Top-k Dominating Queries in Uncertain Databases

Xiang Lian and Lei Chen
Department of Computer Science and Engineering
Hong Kong University of Science and Technology
Clear Water Bay, Kowloon
Hong Kong, China
{xlian, leichen}@cse.ust.hk

ABSTRACT

Due to the existence of uncertain data in a wide spectrum of real applications, uncertain query processing has become increasingly important, which dramatically differs from handling certain data in a traditional database. In this paper, we formulate and tackle an important query, namely *probabilistic top-k dominating* (PTD) query, in the uncertain database. In particular, a PTD query retrieves k uncertain objects that are expected to *dynamically dominate* the largest number of uncertain objects. We propose an effective pruning approach to reduce the PTD search space, and present an efficient query procedure to answer PTD queries. Furthermore, approximate PTD query processing and the case where the PTD query is issued from an uncertain query object are also discussed. Extensive experiments have demonstrated the efficiency and effectiveness of our proposed PTD query processing approaches.

1. INTRODUCTION

Uncertain data exist in many real-world applications such as sensor networks [10, 16], object identification [2], *location-based services* (LBS) [21], and moving object tracking [5, 4, 19]. Uncertain data objects are usually modeled as *uncertainty regions* [5, 29], in which objects can reside with any data distribution. In some applications, the *probabilistic density function* (pdf) of each object is known [6, 2]. In other cases, the pdf is practically not explicitly available [23]. Therefore, a number of instances are collected to mimic such pdf. For example, in sensor networks, sensory data collected at a specific timestamp often contain noises due to environmental factors or device failure. In this case, samples obtained within a short period around that timestamp can be used to represent the distribution of its possible values.

Table 1 depicts an example of uncertain database, which consists of 3 uncertain objects u, v, and w. Each uncertain object t may have one or more instances, t_i, with *appearance probability* $t_i.p \in [0, 1]$ (i.e. the probability that an instance appears at a position). For instance, object u has one instance u_1 residing at location $\langle 6, 2\rangle$ with probability $u_1.p = 1$. Similarly, object v has two instances v_1 and v_2, with probabilities $v_1.p = 0.3$ and $v_2.p = 0.7$, respectively. Furthermore, object w contains two instances w_1 and w_2 with appearance probabilities $w_1.p = 0.5$ and $w_2.p = 0.5$, respectively.

uncertain object t	**instance t_i**	**appearance probability $t_i.p$**
u	$u_1\langle 6, 2\rangle$	1
v	$v_1\langle 6, 4\rangle$	0.3
	$v_2\langle 4, 3\rangle$	0.7
w	$w_1\langle 3, 3\rangle$	0.5
	$w_2\langle 2, 2\rangle$	0.5

Table 1: An Example of Uncertain Database

Query processing over uncertain database has played an increasingly important role in applications like multi-criteria decision making, data cleansing, and so on. One important query type in the uncertain database is called *probabilistic ranked* (PRank) query [18], which retrieves uncertain objects that are expected to have the i-th rank with the highest probability, for $1 \leq i \leq k$, where k is a user-specified integer and the rank of each object is determined by score computed with a linear function. A clear advantage for the PRank query is that users can control the size of the PRank answer set through parameter k. On the other hand, however, it might not be convenient for users to specify an appropriate ranking function, since ranking scores are sensitive to different scales in different dimensions and moreover there are no explicit guidelines to select ranking functions.

In literature, Pei et al. [23] proposed the *probabilistic skyline query*, which retrieves all the uncertain objects that have the expected probability of being skyline greater than a threshold. The skyline query has one nice property in the sense that its query processing does not require users to specify a ranking function. Furthermore, its query result is invariable to scales in different dimensions. However, one problem with the skyline definition is that the size of the skyline answer set cannot be flexibly controlled by users. That is, users might be overwhelmed by too many returned skyline objects.

Motivated by the shortcomings of both queries above, in this paper, we formulate an important query, namely *probabilistic top-k dominating* (PTD) query, in the context of uncertain databases. Specifically, a PTD query obtains k uncertain objects in an uncertain database that are expected to be better than (called *dynamically dominate*) the largest number of objects with respect to a query point. Note that, the PTD query has the advantages of both probabilistic ranked and skyline queries, that is, invariable to scales in different dimensions, without users' efforts to specify ranking functions, and with the control on the size of the answer set.

