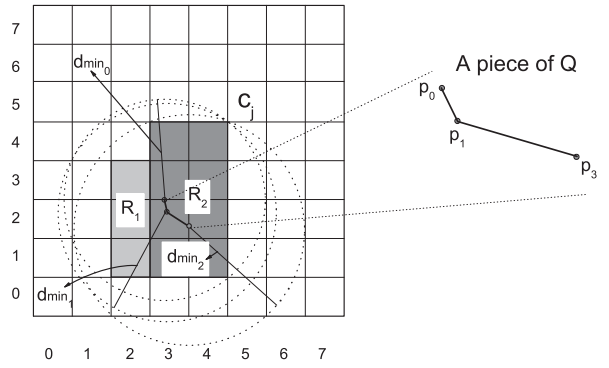


Fig. 5. An example of computing $PD_{min}(t)$.

Algorithm 2: ComputePDmin(Q, c_j, $[t_l^-, t_l^+]$)

```
1  PD_min(t) ≐ 0;
2  Create a set S_c containing all cells ;
3  for every breaking point p_f of Q do
4      dmin_f ← the minimum distance between c_j and p_f;
5      Create a set S_pf containing all cells completely in a
       circle centered at p_f and with a radius dmin_f;
6      S_c ← S_c ⋂ S_pf;
7  Cluster cells in S_c into several rectangles R;
8  for every rectangle R do
9      Compute P_R(t) based on Equation 8;
10     PD_min(t) ← PD_min(t) + P_R(t);
11 return PD_min(t);
```

In the second step, we sum up the probability functions of all pieces contained in the cells in S_c. Note that the cells in S_c are naturally adjacent. So, we can easily partition the region covered by S_c into a few rectangles. As shown in Fig. 5, the cells in S_c of our running example are clustered into two rectangles R_1 and R_2. Instead of retrieving all pieces in all cells in S_c, we compute the aggregation of probability functions of each rectangle R (denoted as $P_R(t)$) based on the aggregated probability functions of only four cells. Remember that the $APF_{c_j}(t)$ of a cell c_j is the aggregation of probability functions of pieces in all cells dominated by c_j. Let the bottom left corner cell of R be c_s, and top right one be c_e, we have

$$\begin{aligned} P_R(t) = {} & APF_{c_e}(t) - APF_{(c_s.cid_1-1, c_e.cid_2)}(t) \\ & - APF_{(c_e.cid_1, c_s.cid_2-1)}(t) \\ & + APF_{(c_s.cid_1-1, c_s.cid_2-1)}(t). \end{aligned} \tag{8}$$

After obtaining the aggregation of probability functions of all relevant rectangles, we sum them up and get $PD_{min}(t)$. Note that $PD_{min}(t)$ is also represented piecewise in the same way as $APF(t)$.

We further explain this with the example in Fig. 5. Rectangle R_1 containing three cells (2,1), (2,2), and (2,3) is bounded by cells $(2,1)$ and $(2,3)$. Then, $P_{R_1}(t) = APF_{(2,3)}(t) - APF_{(1,3)}(t) - APF_{(2,0)}(t) + APF_{(1,0)}(t)$ contains exactly the probability functions of pieces in these three cells. Following the same line of reasoning, R_2 is bounded by cells $(4,4)$ and $(3,1)$, and thus, $P_{R_2}(t) = APF_{(4,4)}(t) - APF_{(2,4)}(t) - APF_{(4,0)}(t) + APF_{(2,0)}(t)$. Consequently, $PD_{min}(t) = P_{R_1}(t) + P_{R_2}(t)$.

$PD_{max}(t)$ is computed in a similar manner. Details can be found elsewhere [19].

5.3 Updating Entries in L_{cand}

Every time we pop an entry $(c_j, key, [t_l^-, t_l^+], PD_{min}(t), PD_{max}(t))$ from H, with support of the corresponding temporal index, we visit c_j by retrieving every piece $(X, [t_X^-, t_X^+], P_{X,c_j}(t))$, with $[t_X^-, t_X^+]$ intersecting $[t_l^-, t_l^+]$, in the buckets of c_j and then updating the entry of X in L_{cand} in the following way.

We retrieve $(X, AggP, LB_{cur}, UB_{cur})$ from L_{cand}. If there is no entry for X in L_{cand}, we create a new entry for X in L_{cand} in the form of $(X, AggP, LB_{cur}, UB_{cur})$, with $AggP, LB_{cur}$ and UB_{cur} all equal to 0. We also mark the new entry as "active."

If X is not marked as "pruned," we update the entry such that

$$\begin{aligned} AggP &= AggP + \int_{max(t_l^-, t_X^-)}^{min(t_l^+, t_X^+)} P_{X,c_j}(t)dt \\ LB_{cur} &= LB_{cur} + \frac{\int_{max(t_l^-, t_X^-)}^{min(t_l^+, t_X^+)} PD_{min}(t) \cdot P_{X,c_j}(t)dt}{t_e - t_s} \\ UB_{cur} &= UB_{cur} + \frac{\int_{max(t_l^-, t_X^-)}^{min(t_l^+, t_X^+)} PD_{max}(t) \cdot P_{X,c_j}(t)dt}{t_e - t_s}. \end{aligned} \tag{9}$$

