The VLDB Journal (2011) 20:819–840
DOI 10.1007/s00778-011-0225-y

REGULAR PAPER

Shooting top-*k* stars in uncertain databases

Xiang Lian · Lei Chen

Received: 13 November 2009 / Revised: 26 November 2010 / Accepted: 14 February 2011 / Published online: 9 March 2011

Abstract Query processing in the uncertain database has played an important role in many real-world applications due to the wide existence of uncertain data. Although many previous techniques can correctly handle precise data, they are not directly applicable to the uncertain scenario. In this article, we investigate and propose a novel query, namely *probabilistic top-k star* (PT*k*S) query, which aims to retrieve *k* objects in an uncertain database that are "closest" to a static/dynamic query point, considering both distance and probability aspects. In order to efficiently answer PT*k*S queries with a static/moving query point, we propose effective pruning methods to reduce the PT*k*S search space, which can be seamlessly integrated into an efficient query procedure. Finally, extensive experiments have demonstrated the efficiency and effectiveness of our proposed PT*k*S approaches on both real and synthetic data sets, under various parameter settings.

Keywords Probabilistic top-*k* star query · Uncertain databases · *k*-NN query · Moving object query

1 Introduction

Recently, query processing over uncertain data has become increasingly important in many real applications like *location-based services* (LBS) [13,29], sensor network monitoring [15], object identification [4], and moving object search [8,10,27]. In many of these applications, data are inherently uncertain and imprecise. For example, sensory data collected from sensors are often noisy due to environmental factors, network latency, or device failures. In other applications such as LBS, the uncertainty of users' locations is caused not only by the accuracy of positioning devices (e.g., GPS, GSM, or RFID), but also by human's intentional injection (e.g., blurring or distorting trajectory data) for the sake of privacy preserving [32]. Therefore, it is essential for these applications to *efficiently* and *effectively* handle uncertain data.

X. Lian · L. Chen (✉)
Department of Computer Science and Engineering,
The Hong Kong University of Science and Technology,
Clear Water Bay, Kowloon, Hong Kong, China
e-mail: leichen@cse.ust.hk

X. Lian
e-mail: xlian@cse.ust.hk

1.1 Uncertainty model

In the literature of uncertain database [6,9,36], uncertain data objects are assumed to be definitely part of the database, whereas their attributes are imprecise and uncertain. Thus, each uncertain object can be modeled by an *uncertainty region* with an uncertain interval on each attribute. Within the uncertainty region, object distribution can be represented by either discrete instances/samples [22,23,33] or continuous *probability density function* (pdf) [6,11]. Practically, the pdf function of an uncertain object is often unavailable explicitly [33]. In most cases, a set of discrete instances is used to represent the distribution of an object.

Figure 1a illustrates an example of the uncertain database in the mixed reality game, where 4 GPS-equipped players u, v, w, and x are located in a two-dimensional space. Due to the inaccuracy of GPS devices, the positions of these GPS players can be imprecise, which can thus be modeled as uncertain objects. For example, player u (i.e., uncertain object) has three different reported positions $\{u_1, u_2, u_3\}$, and users v, w, and x have two possible positions each (i.e., $\{v_1, v_2\}$, $\{w_1, w_2\}$, and $\{x_1, x_2\}$, respectively). These reported positions exactly correspond to instances for each uncertain object. Inferred from historical GPS data, we

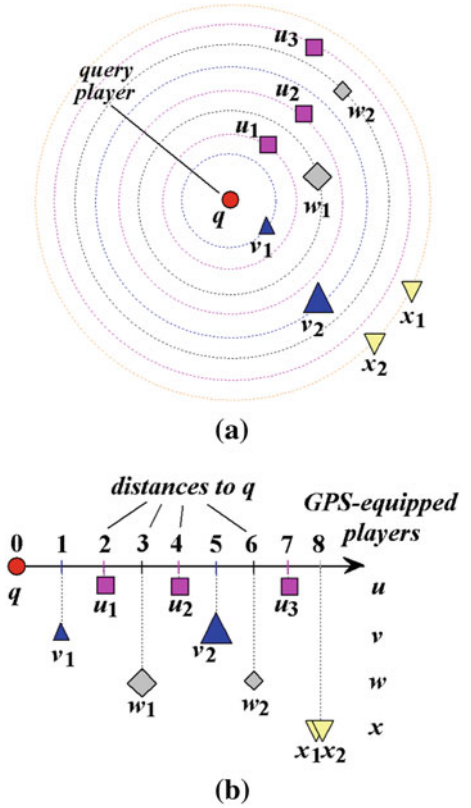

Fig. 1 An example of the mixed reality game. **a** Possible positions of GPS users, **b** distances from instances to q

Table 1 Uncertain database in Fig. 1

Uncertain object t	Instances t_i	Distances $dist(q, t_i)$	Appearance probability $t_i.p$
u	u_1	2	1/3
	u_2	4	1/3
	u_3	7	1/3
v	v_1	1	1/5
	v_2	5	4/5
w	w_1	3	2/3
	w_2	6	1/3
x	x_1	8	1/2
	x_2	8	1/2

can obtain the accuracy of such reported locations. In particular, as depicted in Table 1, we associate each possible position (instance) t_i of object t with an *appearance probability* $t_i.p \in (0, 1]$ indicating the probability that this object t appears at the location of t_i (e.g., object v appears at locations v_1 and v_2 with probabilities 1/5 and 4/5, respectively).

Following the convention of uncertain databases [10–12, 33], all instances t_i of an uncertain object t are considered to be *mutually exclusive* to exist (e.g., object u cannot appear at two locations u_1 and u_2 at the same time); uncertain object t must appear at one location of some instance t_i in reality (i.e., $\sum_{\forall t_i \in t} t_i.p = 1$); moreover, uncertain objects are assumed to be independent of each other.

1.2 Motivation example

In the previous example of the mixed reality game (as shown in Fig. 1a), each player tends to attack other players (enemies) nearby. Now assume that there is a query player q, who wants to attack and meanwhile watch out for his/her close players. This is because those players close to him/her may potentially attack him/her. However, due to the uncertain positions of players, it is not trivial for the query player q to analyze players' positions, and identify those enemies who are the most dangerous to him/her.

Figure 1b illustrates the distances from positional instances of players (in Fig. 1a) to query point q on a horizontal axis, where the appearance probability of each instance is given in Table 1. In the figure, instance v_1 resides at a position with the smallest distance to q (i.e., $dist(q, v_1) = 1$) and with the appearance probability $1/5 (= v_1.p)$. In other words, within the circle centered at q and with radius 1, object v has the highest appearance probability, compared with other objects.

On the other hand, although instance u_1 has the second smallest distance to q (i.e., $dist(q, u_1) = 2 > dist(q, v_1)$), its appearance probability is greater than that of object v, that is, $u_1.p = 1/3 > v_1.p = 1/5$. In other words, within the circle centered at q and with radius 2, object u has the highest existence probability (even if compared with v with the smallest distance).

In the aforementioned example, there are two factors involved for an uncertain object t to be near to q, that is, the distance, $t_i.dist$, from its possible instance t_i to q and the probability, $t_i.prob$, that it appears within distance $t_i.dist$ from q. Intuitively, our desired answers are those players having positional instances with small distances $t_i.dist$ to q and high probabilities $t_i.prob$, compared with other objects. However, a trade-off between distance and probability, like the example of players v and u above, is always possible to occur. That is, v has closer distance to q but with lower appearance probability, whereas u has a bit larger distance to q by with higher probability. In this case where uncertain objects have conflicting qualities between distances to q and existence probabilities, it is challenging to determine which players are more likely to be dangerous (close) to query player q.

Note that, this problem would not be simply solved by setting up a probability threshold and returning objects with probability above the threshold. This is because if we are doing so, we might miss some important objects with probabilities a little lower than the threshold. For example, in Figure 1b, instance v_1 (with probability $v_1.p = 0.2$) will not

be counted to be near to q if we set the probability threshold to 0.3. However, player v indeed has chance (i.e., 0.2, though a bit smaller than 0.3) to be the closest one to q. At least, within a circle centered at q with radius 1, v is the only probable player nearest to q (i.e., other players have zero probabilities). On the other hand, even if players have appearance probabilities above the threshold, we are still facing the same fuzzy situation mentioned above. That is, we are not sure which player is the "nearest" (most dangerous) one to query player q (e.g., v or u?) considering both distance and probability dimensions.

Therefore, in this article, we propose a *probabilistic top-k star* (PTkS) query over such uncertain data. In particular, given a query point q, we say an uncertain object t is a candidate (called "star") of the PTkS query result, if it has at least one instance t_i *nearer* to q and/or with *higher* confidence (to be near q), compared with other objects' instances. In other words, if we conceptually convert all instances of objects into a 2D *distance-and-probability* space (with respect to attributes, distance $t_i.dist$ and probability $t_i.prob$, dynamically computed with q), then PTkS candidates are those objects containing *skyline points* [5] (instances) in the 2D space. In addition, we assign each PTkS candidate with a natural ranking score. The actual PTkS answers are the top-k candidates with the highest scores. This way, our PTkS query can capture both distance and probability aspects of an object to be "closest" to q, rather than one attribute at a time.

Previous studies on PNN [10,11,23] retrieve uncertain objects that are expected to be nearest neighbor of a query point q with probabilities greater than a threshold T_p, which is however non-trivial to specify. In the previous example of Fig. 1, even if we set $T_p = 0$, there are many PNN players, that is, v, u, and w, returned, which may exceed query player q's ability to monitor. Moreover, such PNN answers can only give the expected results, while ignoring those important answers with small probabilities, which may lead to attacks from enemy players due to the taking of such an expectation. In addition, if we consider the expected distance for each uncertain object, and conduct NN queries on the (certain) expected distance, then only player w is returned, which however may miss some important result such as player v, who indeed has some (though low) probability to be closest to q. In contrast, when $k = 2$, our PTkS query will return players v and w, who are the most dangerous (i.e., closest) players to q.

While PTkS computes skylines with dynamic attributes of both distance and probability, existing techniques for dynamic skyline processing [7,14,31,35] can only deal with dynamic distance attributes. Furthermore, previous studies on static skyline processing [5,21,31,38] are not applicable to our PTkS problem, since it is inefficient to materialize attributes for all instances or online build an index from scratch for the skyline retrieval. In this case, the materialization or index construction has to scan the entire database, which is very costly in terms of both computation and I/O costs. Thus, this motivates us to design effective pruning methods to reduce the PTkS search space and propose an efficient approach for online skyline retrieval in a streaming fashion (as discussed in Sect. 3).

1.3 Applications

The PTkS query has many practical applications, in which the query point can be either static or dynamic (i.e., moving).

1.3.1 PTkS query with static query point

In *location-based services* (LBS), a mobile user may ask "*what is the nearest taxi from my current location?*". Since taxis are moving around the city, their locations are uncertain (due to the inaccuracy of positioning devices, movement, or transmission delay). Naturally, the mobile user does not want to miss any (small) chance of catching his/her (actual) nearest taxi, which can save much precious time. Therefore, a PTkS query can be issued to find k "nearest" taxis considering both their distances to user's location and their existence probabilities within such distances (i.e., including near-by taxis even with low probability and a bit farther taxis with high probability). Note that, these k PTkS answers (taxis) are highly likely to be nearest to the query point (i.e., the mobile user), and they are all candidates for the mobile user to choose. Since a PTkS taxi with higher score indicates higher confidence that it is close to the mobile user, the mobile user can call these taxi candidates (the calling service can be provided by LBS application utilities) in descending order of their ranking scores (until a taxi is available), and confirm their availability. This way, the mobile user can efficiently find a satisfactory taxi close to oneself, due to the order ranked by scores.

In the coal mine surveillance application [43], sensors are deployed in the tunnel to collect data such as density of gas, oxygen, and dust, as well as temperature and humidity. Dangerous events like fires or gas leakage usually correspond to patterns (called *contour maps* [43]) in the collected sensory data. Thus, a mine manager may ask questions like "*which sensor(s) report data matching with the fire pattern?*". In order to keep the safety of workers, the manager cannot ignore any highly similar pattern matching even with low probability, or that with low similarity but high probability. Note that, here we aim to retrieve the "nearest" sensor data to the query fire pattern q. Thus, the PTkS answer set should include those sensor data with instances having either better distance or better probability dimension (i.e., with instances not dominated by others). Previous studies [10,11,23] that retrieves PNN objects cannot be used in such emergency applications, since they only consider the expected

probability which may ignore dangerous events with small probabilities and cause loss of lives. Thus, this scenario exactly corresponds to the PTkS problem which can identify dangerous events on noisy sensor data.

In the mixed reality games (e.g., Botfighters [20]), due to the movement of players, GPS inaccuracy, or the latency of the server, only imprecise locations of players can be obtained. Thus, one can conduct a PTkS query to find out "*what are the nearest enemies to me?*". This is because those enemies nearest to the player (even with small probability) may potentially attack the player, and they should be watched out for.

1.3.2 PTkS query with moving query point

One interesting extension of the applications above is to consider the PTkS problem with a moving query point instead of static one. For example, in LBS, a mobile user walks toward a direction, and s/he may occasionally ask for the "closest" taxis around himself/herself. In the battlefield, the troop may also move to a target while being on guard against the surrounding enemies. In the mixed reality games, the player may also move to a target while being on guard against the surrounding enemies.

1.4 Contributions

In this article, we formulate and tackle the problem of the PTkS query and make the following contributions.

- We identify and formalize the PTkS query in the context of uncertain databases which retrieves k uncertain objects in the database that are "nearest" to a static/dynamic query object, with the consideration of both distance and probability.
- We propose effective pruning methods to reduce the search space of the PTkS query with a static query point, which can be seamlessly integrated into our efficient query procedure.
- We extend the proposed solution with static query point to that with continuously moving query object.
- Last but not least, we conduct extensive experiments to verify the efficiency and effectiveness of our pruning methods and PTkS query procedures under various parameter settings.