The PTD query has many practical applications. For example, in a coal mine surveillance application [33, 16], sensors are deployed in the mine to collect sensory data (i.e. samples) such as density of oxygen, gas, and dust, as well as temperature and humidity. Dangerous events like fires or gas leakage in the coal mine usually follow some patterns (a.k.a. contour maps [33]) in the data. Thus, once such a dangerous event is detected, workers should evacuate from the mine. For the sake of environmental factors, device failure, or transmission delay, the collected samples from sensors inherently contain noises. It is therefore important for coal manager to *accurately* detect dangerous patterns on such uncertain data (note: false alarms may lead to loss of millions of dollars for each evacuation [33]). Here, samples collected from each sensor within a period can be considered as instances of an uncertain object (with attributes like temperature and oxygen density) in the uncertain database. Given a query pattern (e.g. a fire pattern with temperature and oxygen density thresholds), a coal manager can conduct a PTD query to obtain k sensors that are most likely to encounter fire events among all the monitored places. Intuitively, a place has high chance to be on fire, if sensor data reported in many other places are farther away from a fire pattern than data collected in this place on all attributes.

To our best knowledge, no previous work has studied the PTD problem in the uncertain database. Yiu and Mamoulis [34] explored the top-k dominating query in the "certain" database with a static setting, which is however not directly applicable to uncertain data (otherwise, inaccuracy or even errors may be introduced due to the data uncertainty).

Basically, the complexity and challenges of our PTD query processing are twofold. First, rather than *static skyline* where attribute values of each object are fixed (in the definition of [34, 23]), in this paper, we consider *dynamic skyline* [22, 9] such that each attribute of objects is dynamically computed with respect to an ad-hoc query point. Second, given an uncertain database, our PTD query is equivalent to processing a top-k dominating query over each combination of instances from uncertain objects and then aggregating (condensing) the query results in all combinations to obtain answers with the highest ranks. However, it is inefficient and even infeasible to materialize every possible combinations (due to the exponential size) to conduct queries, which results in our efficiency concerns of PTD query processing. Thus, specific techniques should be designed for efficiently answering PTD queries without materializing all the instance combinations.

Therefore, in the sequel, we first formalize the PTD query in the uncertain database. Then, to efficiently answer PTD queries, we propose effective pruning methods to reduce the PTD search space and seamlessly integrate them into an efficient query procedure. Moreover, by further trading the accuracy for efficiency, we propose an approximate approach which utilizes a *probabilistic FM-sketch* to facilitate the PTD query processing. In addition, the PTD query processing with uncertain query object is also discussed.

In particular, we make the following contributions.

1. We formulate and tackle the problem of *probabilistic top-k dominating* (PTD) query in the context of uncertain databases in Section 3 .
2. We present heuristics of our pruning methods and propose an efficient approach to retrieve the exact answer to PTD queries in Section 4.
3. We propose an efficient approach to obtain approximate PTD query answers in Section 5, trading accuracy for efficiency.
4. We extend the proposed techniques to the case where query point is also uncertain in Section 6.

In addition, Section 2 briefly overviews the top-k dominating query processing over precise data and previous works on query processing in uncertain databases. Section 7 demonstrates the performance of PTD query processing through extensive experiments. Section 8 concludes this paper.

2. RELATED WORK

In this section, we briefly overview previous works on the top-k dominating query in the "certain" database and query processing in the context of uncertain databases.

Yiu and Mamoulis [34] recently proposed the top-k dominating query in the spatial database that contains precise data points, which ranks data points by the number of dominating points in the database. As mentioned in Section 1, the top-k dominating query has the advantages of both top-k and skyline queries. In contrast, our *probabilistic top-k dominating* (PTD) query aims to handle this query type over uncertain data, which is more complex and challenging due to the handling of exponential number of possible instance combinations. Furthermore, we consider the dynamic dominance relationship between pairs of instances with respect to an ad-hoc query point, rather than the static dominance in [34], which is more general in many real applications like image retrieval or sensor data monitoring.

Uncertain query processing has received an increasing attention in many applications. For example, the Orion system [7] manages uncertain data in applications like sensor data monitoring; the TRIO system [1] proposes working models to capture data uncertainty on different levels. In the context of uncertain databases, various queries have been proposed, including *range query* [6, 29, 3], *nearest neighbor query* [5, 6, 14], *skyline query* [23], *reverse skyline query* [17], *ranked query* [18], and *similarity join* [13, 20]. In particular, Lian and Chen [18] illustrated the ranked query processing over uncertain data, where the rank of each object is determined by a user-specified linear function. Pei et al. [23] proposed a probabilistic skyline query over uncertain objects which have static attributes. As mentioned earlier, the specification of the ranking function in the ranked query requires users' efforts, and for skyline query, we cannot control the size of the resulting answer set (in [23], although a probabilistic threshold can be specified, users still have to guess which threshold to use). In contrast, users do not need to specify ranking functions in the PTD query, and the size (i.e. k) of the PTD answer set can be controlled by users.