Recall that $PD_{min}(t), PD_{max}(t)$, and $P_{X,c_j}(t)$ are all represented piecewise as sequences of probabilities and error bounds. So, within (9),

$$\begin{aligned} &\int_{max(t_l^-, t_X^-)}^{min(t_l^+, t_X^+)} PD_{min}(t) \cdot P_{X,c_j}(t)dt \\ &= \sum_{k=k_\vdash}^{k_\dashv} ((PD_{min}(t).\Phi_k - PD_{min}(t).\varepsilon_k) \\ &\qquad \cdot (P_{X,c_j}(t).\Phi_k - P_{X,c_j}(t).\varepsilon_k) \cdot LT_k) \\ &\int_{max(t_l^-, t_X^-)}^{min(t_l^+, t_X^+)} PD_{max}(t) \cdot P_{X,c_j}(t)dt \\ &= \sum_{k=k_\vdash}^{k_\dashv} ((PD_{max}(t).\Phi_k + PD_{max}(t).\varepsilon_k) \\ &\qquad \cdot (P_{X,c_j}(t).\Phi_k + P_{X,c_j}(t).\varepsilon_k) \cdot LT_k), \end{aligned}$$

where $k_\vdash$ and $k_\dashv$ are the first and last unit time intervals intersecting $[max(t_l^-, t_X^-), min(t_l^+, t_X^+)]$, and LT_k is the length of intersection of the kth unit time intervals and $[max(t_l^-, t_X^-), min(t_l^+, t_X^+)]$.

Subsequently, we estimate the final lower and upper bounds of p-distance of X based on two assumptions. First, to estimate the final upper bound of X, we assume that all unseen pieces of X are farthest to Q and will add to the maximum possible p-distance to X. Hence, $UB_{est} = UB_{cur} + \frac{N \cdot (t_e - t_s - AggP)}{t_e - t_s}$. Second, to estimate the final lower bound of X, we assume that all unseen pieces of X are being the nearest to Q among all unseen pieces and will add to the minimum possible p-distance, the key value of the top entry in H, to X. As a result, $LB_{est} = LB_{cur} + \frac{top(H).key \cdot (t_e - t_s - AggP)}{t_e - t_s}$.

After that, if $UB_{est} < k_{up}$, we update k_{up} to be the kth current minimum estimated final upper bound of p-distance of trajectories. If $LB_{est} > k_{up}$, we mark the entry corresponding to trajectory X as "pruned," because the p-distance of X will be at least higher than k other trajectories.

The process described above is encapsulated in a function *UpdateLcand* (called in line 15 in Algorithm 1), which is shown as Algorithm 3.

Algorithm 3: UpdateLcand($(c_j, key, [t_l^-, t_l^+], PD_{min}(t), PD_{max}(t))$, L_{cand})

```
1  for every piece (X, [t_X^-, t_X^+], P_{X,c_j}(t)) in the buckets of c_j do
2      if [t_X^-, t_X^+] intersect with [t_l^-, t_l^+] then
3          Find the entry (X, AggP, LB_cur, UB_cur) in L_cand;
4          if There is no entry of X in L_cand then
5              Create a new entry (X, AggP, LB_cur, UB_cur);
6              L_cand ← L_cand ⋃ (X, AggP, LB_cur, UB_cur);
7          if X is not marked as "pruned" then
8              L_cand ← L_cand − (X, AggP, LB_cur, UB_cur);
9              Update (X, AggP, LB_cur, UB_cur) according
               to Equation 9;
10             L_cand ← L_cand ⋃ (X, AggP, LB_cur, UB_cur);
11             LB_est ← LB_cur + (top(H).key·(t_e − t_s − AggP)) / (t_e − t_s) ;
12             UB_est ← UB_cur + (N·(t_e − t_s − AggP)) / (t_e − t_s);
13             if UB_est < k_up then
14                 Update k_up;
15             if LB_est > k_up then
16                 Mark X as "pruned";
```

5.4 Cost Analysis

In this section, we give a brief cost analysis for the filtering phase of Algorithm 1. Such an analysis requires the distributions and other relevant characteristics of uncertain trajectories, which are very complex to capture. To ease our analysis, we make a few simplifying assumptions. We assume at the beginning of the time interval $[t_s, t_e]$, the moving objects are uniformly distributed in the entire data space, and during $[t_s, t_e]$, each object is moving toward a random direction, and it may change direction after a random period of time.

In a 2D unit domain space, there are totally I^2 cells in the UTgrid. Given a database containing totally N objects that are uniformly distributed in the space, the density (sum probability of objects) of each cell is $\frac{N}{I^2}$ during $[t_s, t_e]$.

The UTgrid cells contain lightweight information that can be stored in main memory. So, the most expensive part of Algorithm 1 involves access and computation based on the probability functions of pieces stored in buckets. Note that every time when a trajectory piece in a bucket is visited, only the part of probability function intersecting the valid time is needed for computation. Therefore, we measure the cost of Algorithm 1 by the total "portion" of probability functions of trajectory pieces that are used for computation during query processing. We denote it as $\mathcal{P}_{com}$.

To ease the explanation, we consider each valid time interval as a set of infinite discrete time instants. Let $M_{vst}(t)$ be the number of cells visited corresponding to a time instant t, we have the following equation:

$$\mathcal{P}_{com} = \frac{\int_{t_s}^{t_e} \left(\frac{N}{I^2} \cdot M_{vst}(t)\right) dt}{N \cdot |t_e - t_s|}. \tag{10}$$

Now, we discuss $M_{vst}(t)$. Recall that Algorithm 1 visits cells one by one in ascending order of their possible p-distance. For any two time instants t and t', we partition the cells into tracks based on the corresponding center cells of t and t'. Because the density of each cell is constantly $\frac{N}{I^2}$, the cells in the ith track of time t and ith track of time t' have the same possible p-distance. Therefore, $M_{vst}(t)$ corresponding to each time instant is the same, and (10) becomes

$$\mathcal{P}_{com} = \frac{M_{vst}(t)}{I^2}. \quad (11)$$

For each trajectory, we assume that only Δ "portion" of possible locations at each time instant can be found outside a circle centered at the reported location and with a radius r.