The rest of the article is organized as follows. Section 2 formally defines our PTkS problem. Sections 3 and 4 propose effective pruning methods and efficient query processing approaches to answer PTkS queries, with static and dynamic query objects, respectively. Section 5 demonstrates the PTkS query performance. Finally, Sect. 6 reviews related works, and Sect. 7 concludes this article.

2 Problem definition

In this section, we formally define the *probabilistic top-k star* (PTkS) query over uncertain data. Specifically, assume we have a d-dimensional uncertain database $\mathcal{D}$ containing N uncertain objects. Each uncertain object $t \in \mathcal{D}$ consists of $|t|$ instances $t_1, t_2, \ldots$, and $t_{|t|}$, where each instance t_i ($1 \leq i \leq |t|$) is associated with an *appearance probability* $t_i.p$ satisfying $\sum_{i=1}^{|t|} t_i.p = 1$. Note that, in this article, we consider each uncertain object consisting of discrete instances. Our proposed approaches, however, can be easily extended to the case where pdf is known by sampling the continuous pdf [36].

As mentioned in Sect. 1.2, in the uncertain database, there is a trade-off between distance $t_i.dist$ and probability $t_i.prob$ for an uncertain object to be nearest to a query point q. Therefore, we can *conceptually* convert all instances of uncertain objects into a two-dimensional *distance-and-probability* space. In particular, for each instance t_i of an object t, we transform it to a 2D point $\langle t_i.dist, 1 - t_i.prob\rangle$ in a converted space, where the *distance*-axis, $t_i.dist$, corresponds to the Euclidean distance from q to t_i, denoted as $dist(q, t_i)$, and the *probability*-axis, $(1-t_i.prob)$, is defined as the cumulative probability that uncertain object t has distance to q greater than $t_i.dist$ (formally, $1 - t_i.prob = 1 - Pr\{dist(q, t) \leq t_i.dist\}$). Note that, there are two types of probabilities, appearance probability $t_i.p$ of instance t_i and cumulative probability $t_i.prob$. In the distance-and-probability space, we consider the latter $t_i.prob$, since larger $t_i.prob$ can indicate higher (cumulative) probability that object t is within $t_i.dist$ distance from q, compared with other objects. Moreover, since our problem aims to obtain objects with small $t_i.dist$ and large $t_i.prob$, for the sake of consistency and following the "the smaller, the better" convention of *skyline* computation [5] mentioned later, we convert the second coordinate of t_i into $(1 - t_i.prob)$ instead of $t_i.prob$.

Let us take Fig. 1 as an example. We can convert each instance in Fig. 1 into a 2D point in a distance-and-probability space, as shown in Fig. 2. Specifically, instance v_1 is mapped to a 2D point $\langle 1, 4/5\rangle$, since $v_1.dist = dist(q, v_1) = 1$ and $1-v_1.prob = 1-v_1.p = 4/5$. Similarly, instance v_2 is transformed to a point $\langle 5, 0\rangle$, since $v_2.dist = dist(q, v_2) = 5$ and $1 - v_2.prob = 1 - (v_1.p + v_2.p) = 0$ (i.e., the probability that $dist(q, v)$ is greater than 5 equals to zero). The conversion of other instances are similar.

In Fig. 2, we say an instance t_i is important, if its distance, $t_i.dist$, and/or probability, $(1 - t_i.prob)$, attributes are smaller than others. As a simple example, instance v_2 is clearly better than x_1, since both distance and probability

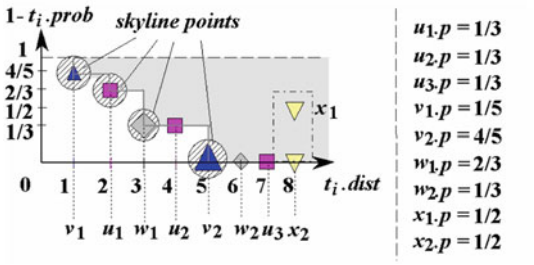

Fig. 2 Illustration of star instances and objects

attributes of v_2 are smaller than those of x_1 (i.e., v_2 is closer to q than x_1, and v_1 has higher confidence than x_1 to be close to q). Following the terminology of *skyline* [5], we say that point v_2 *dominates*[1] point x_1. Moreover, since v_2 is not dominated by other points, v_2 is called a *skyline point* in this 2D space (other skylines are v_1, u_1, and w_1). Note that, these skyline points correspond to instances that are more important than others, in terms of both distance and probability. Therefore, we call them *star instances* in our problem below.

Definition 1 (*Star instance*) Given a query point q and an uncertain database $\mathcal{D}$, let $t_i.dist$ and $(1 - t_i.prob)$ be the two attributes of an instance $t_i \in \mathcal{D}$ in the converted 2D space, where $t_i.dist = dist(q, t_i)$ and $t_i.prob = \sum_{\forall t_j \in t \wedge dist(q,t_j) \leq dist(q,t_i)} t_j.p$. Then, an instance t_i is a *star instance*, iff it is a skyline point in the converted 2D space (i.e., its 2D attribute vector is not *dominated* by others).

In the example of Fig. 2, since instances v_1, u_1, w_1, and v_2 are skyline points in the converted space, they are thus called star instances.

Intuitively, for any star instance $t_i \in t$, if we draw a circle centered at q with radius $t_i.dist$, then uncertain object t would have the highest probability to reside in this circle, compared with other objects in the database. Thus, we say that the corresponding object t is also important, as defined below.

Definition 2 (*Star object*) An uncertain object t is a *star object*, iff object t has at least one star instance.

Based on Definition 2, we can identify whether or not an object t is a star object by verifying its individual instances. In particular, if there exists one star instance t_i in t (closer to q and/or with higher confidence to be near to q than others), then t itself is a star object. In the example of Fig. 2, since v_1 and v_2 are star instances, object v is thus a star object. Similarly, since uncertain object u (w) has a star instance u_1 (w_1), it is also a star object. In contrast, x is not a star object, since none of its instances are star instances.

[1] We say that a point $x(x[1], x[2])$ *dominates* another point $y(y[1], y[2])$, if it holds that: (1) $x[i] \leq y[i]$ for *all* $1 \leq i \leq 2$, and (2) there exists at least one dimension $j \in \{1, 2\}$ such that $x[j] < y[j]$.

Given a query point, there might exist many star objects in the database (e.g., varying from a few to a hundred). In this case, users may only care about "the most important objects" with a controllable size (e.g., k). Motivated by this, we provide a natural ranking on these star objects. In particular, in the converted 2D space, we can determine the rank of a star object t by score:

$$score(t) = \sum_{\forall s_j \in \mathcal{D} \wedge (\exists t_i, t_i \prec_q s_j)} s_j.p, \quad (1)$$

where s_j is any instance in $\mathcal{D}$, and $t_i \prec_q s_j$ means 2D attribute vector of instance t_i dominates that of s_j. That is, the score, $score(t)$, of a star object t is the expected number of objects that are inferior to (i.e., dominated by) some instance in t.

The score, $score(t)$, for an uncertain object t is given by Eq. (1). Its semantic meaning is the confidence weight that other uncertain objects are inferior to t, that is, having both larger distances to query point q and lower probabilities in the 2D converted distance-probability space. This score is actually counting the expected number of objects that can be dominated by t in the best case (i.e., having both larger distance to q and lower residing probability within the circle mentioned above than some instance of object t). In other words, we always sum up existence probabilities of those instances dominated by t, and use it as a measure to rank object t. As long as the score is larger, it indicates higher confidence weight (i.e., the object is "better" than more other objects).

Naturally, if an object t is better than (i.e., dominates) a large number of objects in the database, then it would have high score (or rank). In Fig. 2, we have $score(u) = x_1.p + x_2.p = 1$, since x_1 and x_2 are both dominated by u_3. Similarly, we can obtain $score(v) = 4/3$, $score(w) = 5/3$, and $score(x) = 0$.

We now formally define the static PTkS query.

Definition 3 (*Static PTkS query*) Given a static query point q and an uncertain database $\mathcal{D}$, a *static PTkS query* retrieves k star objects $t \in \mathcal{D}$ that have the highest scores $score(t)$ as given in Eq. (1).

The semantics of our static PTkS query in Definition 3 can be viewed in two aspects. First, we require that the static PTkS answers should be star objects, that is, uncertain objects having instances being skylines in the 2D distance-probability space. Intuitively, the semantics of a star object t is that there exists at least one instance t_i of uncertain object t, such that object t has higher probability, $t_i.prob$, to appear in a circle centered at query point q and with a radius $dist(q, t_i)$, than other objects. Second, PTkS obtains those important objects, where the importance of objects is imposed by the query semantics, that is, some instance of objects should not be dominated by others in the 2D distance-probability space.

Thus, those objects that are close to star objects but are dominated by them will not be considered as PTkS answers since they are inferior to star objects. Third, in case the number of such star objects is more than k, we should rank them according to their scores defined w.r.t. their distances to q and object existence probabilities. Here, the score is defined as the summation of existence probabilities for those instances dominated by (instances of) object t in the 2D converted space. This score indicates the confidence weight that other uncertain objects are inferior to object t, that is, having larger distances to q and lower probabilities. Thus, by ranking star objects with their scores, we can obtain k star objects with high confidence weights.

In the example of Fig. 2, a PT2S query ($k = 2$) would return star objects v and w with scores 5/3 (higher than u and x). Note that, in a special case where multiple objects have the same score as the kth largest score, we will return all these objects as our answers (since we cannot distinguish them simply via scores).

In addition to the static PTkS query, we also consider the dynamic case where query object q moves from source q_A to destination q_B along a line segment q_Aq_B, and occasionally asks for its PTkS answers (i.e., with its location as query point at the query time). In this case, our problem is to efficiently find all possible PTkS candidate answers on the path of q from q_A to q_B, and thus reduce the search space to obtain the exact PTkS answer for a particular point $q \in q_Aq_B$. As mentioned in Sect. 1.3, this problem is useful for moving users (e.g., mobile users or troops) to find nearest and potentially important uncertain targets. The dynamic PTkS query with moving query point can be formalized below.

Definition 4 (*Dynamic PTkS query*) Denote the answer set of a PTkS query with static query point q as PTkS$(q, \mathcal{D})$. Given an uncertain database $\mathcal{D}$ and a line segment q_Aq_B along which query object q moves, a *dynamic PTkS query* retrieves star objects t such that $t \in \bigcup_{\forall q \in q_Aq_B}PTkS(q, \mathcal{D})$.

The semantics of dynamic PTkS query in Definition 3 is similar to that of static PTkS query in Definition 4, which obtains all the star objects with respect to a moving query point q on a line segment q_Aq_B. Table 2 summarizes the commonly-used symbols in this article.

Table 2 Meanings of notations

Symbols	Descriptions
$\mathcal{D}$	The uncertain database of size N
d	The dimensionality of the data set
q, q_Aq_B	The user-specified static/dynamic query point
t, s, u, v, w, x	The uncertain object
$\|t\|$	The number of instances for uncertain object t
t_i	The instance of an uncertain object t
$t_i.p$	The appearance probability of instance t_i
$dist(\cdot, \cdot)$	The *Euclidean distance* between two points

```
Procedure PTkS_Framework {
  Input: a d-dimensional uncertain database D, a query point q,
         and an integer k
  Output: the PTkS query result
  (1) construct a multidimensional index I over D
                                          // indexing phase
  (2) perform the pruning through index I and obtain a candidate
      set S_cand                          // pruning phase
  (3) refine candidates in S_cand and return the answer set
                                          // refinement phase
}
```

Fig. 3 The general framework for PTkS queries

3 Static PTkS query

Clearly, one straightforward way to answer PTkS queries is based on its definition. That is, for instances of *each* uncertain object, we sequentially scan the entire database $\mathcal{D}$, and check whether or not this uncertain object contains any star instance (defined in Definition 1). Once a star instance is found, its corresponding object will be the star object (as given in Definition 2). Then, among all the star objects, we compute their ranking scores (as given in Eq. (1)) and find out k objects with the highest scores as query answers. This method, however, incurs quadratic I/O and CPU processing costs, with respect to the number of instances in the database, which is obviously not efficient. Thus, instead of the sequential scan, in this article, we explore fast PTkS query processing with the help of spatial indexes.

Figure 3 illustrates a general framework for our PTkS query processing, which consists of three phases, *indexing*, *pruning*, and *refinement*. In the first phase (line 1), we insert uncertain objects into a multidimensional index, on which PTkS queries can be processed. In the second pruning phase (line 2), we traverse the index and reduce the search space of the PTkS query. After this phase, a PTkS candidate set S_{cand} is returned. Finally, in the refinement phase (line 3), for each candidate in S_{cand}, we refine it by computing its score given in Eq. (1), and return k star objects with the highest scores. Note that, this framework is applicable not only to the static PTkS query, but also to the dynamic case with moving query point, which will be discussed in Sect. 4.

3.1 Indexing uncertain data

Without loss of generality, in this article, we index uncertain objects using one of the most popular indexes, R-tree [17] and its variants (R*-tree [2]). Specifically, an R-tree index recursively groups spatially close points /regions with a

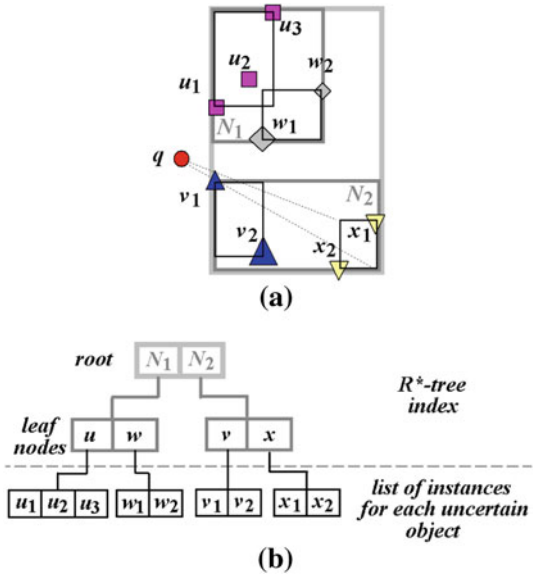

Fig. 4 R-tree construction over uncertain database

minimum bounding rectangle (MBR) until finally one node (i.e., root) is obtained.