In literature of probabilistic databases [25, 28, 12, 27, 24, 26, 11], the probability that an object belongs to the database might be smaller than 1 (in contrast, uncertain objects must exist in the uncertain database). There are some existing works on top-k query processing [28, 12, 24, 11], which ranks probabilistic data by aggregating query results under *possible worlds* semantics. In order to retrieve semantically meaningful results, different aggregation methods have been proposed. Similar to the ranked query in the uncertain database, the top-k query processing in the probabilistic database requires users to specify a ranking function, which is highly sensitive to scales of dimensions (given that ranking function is fixed). In our work, we focus on the PTD query in the uncertain database where uncertain objects must belong to the database, and users do not need to give ranking functions.

3. PROBLEM DEFINITION

3.1 Dynamic Dominance Relationship

First, we give the definition of dynamic dominance between two points u and v with respect to a query point q.

DEFINITION 3.1. (*Dynamic Dominance [9]*) *Given a query point q and two points X and Y, point X dynamically dominates point Y with respect to q (denoted as $X \prec_q Y$), iff 1) $|X.A_i - q.A_i| \leq |Y.A_i - q.A_i|$ for all dimensions $1 \leq i \leq d$, and 2) $|X.A_j - q.A_j| < |Y.A_j - q.A_j|$ for at least one dimension $1 \leq j \leq d$, where $X.A_i$ is the coordinate of point X on the i-th dimension.*

Example 1. *Fig. 1 illustrates a 2D example of dynamic dominance relationship between points. In particular, we have $|u.A_i - q.A_i| < |v.A_i - q.A_i|$ for $i = 1, 2$ (i.e. point v is in the shaded regions of the figure). Thus, according to Definition 3.1, point u dynamically dominates v with respect to q (i.e. $u \prec_q v$). Similarly, we have point u dynamically dominates point w (i.e. $u \prec_q w$).*

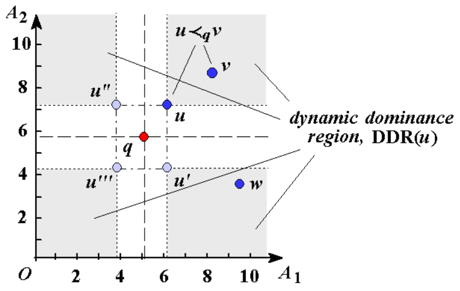


Figure 1: Illustration of Dynamic Dominance Relationship

Since the attributes of objects are calculated as the absolute differences of coordinates from objects to query point q, in Fig. 1, we can symmetrically map u to points u', u'', and u''', with respect to lines $y = q.A_2$, $x = q.A_1$, and point q, respectively. This way, we can obtain totally four shaded regions (with corner points u, u', u'', and u''', respectively), which are called *dynamic dominance regions*, denoted as DDR(u). Obviously, any object that falls into the shaded regions is dynamically dominated by point u.

3.2 PTD Query in Uncertain Databases

In the above example of Fig. 1, a *top-k dominating query* with dynamic dominance in a "certain" database [34] is to find k points in the data space such that their dynamic dominance regions cover the largest number of data points. In the sequel, we will define probabilistic top-k dominating query in the uncertain database.

DEFINITION 3.2. (*Uncertain Database [5, 23]*) *An uncertain database $\mathcal{D}$ contains N uncertain objects. Each object t can be represented by a set $\{t_1, t_2, ..., t_{|t|}\}$, where t_i $(1 \leq i \leq |t|)$ are instances of t which contain d numerical attributes A_1, A_2, ..., and A_d, as well as their appearance probabilities $t_i.p$ satisfying $\sum_{i=1}^{|t|} t_i.p = 1$.*

Following the convention [6, 5, 23], we assume that objects in the uncertain database $\mathcal{D}$ are independent of each other. The instance combinations over $\mathcal{D}$ are defined as follows.