Given a data set with N trajectories, for a time instant t, we sort the reported location of N trajectories in ascending order of their distance to Q; therefore, each trajectory gets a rank. Intuitively, the k trajectories that have the smallest p-distance should be close to Q during most of Q's lifespan and have relatively higher ranks. We assume that the k resulting trajectories are always ranked higher than K during $[t_s, t_e]$, where $K \geq k$. We call any time t an outlier when this assumption does not hold and use p to denote the percentage of time that are outliers.

Let D be the distance between the reported location of the Kth nearest object and Q. Since the objects are uniformly distributed in a space, we have $\frac{\pi D^2}{1} = \frac{K}{N}$, so $D = \sqrt{\frac{K}{\pi \cdot N}}$. In $1-p$ of Q's lifespan, $(1-\Delta)$ "portion" of the k answer trajectories can be found within a circle with Q as center and $D+r$ as radius. Such a circle is covered by totally $M_{cover} = (\frac{2(D+r)}{\frac{1}{I}})^2$ cells. We assume that unseen portion of answer trajectories all have the maximum possible instant p-distance N. Then, the maximum instant p-distance of k answers equals

$$\mathcal{D}_p(X,t) = \frac{N}{I^2} \cdot M_{cover} \cdot (1-\Delta) + \Delta \cdot N. \quad (12)$$

For the other p of Q's lifespan, we also assume the maximum possible instant p-distance N. Then, the maximum interval p-distance of k answers equals

$$k_{up} = \frac{\mathcal{D}_p(X,t)(1-p) \cdot (t_e - t_s) + N \cdot p \cdot (t_e - t_s)}{t_e - t_s}. \quad (13)$$

After M_{cover} cells are visited, Algorithm 1 continues to visit other cells until the minimum possible instant p-distances of remaining cells are no less than k_{up}, which means the number of cells visited ($M_{vst}(t)$) is equal to $\frac{k_{up}}{\frac{N}{I^2}}$. Then, based on (11), (12), and (13), we have

$$\mathcal{P}_{com} = \left(4\left(\sqrt{\frac{K}{\pi \cdot N}} + r\right)^2 \cdot (1-\Delta) + \Delta\right) \cdot (1-p) + p. \quad (14)$$

In most real applications, the probability of an uncertain object being outside a certain bound is very small, which actually means for most data sets Δ equals 0, and r becomes a parameter of the data set. Then, (14) becomes

$$\mathcal{P}_{com} = 4\left(\sqrt{\frac{K}{\pi \cdot N}} + r\right)^2 \cdot (1-p) + p. \quad (15)$$

Note that $\mathcal{P}_{com}$ increases as K and p increases. However, these two values cannot increase at the same time. More details are provided elsewhere [19]. We will further investigate the value of $\mathcal{P}_{com}$ in the experiments in Section 6.

6 Experiments

We present the experimental results in this section. All the experiments are run on a desktop PC with a 2.66-GHz CPU and 4-GB RAM. The page size is set to 4,096 bytes. The details of data sets and queries are given in Section 6.1. The quality of p-distance is evaluated in Section 6.2. The performance of UTgrid is studied in Section 6.3. Finally, KSQs are evaluated in Section 6.4.

6.1 Generating Data Sets and Queries

Real data. We extract 8,089 2D trajectories from the Geolife GPS trajectory data set [14]. The average number of time stamps per trajectory is over 3,000. In our experiments, we consider only the first 128 positions of each trajectory. We normalize the positions of trajectories into a unit space. We obtain uncertain trajectories by attaching a probability distribution function to each certain trajectory. In particular, the probability distribution function of each trajectory is randomly chosen as constrained-uniform or constrained-Gaussian distributions (the same type of distributions used elsewhere [28]). Given a certain position p, a constrained uniform (constrained Gaussian) means that all possible locations follow uniform (Gaussian) distribution within a circle centered at p and with a radius r, where r is randomly chosen from [0.02, 0.04] in all experiments.

Synthetic data. We first generate certain trajectories with the same length 128 based on the GSTD data generator [29]. We then convert these certain trajectories to uncertain trajectories in the same way as described above.

Queries. The query trajectory of a KSQ is a trajectory randomly chosen from the database. Given a chosen uncertain trajectory, only its expected locations at all sample times are used to form the certain query Q. For every experiment, we run 100 queries, and the result presented is the average on the 100 queries.

We measure the pruning power, total query processing time, and the number of disk page accesses (I/O) in our experiments. The pruning power is measured by the percentage of objects whose exact p-distances are computed in the refinement phase.

6.2 The Effect of p-Distance

We test the effect of p-distance by comparing it with the expected euclidean distance (denoted as ED) on a set of synthetic data of different distributions. We generate three basic synthetic data sets with uniform, Gaussian and clustered distributions, respectively. Each data set contains 10,000 uncertain trajectories with the same length 128.