Figure 4 illustrates an example of R-tree construction over uncertain objects. In particular, for each uncertain object (e.g., u) in the database, we group all its instances (e.g., u_1 and u_2) with an MBR. Then, we insert each MBR into an R-tree index (using standard insertion operator of R-tree). Each entry in the leaf node of R-tree represents an uncertain object t and contains a pointer pointing to a list of instances $t_i \in t$ (as well as their appearance probabilities $t_i.p$). These instances from uncertain objects in the same leaf nodes are stored sequentially on disk pages. Note that, the list of instances may span across a few, say B_t, disk pages, which correspond to B_t groups $G_1(t)$, $G_2(t)$, ..., and $G_{B_t}(t)$, respectively. For the sake of clear illustration, in the sequel, we first assume no extra data structures except lists are used to store instances, and later we will consider optimization techniques via pre-computation to achieve good query efficiency. Further, we use R-tree to index uncertain data on the object level, rather than the instance level. This is because by directly indexing uncertain objects in the leaf nodes, we can enable the pruning in intermediate nodes of the R-tree, by utilizing minimum/maximum distances from nodes to query point. In contrast, if we index instances in the R-tree, instances of an uncertain object may be scattered in different leaf nodes of the R-tree. As a result, we cannot compute distance bounds, that is, the minimum/maximum distance bounds from any uncertain object in the node to a query point q, simply derived from either non-leaf or leaf nodes to facilitate the pruning, before we completely visit *all* the instances of an object.

3.2 Pruning heuristics

In this subsection, we illustrate the rationale of reducing the search space of PTkS queries. As mentioned in Sect. 2, in order to answer the PTkS query, we need to equivalently obtain skyline points in a converted distance-probability space. However, in contrast to traditional skyline computation [5] where attributes are static, in our PTkS problem, both distance, $t_i.dist$, and probability, $(1 - t_i.prob)$, attributes of each instance t_i are dynamically computed with respect to query point q. Therefore, it is quite inefficient, in terms of both CPU time and I/O cost, to online materialize attributes of instances before the skyline computation. Observing this, in the sequel, we will only perform the *conceptual space conversion*, that is, we do not explicitly transform all instances in the database to 2D points. Instead, we still process objects/instances through the index in the original d-dimensional space, however, we answer PTkS queries in a way that implicitly executes the skyline computation in the 2D converted space.

Specifically, from Definition 1, since those star instances are skyline points (i.e., not dominated by other instances) in the converted space, we can thus safely discard instances that are dominated by others. As in the previous example of Fig. 2, instances x_1 and x_2 can be safely pruned, since they are dominated by instance v_2 (or w_2) and they are not star instances. We have the pruning lemma below.

Lemma 1 (Pruning Rule 1) *Instance t_i can be safely pruned, iff there exists an instance s_j such that $s_j \prec_q t_i$, that is,* (1) $s_j.dist \leq t_i.dist$, (2) $1 - s_j.prob \leq 1 - t_i.prob$, *and* (3) *two equalities in* (1) *and* (2) *do not hold at the same time.*

Since we use a spatial index like R-tree to facilitate query processing, we also need to design rules for pruning an intermediate node in the R-tree.

As illustrated in Fig. 4a, uncertain object x corresponds to a rectangle bounding two instances x_1 and x_2. Therefore, we can compute the minimum and maximum distances from query point q to (rectangle) x, denoted as $mindist(q, x)$ and $maxdist(q, x)$, respectively. This way, we can also obtain a rectangle with *distance*-axis within $[mindist(q, x), maxdist(q, x)]$ in the 2D converted space (see dotted rectangle in Fig. 2). In this case, even without knowing the exact interval of rectangle along the *probability*-axis, we can conclude that x does not contain any star instances, since any point within the rectangle is dominated by point v_2.

From this observation, we have the node pruning lemma:

Lemma 2 (Pruning Rule 2) *Let q be a query point, e be an MBR node or an uncertain object (or instance), and t be an uncertain object that we have seen. Without loss of generality, assume that instances $t_1, t_2, \ldots,$ and $t_{|t|}$ of t*

satisfy the condition that $dist(q, t_1) \leq dist(q, t_2) \leq \cdots \leq dist(q, t_{|t|})$. *Then,* e *can be safely pruned, if it holds that* $dist(q, t_{|t|}) < mindist(q, e)$, *where* $mindist(q, e)$ *is the minimum distance from* q *to MBR/object/instance* e.

Proof For instance $t_{|t|}$, since it has the farthest distance to q among all instances of t, we have $t_{|t|}.prob = 1$. Thus, the converted 2D point of instance $t_{|t|}$ is given by $\langle dist(q, t_{|t|}), 0\rangle$. Furthermore, from the lemma assumption, $dist(q, t_{|t|}) < mindist(q, e)$, for any instance $s_j \in e$, we have $dist(q, t_{|t|}) < mindist(q, e) \leq dist(q, s_j)$. That is, both conditions $dist(q, t_{|t|}) < dist(q, s_j)$ and $1 - t_{|t|}.prob = 0 \leq 1 - s_j.prob$ hold. From Lemma 1, it holds that any instance s_j in e can be safely pruned (i.e., e can be pruned). □

3.3 Identifying k star objects

In this subsection, we discuss how to efficiently obtain star instances/objects (i.e., skyline points in 2D converted space) by accessing each instance at most once. In the sequel, we first focus on identifying star instances while we are sweeping along the distance-axis in the 2D converted space. Then, we consider computing scores for uncertain objects in order to obtain k star objects.

3.3.1 Identification of star objects

Recall from Definition 2, that any star object must have at least one star instance. Therefore, in order to identify star objects, we have to obtain star instances first. Our basic idea is to scan the instances in ascending order of their distances to q (i.e., along the distance-axis in the 2D converted space; this can be achieved by accessing the R-tree index, which will be described later in Sect. 3.4), and meanwhile identify star instances by checking the probability-axis.

Specifically, for each candidate object t, we maintain a variable $t.prob$, which is initially set to 0. In particular, variable $t.prob$ stores the *cumulative appearance probability* (CAP) for instances of t that we have seen so far. Below, we illustrate our rationale of identifying star instances using the example in Fig. 2.

Example 1 As shown in Fig. 2, we access instances in the order of distances to q, that is, $v_1, u_1, w_1, u_2, v_2, \ldots$, and so on. When the first instance v_1 is encountered, we increase its CAP, $v.prob$, by $v_1.p$. That is, $v.prob = v.prob + v_1.p = 1/5$. Since $v.prob$ now becomes the highest CAP among all CAPs of candidates (others are 0), no other instances can dominate v_1 (as $1 - v.prob$ is the lowest). Thus, v_1 is a star instance, and v is a star object.

Similarly, when we access the next instance u_1, we increase $u.prob$ by $u_1.p$. That is, $u.prob = u.prob + u_1.p = 1/3$. Since $u.prob$ is greater than $v.prob$ (i.e., $1/3 > 1/5$) as well as CAPs (i.e., 0) of other candidates, u_1 is another star instance and u is a star object. For the third instance w_1 we encounter, $w.prob$ is updated to $2/3$ which becomes the highest CAP. Thus, w_1 is a star instance and w is a star object.

The fourth instance we visit is u_2. We increase $u.prob$ by $u_2.p$. That is, $u.prob = u.prob + u_2.p = 2/3$, which is equal to $w.prob$. However, this time u_2 is not considered as a star instance, since its distance to q is greater than w_1.

Finally, we access instance v_2, and increase $v.prob$ by $v_2.p$. That is, $v.prob = u.prob + v_2.p = 1$. Since no other candidates can have CAPs greater than 1, instance v_2 is clearly a star instance. Based on Lemma 2, any instance having distance to q greater than $dist(q, v_2)$ can be safely pruned. Therefore, we terminate the procedure of identifying star instances/objects.

```
Procedure Identify_Star_Obj {
  Input: instances t_i in ascending order of dist(q, t_i)
  Output: set Star_Obj_Set containing star objects
  // initialization
  (1)  Star_Obj_Set = φ; max_CAP = −∞;
       // variable max_CAP is the highest CAP for objects
       // that we have seen
  (2)  let all candidates t have initial values t.prob = 0
  (3)  while next instance t_i is not Nil and max_CAP ≠ 1
                                   // in a streaming fashion
  (4)    set t.prob = t.prob + t_i.p
  (5)    if t.prob > max_CAP      // t_i is star instance
  (6)      Star_Obj_Set = Star_Obj_Set ∪ {t}
  (7)      max_CAP = t.prob
  (8)  return Star_Obj_Set
}
```

Fig. 5 Procedure of identifying star objects

Figure 5 illustrates the details of identifying star instances/objects by scanning instances for one pass. The parameter max_CAP is a threshold we maintain to decide whether or not an instance t_i is a star instance on the probability-axis (line 5).

3.3.2 Calculating scores of candidates

As given in Definition 3, in order to obtain PTkS query result, we need to calculate scores for star objects. That is, for star object t, its score, $score(t)$, is given by summing up appearance probabilities of instances that are dominated by t's instances in the 2D space. Since directly computing the score is costly (i.e., almost scanning the entire database), in the sequel, we give a more efficient approach, which only involves objects we have seen so far and computes the scores in a streaming fashion.

One interesting observation is that, the probability summation of all instances in the 2D space is fixed (i.e., $\sum_{\forall t \in \mathcal{D}} \sum_{\forall t_i \in t} t_i.p = N$, where N is the total number of uncertain objects in the database). Based on this fact, calculating score $score(t)$ is equivalent to the one of computing

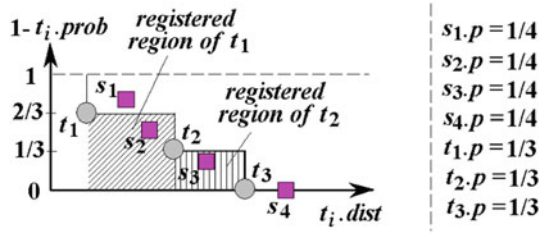

Fig. 6 Illustration of calculating $score'(t)$

$score'(t) = N - score(t)$, where $score'(t)$ can be obtained by summing up appearance probabilities of instances that are *not* dominated by t's instances.

Thus, by using $score'(t)$, our problem is reduced to the one that obtains k star objects with the smallest $score'(t)$. In other words, when computing $score'(t)$, we do not have to access those instances that are dominated by instances of object t (e.g., above or to the top-right of t's registered regions), and only need to care about instances in registered regions of t. One advantage of such reduction is that we can easily obtain $score'(t)$ by visiting each instance once in a streaming fashion.

Example 2 Figure 6 illustrates an example of calculating $score'(t)$, where each instance t_i (s_j) of object t (s) has appearance probability 1/3 (1/4). In particular, when we access an instance, for example t_1 in the figure, we record a probability threshold $1 - t_1.prob = 2/3$. The region filled with sloped lines is called *registered region of* t_1, within $[dist(q, t_1), dist(q, t_2)] \times [0, 2/3]$. Note that, any instance s_i falling into this region needs to update $score'(t)$ (initially 0) by adding $s_i.p$. In the example, $score'(t) = s_2.p = 1/4$.

The registered region of t_1 is canceled when t_2 is visited. In this case, t_2 registers a new region within $[dist(q, t_2), dist(q, t_3)] \times [0, 1/3]$, as shown in the figure (filled with vertical lines). Similarly, any instance falling into this region should update $score'(t)$ with its appearance probability.

Finally, the registered region of t_2 is canceled when instance t_3 is encountered. This way, we can obtain $score'(t)$ for object t.

In summary, whenever an instance t_i is encountered, we need to create/update its own registered region as well as updating other registered regions. Figure 7 illustrates procedure Calculate_PTkS_Obj in detail, which calculates scores of candidates meanwhile identifying star objects. Note that, after the "while" loop (lines 3–12′) in Fig. 7, we obtain all PTkS candidates (star object) t in the set $Star\_Obj\_Set$, however, their $score'(t)$ values might not involve all instances t_i of t. Thus, in order to compute the actual $score'(t)$ values, we need to further access the remaining instances via registered regions (line 13′). Finally, k candidates in $Star\_Obj\_Set$ with the smallest $score'(\cdot)$ are returned as PTkS answers (line 14′).

Procedure Calculate_PTkS_Obj {
Input: instances t_i in ascending order of $dist(q, t_i)$
Output: set $Star\_Obj\_Set$ containing k star objects with the highest scores
Identical to Procedure Identify_Star_Obj except: line 8 is replaced with:
(8') if t_i is not first encountered instance of t
(9') cancel the previously registered region
(10') register a region within $[dist(q, t_i), *] \times [0, 1 - t.prob]$
// $*$ is the farthest distance we have seen so far
(11') if t_i falls into any registered region of an instance $s_j \notin t$
(12') $score'(s_j) = score'(s_j) + t_i.p$
(13')compute the actual $score'(t)$ for candidates t in $Star\_Obj\_Set$
(14')return k objects in $Star\_Obj\_Set$ with the smallest $score'(\cdot)$
}

Fig. 7 Procedure of calculating PTkS answers

3.4 Query processing over the index

In this subsection, we discuss the PTkS query processing over R-tree index in detail. We will illustrate how to obtain an ordered sequence of instances (in ascending order of their distances to query point q) by traversing the spatial R-tree index. In addition, R-tree index can also facilitate reducing the search space of PTkS queries (i.e., we can prune some nodes such that objects under these nodes are not necessary to access, as given in Lemma 2).

3.4.1 Facilitating data structures

Figure 8 illustrates the pseudo code of PTkS query procedure, namely Static_PTkS_Processing, which traverses the R-tree index in a *best-first* manner and meanwhile computes k nearest star objects. Specifically, the query procedure maintains two *minimum heaps* $\mathcal{H}$ and $\mathcal{G}$ (both are initially empty), which contain entries in the form (e, key) (line 1). In particular, heap $\mathcal{H}$ contains entries where e corresponds to uncertain object or MBR node, and key is defined as the minimum distances from q to e. In contrast, heap $\mathcal{G}$ stores entries with instances (rather than objects/nodes in $\mathcal{H}$) and key is defined as the distance from q to instance e. In brief, heap $\mathcal{H}$ is used to traverse the R-tree index, whereas heap $\mathcal{G}$ serves like a "buffer" such that instances are output from $\mathcal{G}$ in ascending order of their distances to q.