DEFINITION 3.3. (*Instance Combination, IC^l*) *Given an uncertain database $\mathcal{D}$, an instance combination IC^l in $\mathcal{D}$ is a combination of instances obtained from all uncertain objects in $\mathcal{D}$ (for every combination, each object contributes one instance), which has the existence probability $Pr(IC^l) = \prod_{\forall t \in \mathcal{D} \wedge \exists t_i \in IC^l} t_i.p$, where l is the index of instance combinations.*

In Definition 3.3, each uncertain object t has one and only one instance t_i that appears in each instance combination IC^l. Thus, the existence probability $Pr(IC^l)$ of IC^l is given by the multiplication of appearance probabilities for all instances t_i in IC^l. Note that, the concept of instance combination here is similar to that of possible world in the probabilistic database [26]. The difference is that in the probabilistic database, each uncertain object (i.e. x-tuple) may not contribute any instance (i.e. alternative) to the instance combination (i.e. possible world) [1, 23].

Next, we consider the dominance probability, $Pr\{t \prec_q s\}$, of two uncertain objects t and s in an uncertain database $\mathcal{D}$, which is the probability that t dynamically dominates s with respect to q. Without loss of generality, assuming uncertain objects t and s have instance sets $\{t_1, t_2, ..., t_{|t|}\}$ and $\{s_1, s_2, ..., s_{|s|}\}$, respectively, we have:

$$Pr\{t \prec_q s\} = \sum_{\forall l, t_i \in IC^l \wedge s_j \in IC^l \wedge t_i \prec_q s_j} Pr(IC^l). \quad (1)$$

Intuitively, the probability that an uncertain object t dynamically dominates another uncertain object s is given by aggregating (summing up) the existence probabilities of those instance combinations IC^l (containing instances $t_i \in t$ and $s_j \in s$), in which instance t_i dynamically dominates s_j.

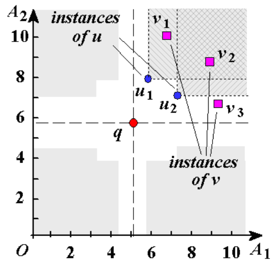


Figure 2: Example of Probabilistic Top-k Dominating Query

Example 2. *As illustrated in Fig. 2, assume an uncertain object u has two instances u_1 and u_2 (both with appearance probabilities $1/2$), and uncertain object v has three instances v_1, v_2, and v_3 (with $1/3$ probability each). Each instance has two attributes A_1 and A_2, corresponding to a point in a 2D space shown in the figure. From Definition 3.1, we know that instance u_1 dynamically dominates instances v_1 and v_2, and instance u_2 dynamically dominates instance v_2. Therefore, the probability that uncertain object u dynamically dominates v is given by $u_1.p \times (v_1.p + v_2.p) + u_2.p \times v_2.p = \frac{1}{2} \times (\frac{1}{3} + \frac{1}{3}) + \frac{1}{2} \times \frac{1}{3} = \frac{1}{2}$.*

After introducing the probability of dynamic dominance between two uncertain objects, we are now ready to define the PTD query below.

Symbol	Description
$\mathcal{D}$	an uncertain database containing N uncertain objects
t, s, u, v, w	the uncertain objects
t_i, s_j, u_i, v_i, w_i	the instances of uncertain object
$\|t\|$	the number of instances in uncertain object t
q	the query object
$Pr\{t \prec_q s\}$	the probability that uncertain object t *dynamically dominates* s with respect to q
T_{min} (T_{max})	the point in the minimum bounding rectangle of t nearest to (farthest from) query object q

Table 2: Meanings of Notations

DEFINITION 3.4. (*Probabilistic Top-k Dominating Query, PTD*) *Given an uncertain database $\mathcal{D}$ and a query point q, a probabilistic top-k dominating query (PTD) retrieves k uncertain objects $t \in \mathcal{D}$ that are expected to dynamically dominate the largest number of uncertain objects (with respect to q) for all instance combinations. That is, a PTD query obtains k objects, t, such that they have the highest scores $score(t) = \sum_{\forall s \in \mathcal{D} \wedge s \neq t} Pr\{t \prec_q s\}$, where $Pr\{t \prec_q s\}$ is given by Eq. (1).*

Definition 3.4 gives a natural ranking on uncertain objects in the uncertain database, that is, the more the uncertain object t dynamically dominates, the higher rank t would achieve. Table 3.2 summarizes the commonly-used symbols in this paper.

4. PROBABILISTIC TOP-K DOMINATING SEARCH

After formalizing the PTD problem, in this section, we study the efficiency issues of answering the PTD query in an uncertain database $\mathcal{D}$. Naturally, one straightforward way to obtain the PTD answer is as follows. For each uncertain object $t \in \mathcal{D}$, we sequentially scan instances s_j in the entire database and check the dominance relationship between instances t_i and s_j for each instance combination in the database, meanwhile evaluate the score, $score(t)$, defined in Definition 3.4. Clearly, this nested loop method is very costly, which requires quadratic cost with respect to the database size $|\mathcal{D}|$, in terms of both CPU time and I/O's.