Quality of p-distance. We first test the quality of p-distance. A KSQ is aimed at retrieving uncertain trajectories that are the most similar to the query trajectory Q. Those trajectories should be nearer to Q than others for most of the

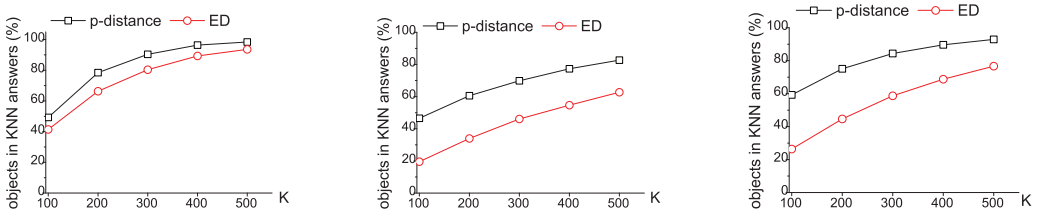


Fig. 6. Quality of p-distance.

time. Therefore, we evaluate the quality of KSQ answers as follows: We choose equally 512 time instants from the time span of Q. For each chosen time instant t, we perform a Topk-probable nearest neighbor (PNN) [3] and get the K objects with the highest probability to be nearest to Q at time t. On the other hand, we find the 100 KSQ answers and calculate the average percentage of KSQ answers that are within the Topk-PNN answers at each chosen time instant. The idea behind the evaluation is that the more effective p-distance is, the more of the KSQ answers should be within the Topk-PNN answers at corresponding time instants.

We vary the K in Topk-PNN queries from 100 to 500. The results are shown in Fig. 6. Clearly, p-distance works better than ED on all three distributions. The advantage of p-distance is more noticeable on Gaussian and clustered distributions, which indicates that p-distance is much more adaptive to realistic data than ED.

Stableness of p-distance. Next, we test the stableness of p-distance on data sets with outliers. An outlier is a faraway location point in an uncertain trajectory X, appearing within a random time interval inside X's lifespan. The stableness reflects the tolerance of outliers of a similarity metric. In particular, given a query trajectory Q and a basic data set that does not contain outliers, the 100 trajectories with the smallest p-distance (or ED) are treated as the standard answer set, denoted as S_{sta}. For a data set with outliers, we have a new set S_{new} of 100 trajectories with the smallest p-distance (or ED). Then, the stableness of p-distance (or ED) is measured by the percentage of objects in S_{sta} that remain in S_{new}, i.e., $|S_{sta} \cap S_{new}|/|S_{sta}|$.

In this set of experiments, for each basic data set described in Section 6.2, we randomly add outliers to some trajectories in it so that the sum of all outlier probabilities achieves a certain value Δ.

We assume that the faraway outlier locations are beyond the unit data space. In particular, we test three cases when the ED (denoted as d_{far}) between the locations of the query object and an outlier is 2, 5, and 10, respectively. Note that when $d_{far} > 1$ the instant p-distance of an outlier always equals the maximum possible instant p-distance N, where N is the cardinality of the data set. Therefore, the results of p-distance are the same for the three different d_{far} values.

We vary Δ from 0 to 2.5 percent, and the results are shown in Fig. 7. The answers based on p-distance are very stable in all used settings. More than 70 percent of answers remain even when 2.5 percent locations of trajectories in a data set are outliers. In contrast, the stableness of ED decreases dramatically as d_{far} gets larger or the percentage of outliers increases, especially for nonuniform data sets. In the worst case, more than 80 percent standard answers are missing (Gaussian data set, $d_{far} = 10$). We conclude that p-distance is much more stable than ED.

6.3 Effects of UTgrid Parameters

In this section, we study UTgrid's behavior under the settings of various parameters on a synthetic data set of Gaussian distribution.

Effect of the number of intervals I. We vary I from 8 to 64. We first study the memory cost of UTgrid with varying I. As we can see from Fig. 8a, the memory cost of UTgrid increases quadratically as I increases. However, even when I is up to 64, which means that UTgrid contains 4,096 cells, the memory cost of UTgrid is still less than 3 MB.

Next, we set the value of U to 8 and test the effect of I on the performance of UTgrid. As shown by Fig. 8b, the pruning power of UTgrid improves apparently as I increases, and then flattens after I goes beyond 48. As demonstrated in Fig. 8c, the query processing time first decreases as I increases because increasing pruning power means that the exact p-distance of less objects are computed in the refinement. However, when I goes beyond 48, the time slightly increases. This is because as I increases the

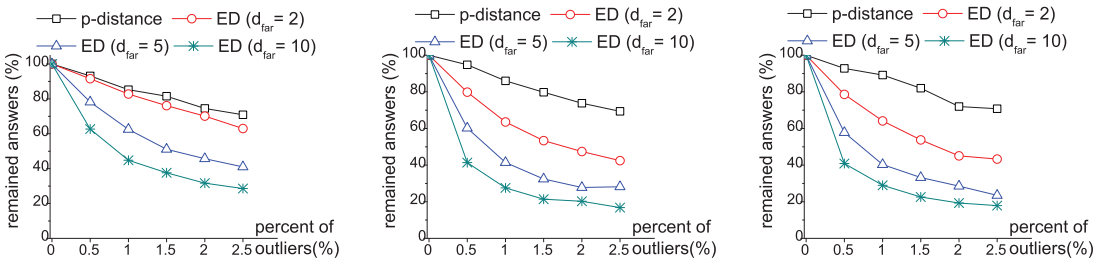


Fig. 7. Stableness of p-distance.