In addition to the two heaps, we also keep an initially empty set $Star\_Obj\_Set$ storing star objects, and a threshold min_UB_dist (with an initial value $+\infty$) indicating the smallest maximum distance from q to uncertain objects that we have seen so far (line 2). Note that, threshold min_UB_dist can be used to perform the pruning (by Lemma 2).

```
Procedure Static_PTkS_Processing {
  Input: R-tree I constructed over D, a query point q,
         and integer k
  Output: the PTkS query result
  (1)  initialize min-heaps H and G accepting entries in the
         form (e, key)
  (2)  Star_Obj_Set = φ; min_UB_dist = +∞;
  (3)  insert (root(I), 0) into heap H
  (4)  while H is not empty
  (5)    (e, key_H) = de-heap H
  (6)    while G is not empty and key_H > top(G).key
  (7)      (t_i, key_G) = de-heap G
  (8)      access t_i by invoking lines 4-7 and 8'-12' of procedure
                 Calculate_PTkS_Obj      // Figure 7
  (9)      if max_CAP = 1 in procedure Calculate_PTkS_Obj
  (10)       terminate the loop of line 4;
  (11)   if e is an uncertain object // rename e to t
         // if t can be pruned as in Section 3.4.3, then continue;
  (12)     for each instance t_i in t
  (13)       if dist(q, t_i) ≤ min_UB_dist // Lemma 2
  (14)         insert (t_i, dist(q, t_i)) into heap G
  (15)   if e is a leaf node
  (16)     for each uncertain object t in e
  (17)       if mindist(q, t) ≤ min_UB_dist // Lemma 2
  (18)         if maxdist(q, t) < min_UB_dist
  (19)           min_UB_dist = maxdist(q, t)      // update
  (20)         insert (t, mindist(q, t)) into heap H
  (21)   if e is a non-leaf node
  (22)     for each entry e_i in e
  (23)       if mindist(q, e_i) ≤ min_UB_dist // Lemma 2
  (24)         insert (e_i, mindist(q, e_i)) into heap H
  (25) compute the actual score'(.) for candidates in Star_Obj_Set
  (26) return k objects in Star_Obj_Set with the smallest score'(·)
}
```

Fig. 8 Static PTkS query processing

3.4.2 Query procedure

The basic idea of our query procedure is to retrieve instances in ascending order of their distances to query object q by traversing the R-tree index, and meanwhile compute star instances (objects) in a streaming fashion by invoking procedure Calculate_PTkS_Obj (as mentioned in Sect. 3.3.2).

We traverse the R-tree index by first inserting the root $root(\mathcal{I})$ into heap $\mathcal{H}$ (line 3). Then, we want to access a node/object/instance in either $\mathcal{H}$ or $\mathcal{G}$ that may potentially contain/be instance with the smallest distance to q among all instances we have not seen so far. In particular, each time we pop out the top entry (e, key_H) from heap $\mathcal{H}$ (line 5). Note that, key_H of e is the smallest key (i.e., minimum possible distance to q) in $\mathcal{H}$. Moreover, let key_G (or $top(\mathcal{G}).key$) be the smallest key in heap $\mathcal{G}$. If it holds that $key_G < key_H$, then there must exist some instance t_i in heap $\mathcal{G}$ with the smallest distance to q among all instances that we have not seen. In this case, we de-heap entry (t_i, key_G) from $\mathcal{G}$, and invoke lines 4–7 and 8′–12′ of procedure Calculate_PTkS_Obj to process instance t_i, that is, identifying whether or not t_i is a star instance and updating scores of objects in set $Star\_Obj\_Set$. In case $max\_CAP = 1$ indicating that all PTkS candidates have been seen, we terminate the while loop of line 4 (lines 6–10).

The loop of lines 6–10 aborts when $key_H < top(\mathcal{G}).key$ holds, which shows that entry e with key key_H may contain instances with the smallest distances to q among the unseen instances. Thus, we need to access the children of entry e (lines 11–24). There are three possible types of entry e, uncertain object, leaf node, or non-leaf node, whose children correspond to instances, uncertain objects, and nodes, respectively. For each child of e, we check whether or not it can be safely pruned by Lemma 2 (lines 13, 17, and 23). If we cannot prune a child, then we need to add it to heap $\mathcal{G}$ or $\mathcal{H}$ for instance and object/node types, respectively (lines 14, 20, and 24).

The query procedure terminates when either heap $\mathcal{H}$ is empty (line 4) or we have seen all PTkS candidates (line 10). After the tree traversal, we can obtain a number of star objects, which were inserted into set $Star\_Obj\_Set$ during the call of procedure Calculate_PTkS_Obj (i.e., inserted when the first instance of an object is encountered; line 8). Moreover, each star object in $Star\_Obj\_Set$ is associated with a score, $score'(\cdot)$, which is also updated when invoking procedure Calculate_PTkS_Obj. Therefore, as mentioned in Sect. 3.3.2, we first obtain the actual $score'(\cdot)$ for star objects in set $Star\_Obj\_Set$ by accessing their unseen instances (in pruned nodes/objects), and report k objects with the smallest $score'(\cdot)$ as PTkS answers (lines 25–26).

Discussions on time and space complexities. As illustrated in lines 13, 17, and 23 of query procedure Static_PTkS_Processing, in the filtering phase (lines 1–24), those instances/objects/nodes can be effectively pruned by Lemma 2, if their minimum distances to q are greater than the threshold min_UB_dist, where min_UB_dist is the smallest maximum distance from q to uncertain object we have seen so far. Thus, in the filtering phase, we only need to access objects with instance distances to q smaller than min_UB_dist, which also form a superset of star objects in $Star\_Obj\_Set$. Note that, since we visit objects in ascending order of keys (i.e., minimum distances to q) in heap $\mathcal{H}$, we can quickly shrink threshold min_UB_dist and achieve high pruning power, as given in lines 18–19. Thus, the number, $|Star\_Obj\_Set|$, of star objects in $Star\_Obj\_Set$ is much smaller than the database size N in practice.

Furthermore, in procedure Calculate_PTkS_Obj of Fig. 7, each star object in the set $Star\_Obj\_Set$ corresponds to one registered region (note: in lines 8′–9′, we cancel the previous region for an object t if it exists). Thus, the total number of registered regions at any time is bounded by $|Star\_Obj\_Set|$, and the space complexity of set $Star\_Obj\_Set$ is $O(|Star\_Obj\_Set|)$. Moreover, let $Ins\_Num\_Visited$ be the total number of instances that we need to visit to identify star objects and compute their scores, that is, the count of instances that we need to access during both filtering and refinement phases (lines 1–25). As our procedure visits instances in the registered regions, and computes $score'(t)$ for star objects, the time complexity of our query procedure is given by $O(Ins\_Num\_Visited \cdot |Star\_Obj\_Set|)$ in the worst case.

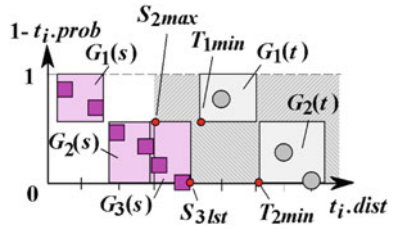

Fig. 9 Illustration of pruning with groups

From our experimental results later in Sect. 5.1 (Fig. 16), we will see that the number of final candidates is small, and the number of visited instances (close to query point) is thus also small. As discussed above, since $|Star\_Obj\_Set| \ll N$, the time and space costs of maintaining set $Star\_Obj\_Set$ are thus low on average.

3.4.3 Optimization

As given in line 13 of procedure Static_PTkS_Processing (in Fig. 8), when we encounter an uncertain object t in heap $\mathcal{H}$, we need to access all its instances t_i, and compute distances $dist(q, t_i)$ from t_i to query point q. Since the number of instances per uncertain object can be rather large (e.g., a few hundred), it is thus inefficient to compute the distance $dist(q, t_i)$ for every instance t_i. In order to reduce the computation cost, we propose a *pivot-based* approach below, which partitions instances of an uncertain object into groups, and selects pivots in groups to reduce the cost of distance computation.

Specifically, as mentioned earlier in Sect. 3, when the number of instances in object t is large, it may go across B_t pages (groups) $G_1(t), G_2(t), \ldots$, and $G_{|t|}(t)$. We illustrate the basic idea of our optimization, that is, pruning with groups using an example in Fig. 9. Assume we have seen an uncertain object s, which can be divided into three groups $G_1(s), G_2(s)$, and $G_3(s)$. As shown in the figure, each group $G_i(s)$ of instances is bounded by a rectangle in the 2D converted space. Now we encounter a new object t with two groups $G_1(t)$ and $G_2(t)$ (bounded by rectangles as well). Let S_{2max} denote the top-right corner of rectangle of $G_2(s)$, T_{1min} (T_{2min}) be the bottom-left corner of $G_1(t)$ ($G_2(t)$), and S_{3lst} be the bottom-right corner of group $G_3(s)$. Obviously, since S_{2max} and S_{3lst} dominate T_{1min} and T_{2min}, respectively, we can conclude that t is not a star object (since all its instances are dominated by others). Note that, such object t can be safely pruned according to the definition of star object. Thus, our optimization is to prune earlier with groups before line 12 of procedure Static_PTkS_Processing.

Next, we discuss how to efficiently obtain the rectangles bounding groups in the 2D converted space. In brief, we present a pivot-based approach, named after adopting the idea of *triangle inequality* via *pivots* to reduce the computation cost. In particular, let piv_i be a representative instance (pivot) in a group $G_i(t)$. If we have offline pre-computed the distances $dist(piv_i, o_i)$ from pivot piv_i to any instance $o_i \in G_i(t)$, then by the triangle inequality, we have:

$$|dist(q, piv_i) - dist(piv_i, o_i)| \le dist(q, o_i) \le dist(q, piv_i) + dist(piv_i, o_i). \quad (2)$$

Let $maxdist(piv_i, G_i(t))$ be the maximum distance from pivot piv_i to any instance in group $G_i(t)$. Therefore, we can bound the distance $dist(q, G_i(t))$ from query point q to any group $G_i(t)$ by $[max\{0, dist(q, piv_i) - maxdist(piv_i, G_i(t))\}, dist(q, piv_i) + maxdist(piv_i, G_i(t))]$, denoted as interval $[LB\_dist_i, UB\_dist_i]$. Once we have obtained the distance interval for each group, we can compute the probabilistic interval $[LB\_prob_i, UB\_prob_i]$ for instances in each group G_i. In particular, we have $UB\_prob_i = \sum_{\forall G_j(t), LB\_dist_j \le UB\_dist_i} G_j(t).agg$; similarly, we have $LB\_prob_i = \sum_{\forall G_j(t), UB\_dist_j \le LB\_dist_i} G_j(t).agg$, where $G_j(t).agg$ is the summed appearance probability for all instances in group $G_j(t)$.

Cost model for grouping. One remaining issue to be addressed is how to select the pivot piv_i in each group $G_i(t)$. Intuitively, we want to select pivots such that the bounding rectangles of groups are as tight as possible, which can achieve high pruning power.

In addition, from the example of Fig. 9, we find that, when distance bounds of $G_1(s)$ do not intersect with those of other groups (e.g., $G_2(s)$ or $G_3(s)$), its probability bounds are simply [0, 1/3] (probability bounds of its instances); in contrast, group $G_3(s)$ intersects with $G_2(s)$ on distance bounds and we have to underestimate its probability lower bound by decreasing 1/3 due to the intersection. Thus, this larger probability bound interval may decrease the pruning power. Based on observations above, in order to obtain tight probability bounds, it is highly desired that distance bounds of groups are disjoint. Thus, we propose a cost model to formalize the disjoint quality of distance bounds, which can guide the selection of pivots.

Specifically, as given in Eq. (2), distance from q to any instance o_i is bounded by an interval. We model the quality of distance bounds, $P_{disjoint}$, by the summed probability that any two pair of instances o_i and o_j have their intervals intersecting with each other. That is,

$$\begin{aligned} P_{disjoint} &= \sum_i \sum_{j \ne i} Pr\{dist(q, piv_i) + dist(piv_i, o_i) \\ &\le |dist(q, piv_j) - dist(piv_j, o_j)|\} \\ &= \sum_i \sum_{j \ne i} Pr\{(dist(q, piv_i) + dist(piv_i, o_i) \\ &\le dist(q, piv_j) - dist(piv_j, o_j)) \end{aligned}$$

$$\bigcup (-(dist(q, piv_i) + dist(piv_i, o_i)) \geq dist(q, piv_j) - dist(piv_j, o_j))\} \quad (3)$$

Since it holds that $Pr\{M_1 \cup M_2\} = Pr\{M_1\} + Pr\{M_2\} - Pr\{M_1 \cap M_2\}$, we can rewrite Eq. (3) as:

$$\begin{aligned} P_{disjoint} = &\sum_i \sum_{j \neq i} Pr\{(dist(q, piv_i) - dist(q, piv_j) \\ &+ dist(piv_i, o_i) + dist(piv_j, o_j)) \leq 0\} \\ &+ \sum_i \sum_{j \neq i} Pr\{(dist(q, piv_i) + dist(q, piv_j) \\ &+ dist(piv_i, o_i) - dist(piv_j, o_j)) \leq 0\} \end{aligned} \quad (4)$$

In the sequel, we only present how to quickly obtain the first term in Eq. (4) and the method of computing the second term is similar. In particular, we consider $(dist(q, piv_i) - dist(q, piv_j))$ as a variable X, $dist(piv_i, o_i)$ as a variable Y, and $dist(piv_j, o_j)$ as variable Z. Since $i \leq j$ holds, variables X and Y are obtained from two different groups, which can be considered as independent of each other. Moreover, X and Y (Z) is also independent, since the distance difference between q and pivots is unrelated to the distance from pivot to other instance. Thus, we can apply the *Central Limit Theorem* (CLT) to the first term of Eq. (4). Note that, although CLT assumes a summation of large number of random variables, there are some studies [16,19] indicating that 3 variables can also achieve a good approximation of the probability.