Therefore, in order to enable fast PTD query processing, we construct a multidimensional index, like aggregate R-tree (aR-tree) [15], over d-dimensional instances (each of the d numerical attributes corresponds to one dimension) in the database, which can provide fast access to uncertain data rather than the sequential scan.

In particular, we first divide each uncertain object $t \in \mathcal{D}$ $(= \{t_1, t_2, ..., t_{|t|}\})$ into $m(t)$ groups (clusters) $G_1(t)$, $G_2(t)$, ..., and $G_{m(t)}(t)$ of approximately the same size via either space- or data-partitioning method. Note that, here the number of groups, $m(t)$, for each uncertain object t is chosen such that group $G_i(t)$ is small enough to fit in one disk page. Then, we bound each of these groups, $G_i(t)$, using a *minimum bounding rectangle* (MBR), and insert them into an aR-tree index (using standard insertion operator of aR-tree), together with its object id and an aggregate, $(G_i(t)).agg$, defined as the total appearance probability of instances in $G_i(t)$ (i.e. $G_i(t).agg = \sum_{t_j \in G_i(t)} t_j.p$). The instance groups are recursively bounded by MBRs until finally one root node is obtained. We use *sum* as the aggregate function in each intermediate node, which sums up all the aggregate values in its children.

Fig. 3 illustrates an example of aR-tree, where the leftmost leaf node (pointed to by N_3) contains groups $G_3(t)$ and $G_2(u)$, with aggregates $G_3(t).agg = 0.2$ and $G_2(u).agg = 0.4$, respectively. Thus, the aggregate of this node is given by $0.2 + 0.4 = 0.6$. Similarly, the second leaf node from left (pointed to by N_4) has aggregate $0.3 + 0.6 + 0.5 = 1.4$. In the parent of these two leaf nodes (pointed to by N_1), we store one aggregate which is sum of aggregates in its children (i.e. $2 = 0.6 + 1.4$).

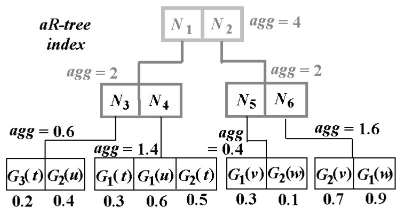


Figure 3: Illustration of aR-tree

In the following subsections, we first present the basic pruning heuristics of our PTD approach. Then, we illustrate the detailed PTD query processing on the constructed index.

4.1 Preliminary

Since our PTD query aims to find k uncertain objects in the uncertain database with the highest scores, we first investigate the formula of the score (defined in Definition 3.4). In particular, the score $score(t)$ of an uncertain object t is given by the summation of probabilities $Pr\{t \prec_q s\}$ that t dynamically dominates other uncertain objects s, where $Pr\{t \prec_q s\}$ is the aggregated (summed) probabilities for all instance combinations defined in Eq. (1). Note, however, that the computation of $Pr\{t \prec_q s\}$ needs to consider exponential number of instance combinations, which is in practically inefficient or even infeasible to calculate directly.

Alternatively, in the sequel, we equivalently consider each individual instance $t_i \in t$, obtain the expected number of instances in the database that t_i dynamically dominates, among all instance combinations, and finally aggregate the expected numbers for all t_i. Formally, we can rewrite the definition of $score(t)$ in Definition 3.4, and obtain the lemma below.

LEMMA 4.1. *The probability $Pr\{t \prec_q s\}$ in score, $score(t)$, of uncertain object t in the uncertain database $\mathcal{D}$ is given by:*

$$Pr\{t \prec_q s\} = \sum_{i=1}^{|t|} \left(t_i.p \cdot \left(\sum_{j \in [1,|s|] \wedge t_i \prec_q s_j} s_j.p \right) \right). \qquad (2)$$

Proof. Derived from Definition 3.4 and Eq. (1). □

Intuitively, in Eq. (2), the probability $\sum_{j \in [1,|s|] \wedge t_i \prec_q s_j} s_j.p$ is the summed appearance probability of instance s_j such that t_i dynamically dominates s_j in the d-dimensional space. This way, our PTD problem of finding k uncertain objects with the highest scores defined with exponential number of instance combinations can be now reduced to the one that answers spatial aggregate queries (i.e. summing up appearance probabilities of instances) in a d-dimensional space for each instance t_i of uncertain object t.