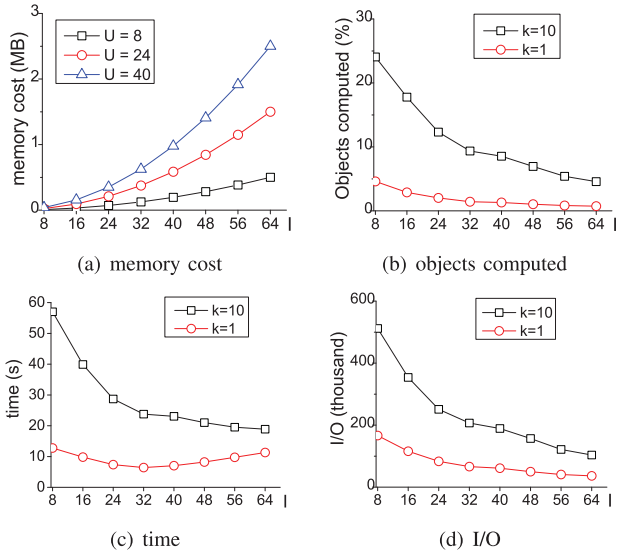


Fig. 8. Effect of the number of intervals I.

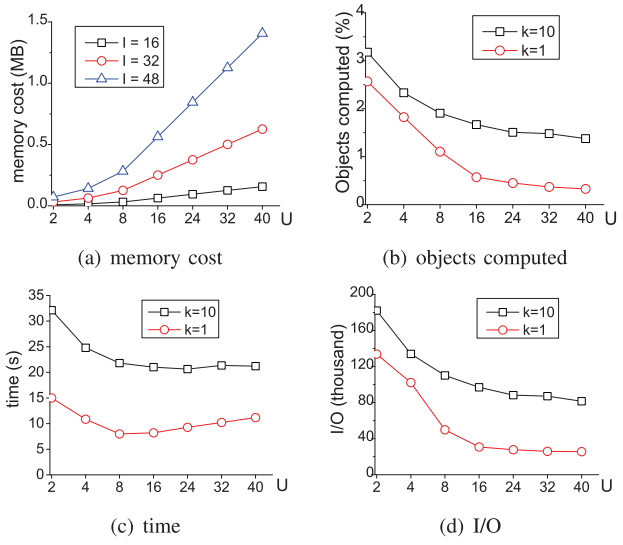


Fig. 9. Effect of the number of unit time intervals U.

cost of visiting UTgrid also increases. When I is larger than 48, the UTgrid visiting overhead overweighs the negligible effect of the slight improvement in pruning power. The variance of I/O is similar to that of the percentage of objects left after pruning with UTgrid, due to the fact that most I/Os occur in the refinement phase. We chose 48 as the value of I for each dimension based on the above results.

Effect of the number of unit time intervals U. We vary U from 2 to 40. The effect of different U is shown in Fig. 9. As shown by the results, the memory cost of UTgrid increases as U increases but is less than 1.5 MB under all circumstances in this set of experiments. The pruning power of UTgrid improves apparently as U increases, and then, flattens very soon after U goes beyond 24. The query processing time first decreases as U increases and then slightly increases when U goes beyond 16 or 24, due to the increasing UTgrid visiting overhead. The variance of I/O is similar to the variance of percentage of objects left after pruning with UTgrid. We use 16 as the value of U based on the above results in the following experiments.

6.4 Query Performance Comparison

No previous technique addresses KSQs for uncertain trajectories. We compare our UTgrid approach with two alternatives, namely *BoundPruning* and sequential scan. The BoundPruning method uses only the bounds rather than probabilities to compute the lower and upper bounds of p-distance (denoted as $LB|_{t_s}^{t_e}$ and $UB|_{t_s}^{t_e}$) of trajectories. Specifically, at any time instant $\hat{t}$, given the circular region of the constrained probability distribution function of a trajectory X, we know the minimum and maximum distance between X and Q, denoted as $X.d_{min}$ and $X.d_{max}$. Let $P_{low}(X \succ Y, \hat{t})$ and $P_{up}(X \succ Y, \hat{t})$ be the lower and upper bounds of the probability of a trajectory Y being nearer to Q than X at time $\hat{t}$. $P_{low}(X \succ Y, \hat{t})$ is 1 if $Y.d_{max} \leq X.d_{min}$ and 0 otherwise. $P_{up}(X \succ Y, \hat{t})$ is 1 if $Y.d_{min} \leq X.d_{max}$ and 0 otherwise. Then, the lower and upper bounds of instant p-distance of X at time $\hat{t}$ (denoted as $LB(X, \hat{t})$ and $UB(X, \hat{t})$) are the sum of $P_{low}(X \succ Y, \hat{t})$ and $P_{up}(X \succ Y, \hat{t})$, respectively, where Y is every trajectory in the database. Therefore, $LB(X, \hat{t}) = \sum_{\forall Y \in DB} P_{low}(X \succ Y, \hat{t})$ and $UB(X, \hat{t}) = \sum_{\forall Y \in DB} P_{up}(X \succ Y, \hat{t})$. Then, $LB|_{t_s}^{t_e}(X)$ and $UB|_{t_s}^{t_e}(X)$ are the expectation of $LB(X, \hat{t})$ and $UB(X, \hat{t})$ during Q's lifespan, respectively. In query processing, we keep the kth minimum upper bound of p-distance in memory, denoted as k_{up}. Any trajectory with $LB|_{t_s}^{t_e}$ larger than k_{up} is pruned without further computing in the refinement phase.