Assume means of X, Y, and Z are μ_X, μ_Y, and μ_Z, respectively; variance of X, Y, and Z are σ_X^2, σ_Y^2, and σ_Z^2, respectively. For simplicity, let variable W be $X + Y + Z (= dist(q, piv_i) - dist(q, piv_j) + dist(piv_i, o_i) + dist(piv_j, o_j))$. Thus, the first term in Eq. (4) can be simplified as $Pr\{W \leq 0\}$, which, by applying CLT, can be further rewritten as:

$$\begin{aligned} Pr\{W \leq 0\} &= Pr\left\{\frac{W - (\mu_X + \mu_Y + \mu_Z)}{\sqrt{\sigma_X^2 + \sigma_Y^2 + \sigma_Z^2}} \leq \frac{-(\mu_X + \mu_Y + \mu_Z)}{\sqrt{\sigma_X^2 + \sigma_Y^2 + \sigma_Z^2}}\right\} \\ &= \Phi\left(\frac{-(\mu_X + \mu_Y + \mu_Z)}{\sqrt{\sigma_X^2 + \sigma_Y^2 + \sigma_Z^2}}\right) \end{aligned} \quad (5)$$

where $\Phi(\cdot)$ is the *cumulative density function* (cdf) of a *standard normal distribution*, and $(\mu_X + \mu_Y + \mu_Z)$ and $(\sigma_X^2 + \sigma_Y^2 + \sigma_Z^2)$ are the mean and variance of W, respectively. The case of the second term in Eq. (4) is similar.

Thus, we can compute the probability $P_{disjoint}$ in Eq. (4) based on statistics of variables (e.g., X). The selection of pivots is somewhat like *k-means algorithm* [30] which iteratively swaps pivot with a non-pivot. The only differences are that, the cost formula is replaced with $P_{disjoint}$ in Eq. (4), and we aim to obtain a pivot set that maximizes $P_{disjoint}$.

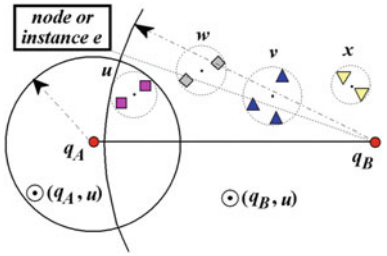

Fig. 10 A PTkS with moving query point

4 Dynamic PTkS query

In this section, we further investigate the dynamic PTkS query, where query point q moves along a line segment $q_A q_B$. As given in Definition 4, our goal is to find a union of PTkS answer sets for every possible position of $q \in q_A q_B$.

4.1 Pruning heuristics

Figure 10 illustrates an example of dynamic PTkS query, where we have instances of uncertain objects u, v, w, x, and an MBR node e. Without loss of generality, we use a sphere to bound instances of each uncertain object. Clearly, when query point q resides at point q_A, uncertain object u has the smallest maximum distance (i.e., $maxdist(q_A, u)$) to q among all objects. We draw a circle, $\odot(q_A, u)$, centered at q_A with a radius $maxdist(q_A, u)$. From Pruning Rule 2 (Lemma 2), we can see that any MBR node/instance e that completely falls outside this circle can be safely pruned (since it holds that $maxdist(q_A, u) < mindist(q_A, e)$).

Similarly, we can draw another circle, $\odot(q_B, u)$, centered at q_B with radius $maxdist(q_B, u)$, as depicted in the figure. From Lemma 2, it also holds that any MBR node/instance e that is completely outside circle, can be safely pruned (since it holds that $maxdist(q_B, u) < mindist(q_B, e)$).

Next, we will prove in the following lemma that, for *any* query point q on the line segment $q_A q_B$, those nodes/instances completely falling outside $\odot(q_A, u) \cup \odot(q_B, u)$ would not be the answer to dynamic PTkS query.

Lemma 3 (Pruning Rule 3) *For any uncertain object $u \in \mathcal{D}$ and a query line segment $q_A q_B$, we draw two circles $\odot(q_A, u)$ and $\odot(q_B, u)$ centered at q_A and q_B, with radii $maxdist(q_A, u)$ and $maxdist(q_B, u)$, respectively. Then, any node/instance e (or t_i) completely falling outside these two circles can be safely pruned (in other words, e or t_i is not PTkS answer with any query point $q \in q_A q_B$).*

Proof Please refer to Appendix I. □

Lemma 3 indicates that only those objects/nodes that intersect with $\odot(q_A, u)$ or $\odot(q_B, u)$ are candidates for dynamic PTkS query.

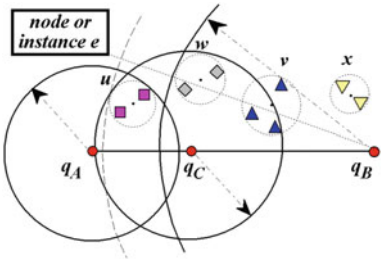

Fig. 11 Reducing size of PTkS candidate set

From the example of Fig. 10, we find that circle $\odot(q_B, u)$ is very large, which may include many false alarms. Motivated by this, we propose a method to reduce the number of candidates. In particular, since object w is fully contained in the circle $\odot(q_B, u)$, instead, we can use a smaller circle $\odot(q_B, w)$ centered at q_B with radius $maxdist(q_B, w)$, as illustrated in Fig. 11. Clearly, if query point q resides at q_B, any objects/MBRs that completely fall outside circle $\odot(q_B, w)$ can be safely pruned. However, in order to guarantee no false dismissals for any query point on line segment q_Aq_B, we need to draw a third circle, denoted as $\odot(q_C, u, w)$, which is centered at some point q_C on q_Aq_B and tangent to both object u and w.

This way, we can reduce the number of candidates by including a new circle. The following corollary can guarantee that, by doing this, we will not introduce any false dismissals.

Corollary 1 *Let $\odot(q_C, u, w)$ be a circle centered at $q_C \in q_Aq_B$ with radius $maxdist(q_C, u)(= maxdist(q_C, w))$, and circles $\odot(q_A, u)$ and $\odot(q_B, w)$ refer to Lemma 3. Then, any node /instance e (or t_i) completely falling outside circles $\odot(q_A, u)$, $\odot(q_C, u, w)$, and $\odot(q_B, w)$ can be safely pruned.*

Proof Please refer to Appendix II. □

Therefore, in a general case, as long as there exists an uncertain object (e.g., w in the example above) that is fully contained in a circle we have obtained so far (e.g., $\odot(q_B, u)$), we can reduce this circle by drawing a smaller circle via this object (e.g., $\odot(q_B, w)$). Moreover, in order to guarantee no false dismissals, we may need to add one additional circle (e.g., $\odot(q_C, u, w)$). This process continues until there are no objects fully contained in the obtained circles.

4.2 Dynamic query processing

In this subsection, we present the procedure of answering dynamic PTkS query over the R-tree index. Specifically, given a line segment q_Aq_B, the basic idea of our query procedure is to retrieve all the possible PTkS candidates when query point q moves along q_Aq_B. Then, given a particular

```
Procedure Dynamic_PTkS_Processing {
  Input: R-tree I constructed over D, a query line segment qAqB,
         a set of query
         points q ⊆ qAqB, and integer k
  Output: dynamic PTkS candidates
  (1)  initialize min-heap H accepting entries in the form (e, key)
  (2)  Scand = ϕ; max_UB_dist = +∞;
  (3)  insert (root(I), 0) into heap H
  (4)  while H is not empty
  (5)    (e, keyH) = de-heap H
  (6)    if keyH > max_UB_dist, terminate the loop; // Lemma 3
  (7)    if e is a leaf node
  (8)      for each object t in e (sorted on maxdist(t, qAqB))
  (9)        if t is fully contained in some circle in Scand
  (10)         update the splitting points and circles in Scand
  (11)         set max_UB_dist to the maximum radius of circles
                 in Scand
  (12)       if t intersects with some circle in Scand
  (13)         add t to the candidate list of this circle
  (14)   else      // intermediate node
  (15)     for each entry ei in e
  (16)       if mindist(qAqB, ei) ≤ max_UB_dist // Lemma 3
  (17)         insert (ei, mindist(qAqB, ei)) into heap H
  // for any query point q ∈ qAqB
  (18) obtain two candidate lists of circles ⊙(qX, ·) and ⊙(qY, ·),
         for q ∈ qXqY ⊆ qAqB
  (19) Identical to Procedure Static_PTkS_Processing except: line 3
         is replaced with:
  (3') insert entry (t, mindist(q, t)) into heap H, for all
         t ∈ (⊙(qX, ·) ∪ ⊙(qY, ·))
}
```

Fig. 12 Dynamic PTkS query processing

user-specified query point on q_Aq_B, the exact PTkS answer can be obtained by applying the static PTkS techniques (discussed in Sect. 3) over the retrieved candidate set of a much smaller size (compared with total database size). Clearly, this method is much more efficient than issuing static PTkS queries on the index from scratch every time a query point is specified, in terms of both I/O and CPU costs.

Figure 12 illustrates the detailed procedure of dynamic query processing. In particular, we maintain a minimum heap $\mathcal{H}$ containing entries (e, key), where e is tree node and key is defined as the minimum distance from a line segment q_Aq_B to e [3] (line 1). Intuitively, the closest nodes to q_Aq_B are more likely to be the dynamic query results. In addition, we keep a candidate set S_{cand} storing circles and their candidates (objects intersecting with circles), and a variable max_UB_dist recording the radius of the largest circle in S_{cand} (line 2).

Similar to the static case, we traverse the R-tree by first inserting the root $root(\mathcal{I})$ into heap $\mathcal{H}$ (line 3). Then, each time we pop out an entry with the minimum key (lines 4–5). The loop terminates when the key is already greater than variable max_UB_dist, indicating no nodes in the heap would contain qualified PTkS query answers via Lemma 3 (line 6).

To traverse the R-tree, we verify whether or not the children of node e with either object or node type need to be visited. For a child t of e with object type, if it is fully contained in some circle in S_{cand}, then we can update S_{cand} as mentioned in Sect. 4.1, as well as the maximum radius max_UB_dist of circles in S_{cand}

(lines 7–11). If t intersects with some circles, then it is a candidate and we add it to the candidate list of this circle (line 12). For a child e_i of e with node type, we apply the pruning rule to e_i, as mentioned in Lemma 3. If e_i cannot be pruned, then we need to insert e_i into heap $\mathcal{H}$ with key $mindist(q_Aq_B, e_i)$ for further refinement (lines 14–17).

Finally, set S_{cand} includes all possible PTkS candidates when q moves from q_A to q_B. When a specific query point q on q_Aq_B is given, we can conduct the refinement over a smaller candidate set, instead of querying over R-tree index from scratch. Specifically, assume q is located on line segment $q_Xq_Y \subseteq q_Aq_B$, where q_X and q_Y are two consecutive splitting points on q_Aq_B. We can obtain candidate lists of two circles $\odot(q_X, \cdot)$ and $\odot(q_Y, \cdot)$ (line 18), and invoke the static PTkS query procedure Static_PTkS_Processing in Figure 8 (line 19). The only difference from the static case is that, line 3 is replaced with line 3′, that is, we insert uncertain objects t in the two candidate lists $\odot(q_X, \cdot)$ and $\odot(q_Y, \cdot)$ into heap $\mathcal{H}$, in the form $(t, mindist(q, t))$. This way, the retrieval of PTkS answers for a query point $q \in q_Aq_B$ can be efficiently conducted on a small data set, which incurs much lower cost than query processing on the index from scratch.

5 Experimental evaluation

In this section, we demonstrate the efficiency and effectiveness of both static and dynamic *probabilistic top-k star* (PTkS) query processing. We conduct the experiments on both real and synthetic data sets. Specifically, we use 2D geographical data sets, LB and RR, which contain bounding rectangles (MBRs) of 53,145 Long Beach county roads and 128,971 Tiger/Line LA rivers and railways, respectively. Here, each MBR can be considered as an uncertainty region of an object's location. These two data sets are available at url: [http://www.rtreeportal.org/]. Furthermore, we also use *GPS* data set, which contains 12 days' trajectories (with 2 dimensions latitude and longitude) between home and office of people. The uncertainties in GPS data result from many reasons such as clock errors, ephemeris errors, atmospheric delays, and multipathing and satellite geometry. In order to describe such GPS uncertainties, we take 10–50 positions at consecutive timestamps in trajectories as independently and identically drawn random samples to represent the location distribution of an uncertain object. For synthetic data sets, we produce each uncertain object t as follows. First, we randomly select a center location C_t for object t in a d-dimensional space $\mathcal{U} = [0, 1000]^d$. Then, we decide a random radius $r_t \in [r_{min}, r_{max}]$ such that instances of t fall into a circle (denoted as $UR(t)$) centered at C_t and with a radius r_t. Finally, we randomly generate instances t_i of object t within $UR(t)$, as well as their appearance probabilities. For brevity, we denote data sets with *location* C_t of *Uniform* (*Skew*, skewness = 0.8) distribution as lU (lS); similarly, denote data sets with *radius* r_t of *Uniform* (*Gaussian* with mean $\frac{r_{min}+r_{max}}{2}$ and variance $\frac{r_{max}-r_{min}}{2}$) distribution as rU (rG). Thus, we can obtain four types of data sets, $lUrU$, $lUrG$, $lSrU$, and $lSrG$, which are indexed by R-tree [2] as mentioned in Sect. 3.1 where page size is set to $1K$ [41] (note: for 2D, 3D, 4D, and 5D data, each page contains 50, 36, 28 and 23 entries, respectively; the results of other page sizes are similar). For all the real/synthetic data sets, we produce instances of uncertain object t uniformly distributed in $UR(t)$ and their appearance probabilities $t_i.p$ following *Gaussian* distribution (we assign probability in $(0, 1]$ with mean 0.5 and variance 0.2 to each instance and normalize them such that $\sum_{i=1}^{|t|} t_i.p = 1$). Note that, for other data sets (e.g., with a different skewness for C_t, different mean and variance of r_t, or distributions of instances or probability $t_i.p$), the experimental results are similar. Thus, we only present the results over data sets mentioned above.