As mentioned earlier, based on the score definition, we can use the straightforward nested loop approach to obtain the PTD answers. However, even by using the rewritten formula, Eq. (2), the direct calculation is still not efficient due to its complexity $O(N^2)$ in the worst case, where N is the data size. Thus, instead, we target at obtaining the lower and upper bounds of score at a low cost, which can help reduce the PTD search space effectively.

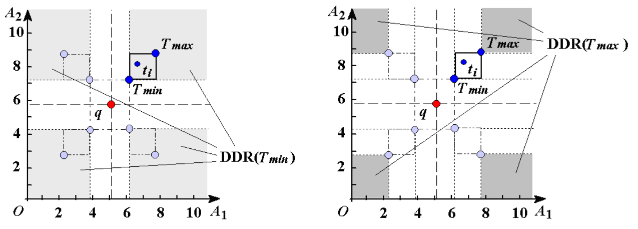
4.2 Bounding Scores of Uncertain Objects

In this subsection, we illustrate how to obtain lower and bounds of $score(t)$ using a 2D example in Fig. 4, where we are given a PTD query point q and an uncertain object t in a 2D space. For the sake of clear illustration, we assume each uncertain object t is represented by one group (i.e. $t = G_1(t)$ for $m(t) = 1$). At the end of this subsection, we will extend our solution to the case of dealing with arbitrary number (i.e. $m(t) > 1$) of groups.

In Fig. 4, let T_{min} and T_{max} be two points in the bounding rectangle of t that are the nearest to and farthest from q, respectively. Fig. 4(a) and Fig. 4(b) show the dynamic dominance regions (shaded areas) DDR(T_{min}) and DDR(T_{max}) of points T_{min} and T_{max}, respectively, with respect to query point q. Clearly, for any instance $t_i \in t$, its dominance region DDR(t_i) always satisfies the containment relationship below:

$$\text{DDR}(T_{max}) \subseteq \text{DDR}(t_i) \subseteq \text{DDR}(T_{min}). \qquad (3)$$

In other words, if we sum up appearance probabilities for all the instances falling into region DDR(T_{max}) (DDR(T_{min})), then we can obtain a lower (upper) bound of probability $\sum_{j\in[1,|s|]\wedge t_i \prec_q s_j} s_j.p$ in Eq. (2). From this interesting observation, we give the lower/upper bounds of $score(t)$ below.



(a) DDR(T_{min}) (b) DDR(T_{max})

Figure 4: Bounding the Score $score(t)$

Formally, in a d-dimensional uncertain database $\mathcal{D}$, the nearest and farthest points in t to q are given by: $T_{min} = \langle min_{i=1}^{|t|}\{|t_i.A_1 - q.A_1|\}, min_{i=1}^{|t|}\{|t_i.A_2 - q.A_2|\}, ..., min_{i=1}^{|t|}\{|t_i.A_d - q.A_d|\}\rangle$ and $T_{max} = \langle max_{i=1}^{|t|}\{|t_i.A_1 - q.A_1|\}, max_{i=1}^{|t|}\{|t_i.A_2 - q.A_2|\}, ..., max_{i=1}^{|t|}\{|t_i.A_d - q.A_d|\}\rangle$, respectively, where $X.A_i$ is the i-th coordinate of point X. We have the following lemma:

LEMMA 4.2. *(Lower and Upper Bounds of Score $score(t)$) Let $score(T_{min}) = \sum_{\forall s_j \in DDR(T_{min})} s_j.p = \sum_{\forall s\in\mathcal{D}\wedge s\neq t\wedge T_{min}\prec_q s_j} s_j.p$, and similarly $score(T_{max}) = \sum_{\forall s_j \in DDR(T_{max})} s_j.p = \sum_{\forall s\in\mathcal{D}\wedge s\neq t\wedge T_{max}\prec_q s_j} s_j.p$. Then, we have:*

$$score(T_{max}) \leq score(t) \leq score(T_{min}). \qquad (4)$$

Proof Sketch. From the definitions of points T_{min} and T_{max}, for any instance $t_i \in t$, we have the dynamic dominance relationship $t_i \prec_q T_{max}$ and $T_{min} \prec_q t_i$.