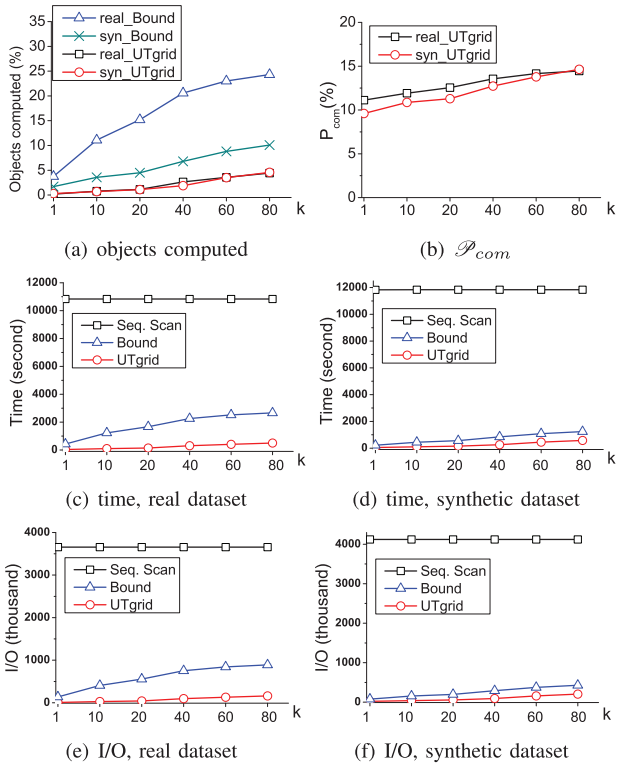
Besides the pruning power, query processing time and the number of disk accesses (I/O), for UTgrid, we also measure the total "portion" of probability functions of pieces that are used for computation ($\mathcal{P}_{com}$) in this set of experiments.

The value of k. We vary the value of k from 1 to 80 in this set of experiments. The size of both real and synthetic data sets is 8,089. The results are shown in Fig. 10.

As shown in Fig. 10a, the pruning power for all techniques decreases as k increases. Nevertheless, as k increases, the pruning power of UTgrid increases more slowly than BoundPruning. When $k = 80$ which means the selectivity of KSQs is 1 percent, UTgrid can still prune more than 95 percent objects. This again indicates that UTgrid is effective in indexing uncertain trajectories. On the other hand, for BoundPruning in the worst case, nearly 25 percent objects are left ($k = 80$, real data set) after pruning.

The results also show that the total response time of UTgrid is much smaller than the other two techniques on both data sets. Note that as demonstrated in Fig. 10b, on both real and synthetic data sets, $\mathcal{P}_{com}$ increases as k increases, which indicates that time cost of visiting UTgrid also increases. However, since UTgrid can always prune most of the objects and leave a candidate list with less than 5 percent objects to the refinement phase, the total time cost of UTgrid is still much smaller than the other two techniques, specially on the real data set. The speedup of UTgrid compared to BoundPruning is up to eight times.

The variance of I/O of all techniques is similar to percentage of objects computed in the refinement phase, according to Figs. 10e and 10f. This is because most I/Os occur in the refinement.

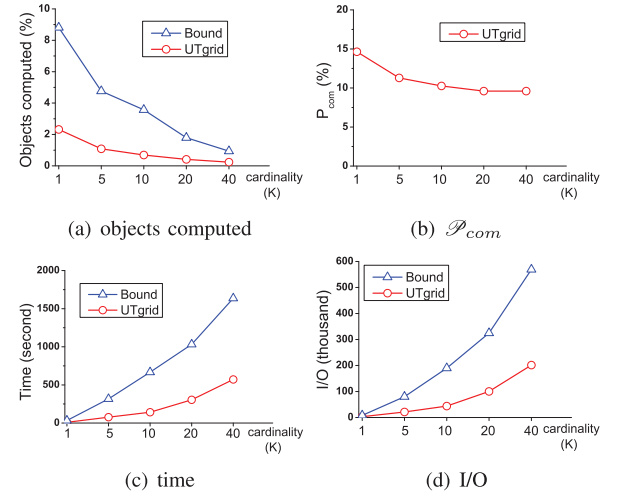


(a) objects computed (b) $\mathscr{P}_{com}$

(c) time, real dataset (d) time, synthetic dataset

(e) I/O, real dataset (f) I/O, synthetic dataset

Fig. 10. Effect of k.



(a) objects computed (b) $\mathscr{P}_{com}$

(c) time (d) I/O

Fig. 11. Effect of data cardinality.

Data set cardinality. We vary the synthetic data set cardinality from 1,000 to 40,000 and fix k to 10. The results are shown in Fig. 11. We omit the sequential scan because its cost is too high. The UTgrid scales very well as data set cardinality increases. For UTgrid, the percentage of objects computed in refinement phase and $\mathcal{P}_{com}$ slowly decreases because with the same value of k the selectivity of KSQs decreases as the cardinality increases. Hence, the superiority of UTgrid becomes more significant as data set cardinality increases.

7 Conclusion

In this paper, we addressed top-k similarity query (KSQ) on uncertain trajectories. We introduced a novel notion p-distance to measure the dissimilarity between uncertain trajectories. Accordingly, we proposed KSQ on uncertain trajectories, which returns from an uncertain trajectory set the k ones that have the smallest p-distances with respect to a given query trajectory. To process such KSQs efficiently, we proposed an effective index structure UTgrid to index a large set of uncertain trajectories. In addition, we developed query algorithms that exploit UTgrid to achieve effective pruning and efficient query processing. Finally, we conducted an extensive experimental study on both real and synthetic data sets to evaluate our proposals. The results show that our proposals are effective, efficient, and scalable.

For future work, it is interesting to extend the KSQ algorithms to handle uncertain query trajectories. It is also worth accommodating more complex models for uncertain trajectories, for example, trajectories with varying probability distribution functions.