In order to evaluate PTkS query processing, we randomly generate query points q in the data space $\mathcal{U}$. Moreover, we test the query performance using two measures, *wall clock time* and *speed-up ratio* compared with *linear scan*. In particular, the wall clock time consists of two parts, CPU time and I/O cost, during query processing over the index, where we incorporate the cost of each page access (i.e., I/O) by penalizing $10ms$, similar to [40,41]. Moreover, the speed-up ratio is defined as the wall clock time of the linear scan method divided by that of our approaches. To our best knowledge, no previous work studies the PTkS problem in the uncertain database. Therefore, we compare our approach with the only available method, (lightweight) linear scan (note: the nested-loop one mentioned in Sect. 3 is more costly), which utilizes Pruning Rule 2. That is, we sequentially access instances of each uncertain object, obtain all instances (candidates) that have their distances to q smaller than or equal to the minimum $maxdist(q, t)$ for some object t that we have seen, and finally refine the candidates by finding k actual star objects.

Without loss of generality, in subsequent experiments, we generate 10–50 instances for each uncertain object, which are divided into 2–5 groups. The trends with different numbers of instances or groups are similar, and we do not present all of them. Table 3 summarizes the parameter settings we tested. In particular, the values in bold font are *default values*. Each time we vary value of one parameter while fixing other parameters to their default values.

In the sequel, Sects. 5.1 and 5.2 illustrate the query performance of static and dynamic PTkS queries, respectively. All our experiments are conducted on a Pentium IV 3.2 GHz PC with 1G memory. The reported results are the average of 100 queries.

Table 3 The parameter settings

Parameter	Values
$[r_{min}, r_{max}]$	[0, 5], [0, 10], **[0, 20]**, [0, 30], [0, 50]
k	3, 5, **10**, 15, 20
d	2, **3**, 4, 5
N	200 K, 300 K, **500 K**, 800 K, 1 M
$\|q_A q_B\|$	5, **10**, 15, 20

5.1 Performance of static PTkS query

In the first set of experiments, we verify the correctness of our proposed cost model for grouping, as mentioned in Sect. 3.4. Specifically, we vary the number of groups per uncertain object from 2 to 5, and compare the estimated measure $P_{disjoint}$ in Eq. (4) with the actual value. Figure 13a shows the experimental results with data set $lUrU$. From figure, we can see that although there are about 10–20% relative errors between the actual and estimated $P_{disjoint}$, the trend of the estimated $P_{disjoint}$ for different group numbers remains similar to that of the actual one. The same phenomenon occurs with other real/synthetic data or parameters which are omitted. Therefore, the trend of estimated $P_{disjoint}$ can reveal that of the actual one, which is useful for our pivot selection. This indicates that we can use our cost model (i.e., estimated one) to iteratively evaluate the quality of the selected pivots (as mentioned in Sect. 3.4.3), and obtain groups with the actual $P_{disjoint}$ as large as possible.

As a second step, Fig. 13b evaluates our PTkS query processing approach with and without optimizations (as mentioned in Sect. 3.4.3) over four types of synthetic data sets, where the number of groups per uncertain object is set to 2, and other parameters are set to default values. Recall that, our optimization techniques utilize pivots (selected based on cost model) to reduce the computation cost. Thus, the wall clock time of PTkS with optimizations is smaller than that of PTkS without optimizations (by randomly selecting pivots), which shows the effectiveness of our proposed optimization techniques based on the cost model. In the subsequent experiments, we will always show the query results by applying the optimization techniques.

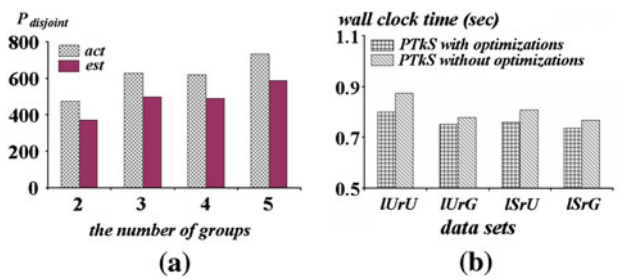

Fig. 13 Model verifications. **a** Actual versus estimated $P_{disjoint}$, $lUrU$, **b** PTkS with and without optimizations

Table 4 PTkS query performance on real data sets

Real data sets	Wall clock time (s)	Speed-up ratio
LB	0.25261	495.014
RR	0.101975	2,452.49
GPS	1.18	105.941

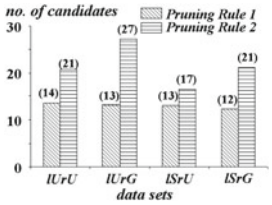

Fig. 14 Effect of pruning rules versus data sets. **a** $lUrU$ and $lUrG$, **b** $lSrU$ and $lSrG$

Next, Table 4 illustrates the results of static PTkS on real data sets LB, RR, and GPS, where the required wall clock time is small (i.e., $\leq$ 1.18 s) and the speed-up ratio is by 2–3 orders of magnitude, compared with linear scan. Note that, the time cost for GPS data is higher than that of LB and RR. This is because GPS data are obtained from trajectories which have many overlaps of objects' uncertainty regions, and it thus takes more time to prune false alarms. Nevertheless, our PTkS approach can still achieve high speed-up ratio by about 2 orders of magnitude.

For synthetic data sets, before we report the efficiency of static PTkS query processing with different parameters below (e.g., radius range $[r_{min}, r_{max}]$, parameter k, dimensionality d, and data size N), we first show in Fig. 14 the pruning effects of Pruning Rules 1 and 2 mentioned in Lemmas 1 and 2, respectively, in terms of the number of remaining candidates, on synthetic data sets, where parameters are set to default values. Specifically, we can see that for each data set, compared with the total data size $N = 500$ K, the number of candidates after pruning rules is small (i.e., 10–30). Pruning Rule 2 is applied on both node and object levels, and the remaining objects are candidates for star objects; Pruning Rule 1 has higher pruning power than Pruning Rule 2, since it can obtain the actual star objects. In our PTkS query procedure, we use Pruning Rule 2 to quickly filter out false alarms on the node level, and then the remaining candidates can be efficiently filtered by Pruning Rule 1.

Figure 15 presents the performance of our static PTkS query over four types of synthetic data sets, by varying the radius range from [0, 5] to [0, 50]. In figures, the numbers over columns represent the speed-up ratios of our approach, compared with the linear scan. Large radii of uncertain objects indicate instances of objects scatter in a large region.

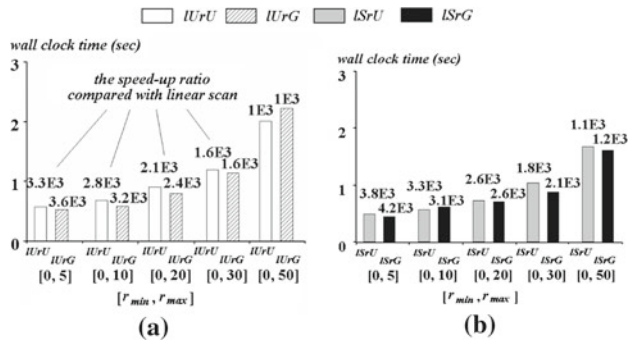

Fig. 15 Static PTkS versus radius range $[r_{min}, r_{max}]$. **a** $lUrU$ and $lUrG$, **b** $lSrU$ and $lSrG$

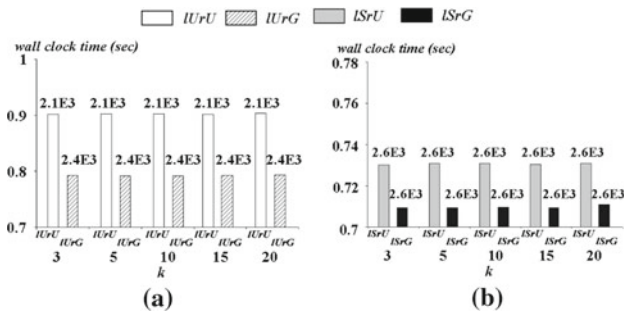

Fig. 17 Static PTkS versus parameter k. **a** $lUrU$ and $lUrG$, **b** $lSrU$ and $lSrG$

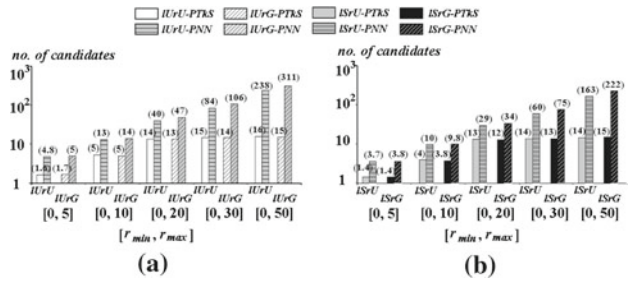

Fig. 16 Comparison between static PTkS and PNN (No. of Candidates versus $[r_{min}, r_{max}]$). **a** $lUrU$ and $lUrG$, **b** $lSrU$ and $lSrG$

Thus, more candidates are expected to become star objects, and the retrieval of them from the index requires more efforts, in terms of wall clock time, as confirmed in figures. Moreover, the speed-up ratios are about 3 orders of magnitude, which shows good efficiency of our method.

We also compare the PNN query [10,11,23] mentioned in Sect. 1 with our PTkS query. In particular, in order to avoid false dismissals, we set the probabilistic threshold T_p of PNN to 0 (i.e., retrieving uncertain objects with nonzero probabilities to be NN), and show the number of PNN candidates returned. Moreover, we report the number of star objects after the index filtering in our PTkS query. We test four types of synthetic data sets in Fig. 16, by varying radius range $[r_{min}, r_{max}]$ from [0, 5] to [0, 50], where other parameters are set to their default values. From the experimental results, we can see that the number of PNN candidates increases with wider radius range (since more objects are possible to be NN of query point), and in the worst case, this number can be as large as 311. Note that, too many candidates may be meaningless for users to choose. Moreover, the refinement of these candidates by computing the PNN probability integration (i.e., using numerical methods [10]) will incur high computation cost. Thus, it is not suitable for applications (requiring fast response) like coal mine surveillance. In contrast, our PTkS queries result in much fewer candidates, that is, star objects. Note that, the refinement does not involve the costly integration like PNN and can be efficiently accomplished in a streaming fashion (as shown in Fig. 15). Finally, only those important star objects with highest ranks are returned as the query answers. Thus, our proposed PTkS is more efficient and suitable than PNN for applications that require small response time (e.g., coal mine surveillance). The results with real data sets are similar and thus omitted. The effectiveness comparison of PTkS with PNN will be later discussed in Sect. 6.

Figure 17 illustrates the experimental results with different values of k. From figures, we find that the wall clock time of our approach remains approximately the same when k is varied. The reason is that, the identification (traversing index) and ranking (in memory) of star objects are not sensitive to k value. The only difference is that we return k star objects with the highest ranks for different k values. The speed-up ratio of our approach remains high (i.e., 3 orders of magnitude).

Figure 18 studies the effect of dimensionality d on the wall clock time and speed-up ratio of static PTkS query processing. In particular, when the dimensionality increases from 2 to 5, the wall clock time of our approach first decreases and then increases. This is due to the effects of two aspects, data density and index efficiency. Since our data size N is fixed (i.e., 500 K), the density of data objects in the data space would become smaller with the increasing dimensionality, which leads to fewer PTkS candidates to retrieve and process. On the other hand, when the dimensionality is high (e.g., 5D), the query efficiency of R-tree index would degrade [42]. Thus, these two factors together determine the trend shown in figures. Similar to previous results, for all the tested dimensionality, our method has high speed-up ratio (i.e., 3 orders of magnitude) compared with linear scan.

Figure 19 presents the PTkS results of scalability test on data sizes. Specifically, the wall clock time of our approach increases smoothly when the data size N becomes larger. Furthermore, the speed-up ratio also increases with respect to the increasing data size, which indicates a good scalability on data size with our approach.

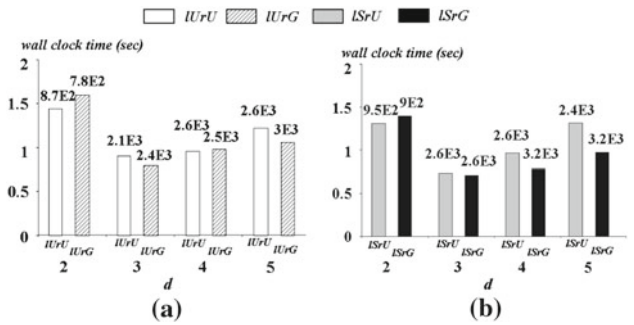

Fig. 18 Static PTkS versus dimensionality d. **a** $lUrU$ and $lUrG$, **b** $lSrU$ and $lSrG$

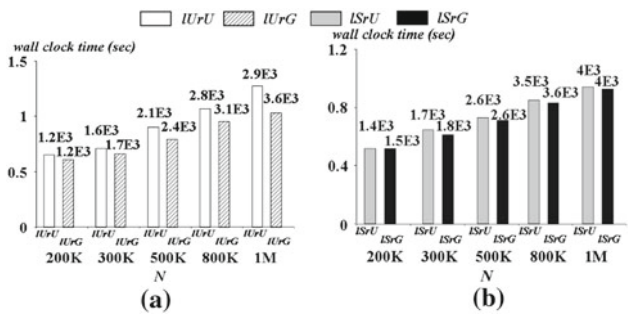

Fig. 19 Static PTkS versus data size N. **a** $lUrU$ and $lUrG$, **b** $lSrU$ and $lSrG$

5.2 Performance of dynamic PTkS query

In this subsection, we test the performance of dynamic PTkS query processing, where query point moves along a line segment q_Aq_B. In particular, we randomly generate line segments q_Aq_B in the data space with arbitrary directions. Assume that the query object issues 10 queries when it moves along line segment q_Aq_B. The experimental results with other numbers (e.g., 20) of queries are similar and omitted. We test the wall clock time of our approach, which first finds out PTkS candidates for all query points on q_Aq_B at a time and then searches PTkS answers among candidates for each of the 10 queries. In contrast, the linear scan method retrieves PTkS results from scratch for each query. In the subsequent experiments, we show the total wall clock time and speed-up ratio (compared with linear scan) for 10 queries.