For any uncertain object s, if we have $T_{max} \prec_q s_j$ for some $s_j \in s$, then it holds that $t_i \prec_q T_{max} \prec_q s_j$ (due to the transitive property of the dynamic dominance). Thus, we have $\sum_{\forall s\in\mathcal{D}\wedge s\neq t\wedge T_{max}\prec_q s_j} s_j.p \leq \sum_{\forall s\in\mathcal{D}\wedge s\neq t\wedge t_i\prec_q s_j} s_j.p$. Therefore, by Eq. (2), we obtain: $score(T_{max}) = (\sum_{i=1}^{|t|} t_i.p)\cdot score(T_{max}) = \sum_{i=1}^{|t|}(t_i.p\cdot(\sum_{\forall s\in\mathcal{D}\wedge s\neq t\wedge T_{max}\prec_q s_j} s_j.p)) \leq \sum_{i=1}^{|t|}(t_i.p\cdot(\sum_{\forall s\in\mathcal{D}\wedge s\neq t\wedge t_i\prec_q s_j} s_j.p)) = score(t)$. Hence, the first part of Inequality (4) is correct. The proof of $score(t) \leq score(T_{min})$ is similar and thus omitted. □

Next, we consider the general case where instances in each uncertain object t are divided into $m(t)$ (≥ 1) groups, $G_1(t)$, $G_2(t)$, ..., and $G_{m(t)}(t)$. Similar to the case with single group, we can compute the lower and upper bounds for score $score(G_i(t))$ as follows. Let $G_i(t)_{min}$ and $G_i(t)_{max}$ be points in the MBR of group $G_i(t)$ nearest to and farther from query point q, respectively. For each group $G_i(t)$, we can find its lower and upper bounds with respect to $G_i(t)_{max}$ and $G_i(t)_{min}$, respectively, by using Lemma 4.2. We have the following lemma for score bounds with multiple groups.

LEMMA 4.3. *Assume an uncertain object t has $m(t)$ groups of instances, $G_1(t)$, $G_2(t)$, ..., and $G_{m(t)}(t)$. Then, we have:*

$$\sum_{i=1}^{m(t)}\left(\left(\sum_{\forall t_j\in G_i(t)} t_j.p\right)\cdot score(G_i(t)_{max})\right) \leq score(t)$$
$$\leq \sum_{i=1}^{m(t)}\left(\left(\sum_{\forall t_j\in G_i(t)} t_j.p\right)\cdot score(G_i(t)_{min})\right). \qquad (5)$$

Proof. For each group $G_i(U)$, by Lemma 4.2, we have:

$$\left(\sum_{\forall t_j\in G_i(t)} t_j.p\right)\cdot score(G_i(t)_{max}) \leq score(G_i(t))$$
$$\leq \left(\sum_{\forall t_j\in G_i(t)} t_j.p\right)\cdot score(G_i(t)_{min}) \qquad (6)$$

Moreover, based on Lemma 4.1, it holds that:

$$score(G_i(t)) = \sum_{\forall t_i\in G_i(t)}\left(t_i.p\cdot\sum_{s\in\mathcal{D}\wedge s\neq t\wedge t_i\prec_q s_j} s_j.p\right).$$

Thus, we have:

$$score(t) = \sum_{i=1}^{m(t)} score(G_i(t)). \qquad (7)$$

Therefore, by combining Inequality (6) and Eq. (7), we obtain Inequality (5), which completes our proof. □

For brevity, in the sequel, we denote the lower and upper bound of $score(t)$ as $LB\_score(t)$ and $UB\_score(t)$, respectively, which are given in Inequality (5).

4.3 Pruning Heuristics

After providing the score bounds for uncertain objects, we are now ready to illustrate the intuition of our pruning method for reducing the PTD search space. Fig. 5 illustrates a 2D $object\text{-}score$ space, where the horizontal axis indicates the uncertain objects (e.g. t, s, or u), and the vertical axis represents the scores of the corresponding uncertain objects. As discussed above, instead of expensively computing the exact scores of uncertain objects, we can obtain score intervals at a lower cost, into which the actual scores fall. For example, for uncertain object t, we have its score $score(t)$ within interval $[LB\_score(t), UB\_score(t)]$.

Now we consider a probabilistic top-1 dominating query (i.e. $k = 1$) in the example of Fig. 5, which retrieves an uncertain object that has the highest score (defined in Eq. (2)). The rationale behind our pruning method is as follows. First, we find the largest (i.e. 1st largest) lower bound of score for all uncertain objects in the database. In the example, this largest lower bound is given by $LB\_score(t)$, which can be used as a threshold for pruning. In particular, for uncertain object u, since its upper bound score has already been smaller than $LB\_score(t)$, that is, $UB\_score(u) < LB\_score(t)$, u cannot be a qualified answer to our PTD query. Thus, uncertain object u can be safely pruned, without calculating its exact score $score(u)$. On the other hand, uncertain object s cannot be pruned, since its upper bound $UB\_score(s)$ is above threshold $LB\_score(t)$ (i.e. s still has chance to have the highest score in the database).