References

[1] A. Anagnostopoulos, M. Vlachos, M. Hadjieleftheriou, E.J. Keogh, and P.S. Yu, "Global Distance-Based Segmentation of Trajectories," *Proc. ACM SIGKDD Int'l Conf. Knowledge Discovery and Data Mining*, pp. 34-43, 2006.

[2] J. Aßfalg, H.-P. Kriegel, P. Kröger, and M. Renz, "Probabilistic Similarity Search for Uncertain Time Series," *Proc. Int'l Conf. Scientific and Statistical Database Management*, pp. 435-443, 2009.

[3] G. Beskales, M.A. Soliman, and I.F. Ilyas, "Efficient Search for the Top-k Probable Nearest Neighbors in Uncertain Databases," *Proc. VLDB Endowment*, vol. 1, pp. 326-339, 2008.

[4] Bikely, http://www.bikely.com/, 2013.

[5] V. Botea, D. Mallett, M.A. Nascimento, and J. Sander, "Pist: An Efficient and Practical Indexing Technique for Historical Spatio-Temporal Point Data," *GeoInformatica*, vol. 12, no. 2, pp. 143-168, 2008.

[6] Y. Cai and R.T. Ng, "Indexing Spatio-Temporal Trajectories with Chebyshev Polynomials," *Proc. ACM SIGMOD Int'l Conf. Management of Data*, pp. 599-610, 2004.

[7] V.P. Chakka, A. Everspaugh, and J.M. Patel, "Indexing Large Trajectory Data Sets with SETI," *Proc. First Biennial Conf. Innovative Data Systems Research*, pp. 164-175, 2003.

[8] L. Chen, M.T. Özsu, and V. Oria, "Robust and Fast Similarity Search for Moving Object Trajectories," *Proc. ACM SIGMOD Int'l Conf. Management of Data*, pp. 491-502, 2005.

[9] R. Cheng, D.V. Kalashnikov, and S. Prabhakar, "Querying Imprecise Data in Moving Object Environments," *IEEE Trans. Knowledge Data Eng.*, vol. 16, no. 9, pp. 1112-1127, Sept. 2004.

[10] B.S.E. Chung, W.-C. Lee, and A.L.P. Chen, "Processing Probabilistic Spatio-Temporal Range Queries over Moving Objects with Uncertainty," *Proc. 12th Int'l Conf. Extending Database Technology: Advances in Database Technology*, pp. 60-71, 2009.

[11] G. Cormode, F. Li, and K. Yi, "Semantics of Ranking Queries for Probabilistic Data and Expected Ranks," *Proc. Int'l Conf. Data Eng. (ICDE)*, pp. 305-316, 2009.

[12] P. Cudré-Mauroux, E. Wu, and S. Madden, "Trajstore: An Adaptive Storage System for Very Large Trajectory Data Sets," *Proc. Int'l Conf. Data Eng. (ICDE)*, pp. 109-120, 2010.

[13] E. Frentzos, K. Gratsias, and Y. Theodoridis, "Index-based Most Similar Trajectory Search," *Proc. Int'l Conf. Data Eng. (ICDE)*, pp. 816-825, 2007.

[14] Geolife GPS Trajectories, http://research.microsoft.com/en-us/downloads/b16d359d-d164-469e-9fd4-daa38f2b2e13/default.aspx., 2013.

[15] GPS Sharing, http://gpssharing.com/, 2013.

[16] M. Hua, J. Pei, W. Zhang, and X. Lin, "Ranking Queries on Uncertain Data: A Probabilistic Threshold Approach," *Proc. ACM SIGMOD Int'l Conf. Management of Data*, pp. 673-686, 2008.

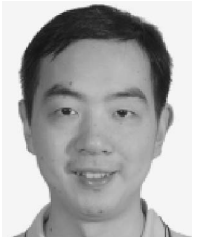
[17] J. Li, B. Saha, and A. Deshpande, "A Unified Approach to Ranking in Probabilistic Databases," *Proc. VLDB*, vol. 2, pp. 502-513, 2009.

[18] X. Lian, L. Chen, and J.X. Yu, "Pattern Matching Over Cloaked Time Series," *Proc. Int'l Conf. Data Eng. (ICDE)*, pp. 1462-1464, 2008.