Figures 20, 21, and 22 illustrate the experimental results on synthetic data by varying radius range $[r_{min}, r_{max}]$, dimensionality d, and data size N, respectively. The trends of wall clock time in figures are similar to those in the static case.

Figure 23 demonstrates the effect of query length $|q_Aq_B|$ on our dynamic PTkS query performance. In particular, since longer line segment would result in more candidates, the wall clock time of our approach increases with the increasing query length, as confirmed in figures. Furthermore, our approach is much better than linear scan by about 4 orders of

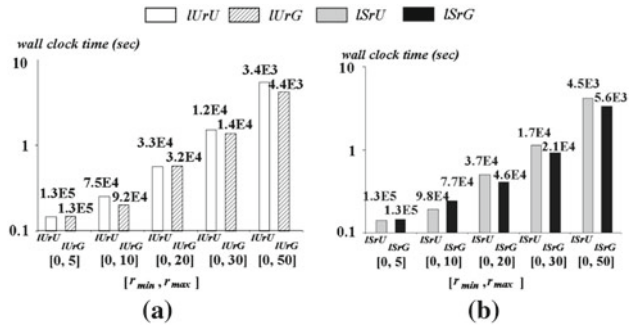

Fig. 20 Dynamic PTkS versus radius range $[r_{min}, r_{max}]$. **a** $lUrU$ and $lUrG$, **b** $lSrU$ and $lSrG$

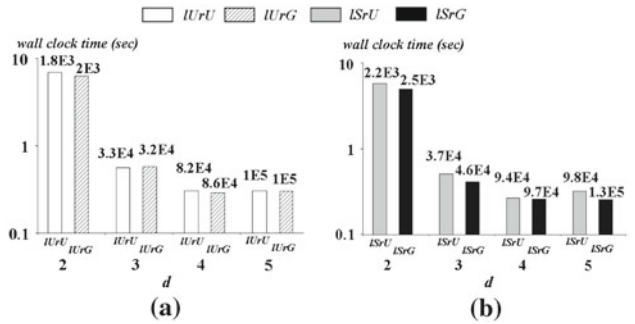

Fig. 21 Dynamic PTkS versus dimensionality d. **a** $lUrU$ and $lUrG$, **b** $lSrU$ and $lSrG$

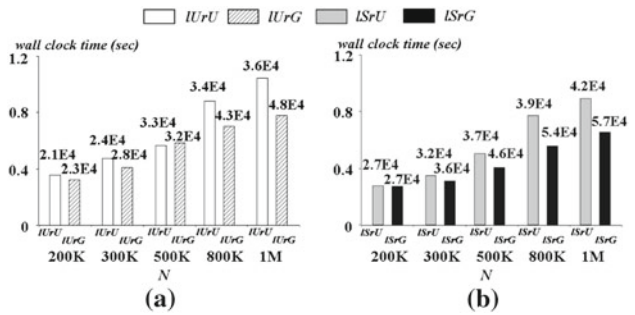

Fig. 22 Dynamic PTkS versus data size N. **a** $lUrU$ and $lUrG$, **b** $lSrU$ and $lSrG$

magnitude. The results on real data, LB and RR, are similar and omitted.

In summary, extensive experiments have verified the efficiency and effectiveness of our proposed methods for answering both static and dynamic PTkS query, in terms of wall clock time and speed-up ratio compared with the linear scan.

6 Related work

In real-world applications, uncertainty is either inherently contained in data [4,8,10,15,27] or intentionally injected [13,29]. Thus, it has recently become crucial to explore how to answer various queries over uncertain data *effectively* and

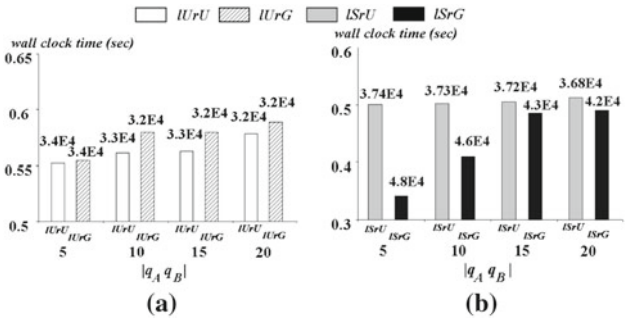

Fig. 23 Dynamic PTkS versus query length $|q_A q_B|$. **a** $lUrU$ and $lUrG$, **b** $lSrU$ and $lSrG$

efficiently. While the focus of our study is on uncertain databases, there are also some other studies on probabilistic databases [1,44,18,34,37] under *possible worlds* semantics.

Since traditional query processing methods usually assume precise data, they are not directly applicable to handling uncertain data. Therefore, many query types in the uncertain database have to be re-defined in order to obtain accurate results from uncertain data, including *range query* [6,11,12,39], *nearest neighbor query* [9–11,23], *skyline query* [33], *reverse skyline query* [24], *ranked query* [25], *top-k dominating query* [26], and *similarity join* [22,28].

Comparison between PTD [26] and PTkS. The top-k dominating (PTD) query [26] retrieves k uncertain objects that are expected to dynamically dominate the most numbers of other objects in the original data space. In contrast, our PTkS query obtains k star objects that are "close" to a query point considering both distance and probability aspects in a 2D converted (distance-probability) space (rather than the original space), where distance is defined as the Euclidean distance from data instance to query point, and probability is the (non-)existence probability of an instance.

Thus, PTD and PTkS have different semantics w.r.t. definitions, which can be used in different applications. For example, PTD is used in applications where attributes can be of various types (as dominance are considered on separate attributes) and users want to obtain the average results over all possible instance combinations; PTkS is applicable to those scenarios that have a meaningful distance function w.r.t. attributes (such as Euclidean distance in LBS applications), and return important star objects (e.g., those even with small existence probabilities). Note that, in some applications such as LBS where attributes of uncertain objects are coordinates of locations, PTD that considers the dominance relationship among objects' coordinates is not meaningful, and thus we should use PTkS in this case.

While the solutions to PTD reduce the problem to the one that condenses instance combinations and conduct the query by directly traversing the R-tree in the original data space, PTkS has to online compute star objects (or skylines) in the 2D converted space in a streaming fashion (by visiting instances of uncertain objects in the R-tree, in ascending order of their distances to query point). The pruning methods for PTD utilize the lower/upper score bounds of objects to enable the pruning. This is different from our PTkS query that accesses instances for one pass, and apply pruning rules based on the object dominance in 2D ad-hoc converted space. In addition, our PTkS query also considers the dynamic case where the query point moves along a line segment. Therefore, previous techniques proposed for PTD cannot be used in our PTkS problem.

The most related works to our PTkS problem are probabilistic nearest neighbor (PNN) query in the uncertain database [10,11,23], which only considers the *expected probability* (greater than or equal to a threshold $T_p > 0$) that an object is a nearest neighbor of query point q. However, in many critical applications like coal mine surveillance, considering the expected probability alone is not sufficient. This is because carelessly ignoring dangerous events with a bit smaller probabilities than threshold T_p may cause loss of lives.

Comparison between PNN [10,11,23] and PTkS. As an example in Fig. 1, assume that objects u, v, w, and x are uncertain data collected from different sensors in the coal mine surveillance application [43], and their instances represent the data distributions. We take object u as an example to illustrate how to calculate the expected PNN probability [10,11,23] that u is the nearest neighbor (NN) of q. We first consider instance u_1 of u. From figures, u_1 is the nearest neighbor of q only if object u resides at u_1, v at v_2, w at w_1 (or w_2), and x at x_1 (or x_2). Thus, the expected probability that u_1 is NN can be calculated by the formula shown in Table 5, that is, with the PNN probability $\frac{4}{15}$. Similarly, from Table 5, we have the probability that u_2 is NN equal to $\frac{4}{45}$, and that of u_3 is 0. Thus, the PNN probability, $P_{PNN}(u)$, of uncertain object u is given by a summation of probabilities of u_1, u_2, and u_3, that is,

$$P_{PNN}(u) = \frac{4}{15} + \frac{4}{45} + 0 = \frac{16}{45}$$

In the same way, from Table 5, PNN probabilities of v, w, and x can be calculated as $\frac{13}{45}$, $\frac{16}{45}$, and 0, respectively. Therefore, if users set threshold T_p in the PNN query to $\frac{16}{45}$, then only objects u and w are returned (i.e., reported dangerous events like fires). However, we can see from Fig. 1a, that we have missed a very important object v, which has the PNN probability $\frac{13}{45}$ a bit smaller than $\frac{16}{45}$, however, indeed has $1/5$ chance to be NN of q (indicating a site on fire) once it resides at v_1. Thus, this PNN definition would not report such a dangerous event. In contrast, our PTkS results will include object v.

On the other hand, in order to avoid missing important answers, one possible remedy is to assign a small PNN

threshold T_p, or in the extreme case let $T_p = 0$ [11]. However, the number of the resulting candidates might be too large. For the coal mine surveillance, the mine manager either needs to filter out false alarms manually, which may delay the precious time for evacuation, or asks workers to evacuate the mine whenever a potential danger is detected, which will waste millions of dollars for each false alarm. Thus, the PNN query is not suitable for such scenario. In contrast, our PTkS query computes the skylines in the distance-and-probability space, and can obtain important answers.

Given another example, Table 6 shows 3 uncertain objects a, b, and c, with existence probabilities of of their instances and distances to query point q. In this example, when $k = 2$, the PTkS answers are objects c (with score 1) and a (with score 2/3). In contrast, the PNN query will rank the top-2 objects as a (with probability 10/27) and b (with probability 6/27). However, we can see that object b has an instance b_2 which is the farthest from q (among all instances) and with high appearance probability 2/3, whereas object c has approximately the same (close) distances to q (i.e., 3 and 5) with high probabilities (i.e., 1/3 and 2/3, respectively). This indicates that object c in our PTkS results is more probable (at least not less probable) to be close to q than object b (within circle centered at q with radius greater than $3 = dist(q, c_1)$). The high ranking of object b in the PNN query is mainly due to the expectation of probabilities. Thus, our PTkS answers make more sense, compared with that of PNN.

Comparison between NN with the expected distance and PTkS. If we consider the expected distance from query object to uncertain objects and retrieve NN of query object q, we may still miss some important results. For example, in the example of Fig. 1, the expected distance from q to object u is given by:

$$2 \times \frac{1}{3} + 4 \times \frac{1}{3} + 7 \times \frac{1}{3} = 4.33;$$

Table 5 The NN probabilities of object instances for PNN

Instance	The NN probability
u_1	$u_1.p \cdot v_2.p \cdot \left(\sum_{i=1}^{2} w_i.p\right) \cdot \left(\sum_{i=1}^{2} x_i.p\right) = \frac{4}{15}$
u_2	$u_2.p \cdot v_2.p \cdot w_2.p \cdot \left(\sum_{i=1}^{2} x_i.p\right) = \frac{4}{15}$
u_3	$u_3.p \cdot 0 \cdot 0 \cdot \left(\sum_{i=1}^{2} x_i.p\right) = 0$
v_1	$v_1.p \cdot \left(\sum_{i=1}^{3} u_i.p\right) \cdot \left(\sum_{i=1}^{2} w_i.p\right) \cdot \left(\sum_{i=1}^{2} x_i.p\right) = \frac{1}{5}$
v_2	$v_2.p \cdot u_3.p \cdot w_2.p \cdot \left(\sum_{i=1}^{2} x_i.p\right) = \frac{4}{45}$
w_1	$w_1.p \cdot \left(\sum_{i=2}^{3} u_i.p\right) \cdot v_2.p \cdot \left(\sum_{i=1}^{2} x_i.p\right) = \frac{16}{45}$
w_2	$w_2.p \cdot u_3.p \cdot 0 \cdot \left(\sum_{i=1}^{2} x_i.p\right) = 0$
x_1	$x_1.p \cdot 0 \cdot 0 \cdot 0 = 0$
x_2	$x_2.p \cdot 0 \cdot 0 \cdot 0 = 0$

Table 6 An example for comparison between PNN and PTkS

Uncertain object t	Instances t_i	Distances $dist(q, t_i)$	Appearance probability $t_i.p$
a	a_1	1	1/3
	a_2	4	1/3
	a_3	6	1/3
b	b_1	2	1/3
	b_2	7	2/3
c	c_1	3	1/3
	c_2	5	2/3

similarly, the expected distances from q to v, w, and x are 4.2, 4 and 8, respectively. Thus, w is the returned result, whose instances are however the third and sixth closest to q. As a result, v is still missing in the NN result, which is important in applications like coal mine surveillance, and in contrast returned by PTkS.

Therefore, the work studied in this article proposes a novel point of view for queries in uncertain databases, that is, there exists a trade-off between query predicate measure (distance in our example) and probabilistic confidence, which is different from previous studies that only consider the expected measure or probabilistic confidence, and may miss some important results in critical applications like coal mine surveillance. For example, previous studies on NN (using the expected distance from query object to uncertain data) can indeed obtain the nearest neighbor of query object. However, we may neglect some important objects since we ignore the uncertainty of object positions. In contrast, our PTkS query not only considers the distance but also the probability with respect to this distance. To achieve this, we propose techniques specifically designed for PTkS such as one-pass approach to retrieve and rank star objects, which cannot be solved by traditional NN method.