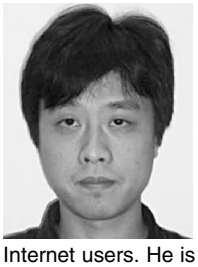
[19] C. Ma, H. Lu, L. Shou, and G. Chen, "KSQ: Top-k Similarity Query on Uncertain Trajectories," technical report, http://db.zju.edu.cn/s/techreport/ksq.html, 2012.
[20] J. Ni and C.V. Ravishankar, "Pa-Tree: A Parametric Indexing Scheme for Spatio-Temporal Trajectories," *Proc. Int'l Symp. Spatial and Temporal Databases,* pp. 254-272, 2005.
[21] J.M. Patel, Y. Chen, and V.P. Chakka, "Stripes: An Efficient Index for Predicted Trajectories," *Proc. ACM SIGMOD Int'l Conf. Management of Data,* pp. 637-646, 2004.
[22] N. Pelekis, I. Kopanakis, E.E. Kotsifakos, E. Frentzos, and Y. Theodoridis, "Clustering Uncertain Trajectories," *Knowledge Information Systems,* vol. 28, no. 1, pp. 117-147, 2011.
[23] D. Pfoser and C.S. Jensen, "Capturing the Uncertainty of Moving-Object Representations," *Proc. Int'l Symp. Advances in Spatial Databases,* pp. 111-132, 1999.
[24] D. Pfoser, C.S. Jensen, and Y. Theodoridis, "Novel Approaches in Query Processing for Moving Object Trajectories," *Proc. Int'l Conf. Very Large Data Bases,* pp. 395-406, 2000.
[25] S. Rasetic, J. Sander, J. Elding, and M.A. Nascimento, "A Trajectory Splitting Model for Efficient Spatio-Temporal Indexing," *Proc. 31st Int'l Conf. Very Large Data Bases,* pp. 934-945, 2005.
[26] Y. Sakurai, M. Yoshikawa, and C. Faloutsos, "FTW: Fast Similarity Search under the Time Warping Distance," *Proc. 24th ACM SIGMOD-SIGACT-SIGART Symp. Principles of Database Systems (PODS),* pp. 326-337, 2005.
[27] E.M.A. Soliman, I.F. Ilyas, and K.C.-C. Changu, "Top-k Query Processing in Uncertain Databases," *Proc. Int'l Conf. Data Eng. (ICDE),* pp. 896-905, 2007.
[28] Y. Tao, R. Cheng, X. Xiao, W.K. Ngai, B. Kao, and S. Prabhakar, "Indexing Multi-Dimensional Uncertain Data with Arbitrary Probability Density Functions," *Proc. Int'l Conf. Very Large Data Bases,* pp. 922-9337, 2005.
[29] Y. Theodoridis, J.R.O. Silva, and M.A. Nascimento, "On the Generation of Spatiotemporal Data Sets," *Proc. Sixth Int'l Symp. Advances in Spatial Databases,* pp. 147-164, 1999.
[30] G. Trajcevski, R. Tamassia, H. Ding, P. Scheuermann, and I.F. Cruz, "Continuous Probabilistic Nearest-Neighbor Queries for Uncertain Trajectories," *Proc. 12th Int'l Conf. Extending Database Technology: Advances in Database Technology,* pp. 874-885, 2009.
[31] G. Trajcevski, O. Wolfson, F. Zhang, and S. Chamberlain, "The Geometry of Uncertainty in Moving Objects Databases," *Proc. 12th Int'l Conf. Extending Database Technology: Advances in Database Technology,* pp. 233-250, 2002.
[32] M. Vlachos, M. Hadjieleftheriou, D. Gunopulos, and E.J. Keogh, "Indexing Multi-Dimensional Time-Series with Support for Multiple Distance Measures," *Proc. ACM SIGKDD Int'l Conf. Knowledge Discovery and Data Mining,* pp. 216-225, 2003.
[33] M.-Y. Yeh, K.-L. Wu, P.S. Yu, and M.-S. Chen, "Proud: A Probabilistic Approach to Processing Similarity Queries over Uncertain Data Streams," *Proc. 12th Int'l Conf. Extending Database Technology: Advances in Database Technology,* pp. 684-695, 2009.
[34] S.M. Yuen, Y. Tao, X. Xiao, J. Pei, and D. Zhang, "Superseding Nearest Neighbor Search on Uncertain Spatial Databases," *IEEE Trans. Knowledge Data Eng.,* vol. 22, no. 7, pp. 1041-1055, July 2010.
[35] M. Zhang, S. Chen, C.S. Jensen, B.C. Ooi, and Z. Zhang, "Effectively Indexing Uncertain Moving Objects for Predictive Queries," *Proc. VLDB,* vol. 2, pp. 1198-1209, 2009.
[36] X. Zhang and J. Chomicki, "On the Semantics and Evaluation of Top-K Queries in Probabilistic Databases," *Proc. Int'l Conf. Data Eng. Workshop,* pp. 556-563, 2008.
[37] K. Zheng, G. Trajcevski, X. Zhou, and P. Scheuermann, "Probabilistic Range Queries for Uncertain Trajectories on Road Networks," *Proc. Int'l Conf. Extending Database Technology,* pp. 283-294, 2011.

Chunyang Ma received the BSc degree in computer science from Zhejiang University, China, in 2006. Since 2006, she has been working toward the PhD degree in the College of Computer Science, Zhejiang University. From May 2008 to April 2009, she was a visiting student in the Department of Computer Science and Software Engineering, University of Melbourne. Her research interests include spatial database, spatiotemporal database and data access methods.

Hua Lu received the BSc and MSc degrees from Peking University, China, in 1998 and 2001, respectively, and the PhD degree in computer science from the National University of Singapore, in 2007. He is an associate professor in the Department of Computer Science, Aalborg University, Denmark. His research interests include databases, geographic information systems, as well as mobile computing. Recently, he has been working on indoor spatial awareness, complex queries on spatial data with heterogeneous attributes, and location privacy in mobile services. He has served on the program committees fof conferences and workshops including ICDE, SSTD, MDM, PAKDD, APWeb, and MobiDE. He is a PC cochair or vice chair for ISA 2011, MUE 2011, and MDM 2012. He is a member of the IEEE.



Lidan Shou received the PhD degree in computer science from the National University of Singapore. He is an associate professor in the College of Computer Science, Zhejiang University, China. Prior to joining the faculty, he worked in the software industry for more than two years. His research interests include spatial database, data access methods, visual and multimedia databases, and web data mining. He is a member of ACM.



Gang Chen received the PhD degree in computer science from Zhejiang University. He is a professor in the College of Computer Science and the director of the Database Lab, Zhejiang University. He has successfully led the investigation in research projects that aim at building China's indigenous database management systems. His research interests range from relational database systems to large-scale data management technologies supporting massive Internet users. He is a member of ACM and a senior member of China Computer Federation.