Moreover, previous studies on PNN retrieve objects that have expected NN probabilities greater than a threshold. Thus, techniques are designed to obtain PNN candidates and refine them. In contrast, our PTkS queries have completely different semantics, which considers online computing the skyline in a converted distance-and-probability space. Therefore, specific to our PTkS problem, we propose our own query procedure, called Static_PTkS_Processing, that correctly outputs instances of uncertain objects in ascending order of their distances to query point q, which is not required by previous studies, via two minimum heaps $\mathcal{H}$ and $\mathcal{G}$. To retrieve star objects (or star instances), we designed an efficient one-pass approach that can achieve this goal in a streaming fashion. Furthermore, to compute scores and rank the star objects, we also integrate the solution of computing scores into the one-pass retrieval process of star objects.

In addition, we also provide optimization techniques to enhance the query performance via pivots, whose selection is guided by our specifically designed cost model. From the above comparisons of PTkS and previous studies with expected distance or probability, we can see that techniques used in previous studies cannot be directly applied in our PTkS problem.

Previous studies on skyline can be classified into two categories, *static* and *dynamic skylines*, in which attributes of objects are fixed and dynamic, respectively. Existing studies on static skyline processing include BNL [5], D&C [5], *bitmap* and *index* [38], NN [21], and BBS [31]. Moreover, the dynamic skyline has been studied in different scenarios, where dynamic attributes are defined as distances from objects to query points in the Euclidean space [31,35], road network [14], or generic metric space [7]. In contrast, our PTkS problem needs to obtain dynamic skyline, where not only distance but also probability attributes are dynamically calculated with respect to query point. Thus, existing techniques in dynamic skyline solely dealing with distance attributes (e.g., [7,14,31,35]) cannot be directly used in our problem involving probability attribute.

7 Conclusions

Query processing on uncertain data is very important in many applications due to the existence of uncertainty in real-world data. In this article, we study the problem of identifying uncertain objects that are "closest" to a query point. Specifically, we propose a novel concept, namely *star object*, which considers both distance and probability aspects with respect to the query point. We also define a *probabilistic top-k star* (PTkS) query aiming to retrieve k star objects from the database, and design effective pruning methods for PTkS query processing. Furthermore, we extend the proposed solution to the dynamic case where query point is moving toward a direction. Extensive experiments have verified the efficiency and effectiveness of our proposed approaches.

Acknowledgments Funding for this work was provided by Hong Kong RGC NSFC JOINT Grant under Project No. N_HKUST61 2/09 and NSFC Grant No. 60736013, 60803105, 60873022, and 60903053.

Appendix

I. Proof of Lemma 3

Proof As illustrated in Fig. 24, it is sufficient to consider point t_i that is outside circles $\odot(q_A, u)$ and $\odot(q_B, u)$ (the case of node e can be easily extended).

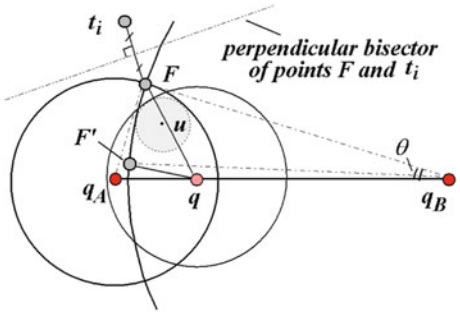

Fig. 24 Proof of Lemma 3

We prove the lemma by contradiction. We assume that there exists a position q on line segment q_Aq_B such that circle $\odot(q, u)$ centered at q with radius $maxdist(q, u)$ contains point t_i (i.e., assume t_i is a candidate).

Let point F be an intersecting point of two circles $\odot(q_A, u)$ and $\odot(q_B, u)$, and angle $\angle Fq_Bq$ be θ. In the sequel, we first prove that $dist(q, F)$ is the maximum distance from q to any point in the intersection of $\odot(q_A, u)$ and $\odot(q_B, u)$.

From the *law of cosines*, we have $dist^2(q, F) = dist^2(q_B, F) + dist^2(q_B, q) - 2 \cdot dist(q_B, F) \cdot dist(q_B, q) \cdot cos\theta$. Since $dist(q_B, F)$ and $dist(q_B, q)$ are constants, $dist(q, F)$ decreases with the decreasing θ value (note: $\theta \in [0, \pi]$). In other words, for point F' which is also on circle $\odot(q_B, u)$ like point F (i.e., $dist(q_B, F) = dist(q_B, F')$) and $\angle F'q_Bq \leq \theta$, we have $dist(q, F') \leq dist(q, F)$. That is, $dist(q, F)$ is the maximum distance from q to any point in $\odot(q_A, u) \cap \odot(q_B, u)$.

Since $\odot(q, u)$ intersects with u at one point and u is completely contained in $\odot(q_A, u) \cap \odot(q_B, u)$, we have $maxdist(q, u) \leq dist(q, F)$. According to our assumption that t_i is in $\odot(q, u)$, we have $dist(q, t_i) \leq maxdist(q, u) \leq dist(q, F)$. Thus, point q must be on the halfplane containing t_i, which is obtained by the perpendicular bisector of points F and t_i.

Furthermore, based on the lemma assumption, we have $dist(q_A, F) < dist(q_A, t_i)$ and $dist(q_B, F) < dist(q_B, t_i)$, which indicates q_A and q_B are on the halfplane containing F. Since q is on q_Aq_B, q is also in the same halfplane. However, this is contrary to the previous result that q is on the halfplane containing t_i. Hence, our initial assumption is not correct, and t_i can be safely pruned. □

II. Proof of Corollary 1

Proof We consider circles $\odot(q_A, u)$ and $\odot(q_C, u, w)$, as illustrated in Fig. 11. From Lemma 3, we know that no matter where query point q is located on line segment q_Aq_C, we can safely pruned nodes/points fully outside $\odot(q_A, u) \cup \odot(q_C, u, w)$. Similarly, we can prune nodes/points fully

outside $\odot(q_C, u, w) \cup \odot(q_B, w)$ when q is on line segment $q_C q_B$. Thus, for any query point on line segment $q_A q_B$ ($= q_A q_C \cup q_C q_B$), nodes/points can be safely pruned if they are located outside $\odot(q_A, u) \cup \odot(q_C, u, w) \cup \odot(q_B, w)$, which completes our proof. □

References

1. Antova, L., Jansen, T., Koch, C., Olteanu, D.: Fast and simple relational processing of uncertain data. In: Proceedings 24th International Conference on Data Engineering. (2008)
2. Beckmann, N., Kriegel, H., Schneider, R., Seeger, B.: The R*-tree: an efficient and robust access method for points and rectangles. In: Proceedings of the ACM SIGMOD International Conference on Management of Data. (1990)
3. Benetis, R., Jensen, C., Karciauskas, G., Saltenis, S.: Nearest neighbor and reverse nearest neighbor queries for moving objects. In: TimeCenter Technical Report. (2002)
4. Böhm, C., Pryakhin, A., Schubert, M.: The Gauss-tree: efficient object identification in databases of probabilistic feature vectors. In: Proceedings 22th International Conference on Data Engineering. (2006)
5. Borzsonyi, S., Kossmann, D., Stocker, K.: The skyline operator. In: Proceedings of the 17th International Conference on Data Engineering. (2001)
6. Chen, J., Cheng, R.: Efficient evaluation of imprecise location-dependent queries. In: Proceedings of the 23th International Conference on Data Engineering. (2007)
7. Chen, L., Lian, X.: Dynamic skyline queries in metric spaces. In: Proceedings of the International Conference on Extending Database Technology. (2008)
8. Chen, L., Ozsu, M.T., Oria, V.: Robust and fast similarity search for moving object trajectories. In: Proceedings of the ACM SIGMOD International Conference on Management of Data. (2005)
9. Cheng, R., Chen, J.: Probabilistic verifiers: evaluating constrained nearest-neighbor queries over uncertain data. In: Proceedings of the 24th International Conference on Data Engineering. (2008)
10. Cheng, R., Kalashnikov, D., Prabhakar, S.: Querying imprecise data in moving object environments. IEEE Trans. Knowl. Data Eng. **16**(9) (2004)
11. Cheng, R., Kalashnikov, D.V., Prabhakar, S.: Evaluating probabilistic queries over imprecise data. In: Proceedings of the ACM SIGMOD International Conference on Management of Data. (2003)
12. Cheng, R., Xia, Y., Prabhakar, S., Shah, R., Vitter, J.: Efficient indexing methods for probabilistic threshold queries over uncertain data. In: Proceedings of the 30th International Conference on Very Large Data Bases. (2004)
13. Cheng, R., Zhang, Y., Bertino, E., Prabhakar, S.: Preserving user location privacy in mobile data management infrastructures. In: Privacy Enhancing Technologies. (2006)
14. Deng, K., Zhou, X., Shen, H.T.: Multi-source skyline query processing in road networks. In: Proceedings of the 23th International Conference on Data Engineering. (2007)
15. Faradjian, A., Gehrke, J., Bonnet, P.: Gadt: a probability space ADT for representing and querying the physical world. In: Proceedings of the 18th International Conference on Data Engineering. (2002)
16. Grinstead, C.M., Snell, J.L.: Introduction to Probability. American Mathematical Society (AMS), USA (1997)
17. Guttman, A.: R-trees: a dynamic index structure for spatial searching. In: Proceedings of the ACM SIGMOD International Conference on Management of Data. (1984)
18. Hua, M., Pei, J., Zhang, W., Lin, X.: Efficiently answering probabilistic threshold top-k queries on uncertain data. In: Proceedings of the 24th International Conference on Data Engineering. (2008)
19. Jovanovic-Dolecek, G.: Demo program for central limit theorem. In: Circuits and Systems. (1997)
20. Kang, J.M., Mokbel, M.F., Shekhar, S., Xia, T., Zhang, D.: Continuous evaluation of monochromatic and bichromatic reverse nearest neighbors. In: Proceedings of the 23th International Conference on Data Engineering. (2007)
21. Kossmann, D., Ramsak, F., Rost, S.: Shooting stars in the sky: an online algorithm for skyline queries. In: Proceedings of the 28th International Conference on Very Large Data Bases. (2002)
22. Kriegel, H.-P., Kunath, P., Pfeifle, M., Renz, M.: Probabilistic similarity join on uncertain data. In: International Conference on Database Systems for Advanced Applications. (2006)
23. Kriegel, H.-P., Kunath, P., Renz, M.: Probabilistic nearest-neighbor query on uncertain objects. In: International Conference on Database Systems for Advanced Applications. (2007)
24. Lian, X., Chen, L.: Monochromatic and bichromatic reverse skyline search over uncertain databases. In: Proceedings of the ACM SIGMOD International Conference on Management of Data. (2008)
25. Lian, X., Chen, L.: Probabilistic ranked queries in uncertain databases. In: Proceedings of the International Conference on Extending Database Technology. (2008)
26. Lian, X., Chen, L.: Top-k dominating queries in uncertain databases. In: Proceedings of the International Conference on Extending Database Technology. (2009)
27. Ljosa, V., Singh, A.K.: APLA: indexing arbitrary probability distributions. In: Proceedings of the 23th International Conference on Data Engineering. (2007)
28. Ljosa, V., Singh, A.K.: Top-k spatial joins of probabilistic objects. In: Proceedings of the 24th International Conference on Data Engineering. (2008)
29. Mokbel, M.F., Chow, C.-Y., Aref, W.G.: The new casper: query processing for location services without compromising privacy. In: Proceedings of the 32nd International Conference on Very Large Data Bases. (2006)
30. Ng, R.T., Han, J.: Efficient and effective clustering methods for spatial data mining. In: Proceedings of the 20th International Conference on Very Large Data Bases. (1994)
31. Papadias, D., Tao, Y., Fu, G., Seeger, B.: Progressive skyline computation in database systems. ACM Trans. Database Syst. **30**(1), 41–82 (2005)
32. Papadimitriou, S., Li, F., Kollios, G., Yu, P.S.: Time series compressibility and privacy. In: Proceedings of the 33rd International Conference on Very Large Data Bases. (2007)
33. Pei, J., Jiang, B., Lin, X., Yuan, Y.: Probabilistic skylines on uncertain data. In: Proceedings of the 33rd International Conference on Very Large Data Bases. (2007)
34. Re, C., Dalvi, N., Suciu, D.: Efficient top-k query evaluation on probabilistic data. In: Proceedings of the 23th International Conference on Data Engineering. (2007)
35. Sharifzadeh, M., Shahabi, C.: The spatial skyline queries. In: Proceedings of the 32nd International Conference on Very Large Data Bases. (2006)
36. Singh, S., Mayfield, C., Shah, R., Prabhakar, S., Hambrusch, S., Neville, J., Cheng, R.: Database support for probabilistic attributes and tuples. In: Proceedings of the 24th International Conference on Data Engineering. (2008)
37. Soliman, M.A., Ilyas, I.F., Chang, K.C.: Top-k query processing in uncertain databases. In Proceedings of the 23th International Conference on Data Engineering. (2007)
38. Tan, K.-L., Eng, P.-K., Ooi, B.C.: Efficient progressive skyline computation. In: Proceedings of the 27th International Conference on Very Large Data Bases. (2001)

39. Tao, Y., Cheng, R., Xiao, X., Ngai, W.K., B.K., Prabhakar, S.: Indexing multi-dimensional uncertain data with arbitrary probability density functions. In: Proceedings of the 31st International Conference on Very Large Data Bases. (2005)
40. Tao, Y., Papadias, D., Lian, X.: Reverse *k*NN search in arbitrary dimensionality. In: Proceedings of the 30th International Conference on Very Large Data Bases. (2004)
41. Tao, Y., Papadias, D., Lian, X., Xiao, X.: Multidimensional reverse *k*NN search. In: The VLDB Journal (2005)
42. Theodoridis, Y., Sellis, T.: A model for the prediction of R-tree performance. In: Proceedings of the ACM SIGACT-SIGMOD Symposiam on Principles of Database Systems. (1996)
43. Xue, W., Luo, Q., Chen, L., Liu, Y.: Contour map matching for event detection in sensor networks. In: Proceedings of the ACM SIGMOD International Conference on Management of Data. (2006)
44. Yi, K., Li, F., Srivastava, D., Kollios, G.: Efficient processing of top-*k* queries in uncertain databases. In: Proceedings of the 24th International Conference on Data Engineering. (2008)