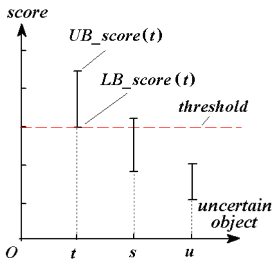


Figure 5: **Illustration of Pruning Heuristics**

We summarize the pruning condition in the example above in the following lemma.

LEMMA 4.4. (*Pruning Condition*) *Let k_lb_score be the k-th largest lower bound of score for any k uncertain objects in the database. Any uncertain object s can be safely pruned if it holds that $UB\_score(s) < k\_lb\_score$.*

Proof. Since k_lb_score is the k-th largest lower bound of score of k uncertain objects (considered as threshold), it indicates that there are at least k uncertain objects whose exact scores are above threshold k_lb_score. Moreover, since $UB\_score(s) < k\_lb\_score$, we have inequality $score(s) \leq UB\_score(s) < k\_lb\_score$. Therefore, uncertain object s has its exact score lower than k_lb_score (in other words, there are at least k uncertain objects having scores higher than s). From the PTD definition (in Definition 3.4), s cannot be the query answer, and thus it can be safely pruned. □

From the pruning condition in Lemma 4.4, only those uncertain objects that have their score upper bounds greater than threshold k_lb_score are the PTD candidates.

4.4 PTD Query Processing

In this subsection, we present the detailed procedure for PTD query processing. In particular, we first give the baseline method, namely PTD_Baseline, in Fig. 6 to answer PTD queries.

4.4.1 PTD Baseline Algorithm

Specifically, procedure PTD_Baseline scans each uncertain object t in the database $\mathcal{D}$ and computes the lower and upper bounds of its score ($LB\_score(t)$ and $UB\_score(t)$, respectively; lines 1-3). Moreover, we also keep a threshold k_lb_score, which is defined as the k-th largest lower bound of score in the PTD candidate set S_{cand}. When S_{cand} has fewer than k uncertain objects, we add any uncertain object t we encounter to the set S_{cand} (lines 4-5). When the size of S_{cand} reaches or exceeds k, we insert an uncertain object t into S_{cand} only if $UB\_score(t) > k\_lb\_score$ holds, which is guided by our pruning condition in Lemma 4.4. Since threshold k_lb_score may become larger due to the inclusion of t in S_{cand}, we can remove some uncertain objects s from S_{cand}, which satisfy the pruning condition $UB\_score(s) < k\_lb\_score$ (lines 6-8). Finally, we refine the remaining candidates in S_{cand} by computing their actual scores and return k uncertain objects with the highest scores as the PTD query results (line 9).

```
Procedure PTD_Baseline {
  Input: aR-tree $\mathcal{I}$ constructed over $\mathcal{D}$, and query point $q$
  Output: the answer to the PTD query
  (1)  $S_{cand} = \phi$, $k\_lb\_score = +\infty$;
  (2)  for each uncertain object $t \in \mathcal{D}$
  (3)    $[LB\_score(t), UB\_score(t)]$=Bounding_Score $(q, t, root(\mathcal{I}))$
  (4)    if $|S_{cand}| < k$
  (5)      add $t$ to $S_{cand}$
  (6)    else if $UB\_score(t) > k\_lb\_score$
  (7)           add $t$ to $S_{cand}$ and set $k\_lb\_score$ to the $k$-th largest lower
                  bound in $S_{cand}$
  (8)           remove uncertain objects $s$ from $S_{cand}$ satisfying $UB\_score(s)$
                  $< k\_lb\_score$
  (9)  refine candidates in $S_{cand}$ by calculating their real scores and return
         the actual PTD answer
}
Procedure Bounding_Score {
  Input: query point $q$, uncertain object $t$, and a node $N$ of aR-tree $\mathcal{I}$
  Output: the lower and upper bounds of $score(t)$
  (1)  let uncertain object $t$ contain groups $G_1(t)$, $G_2(t)$, ..., $G_{m(t)}(t)$
  (2)  $LB\_score(t) = UB\_score(t) = 0$;
  (3)  if $N$ is a leaf node
  (4)    for each entry $G_j(s) \in N$
  (5)      for each group $G_i(t)$
  (6)        if $G_j(s) \subseteq$ DDR$(G_i(t)_{max})$
  (7)          $LB\_score(t) = LB\_score(t) + G_i(t).agg \cdot G_j(s).agg$
  (8)        if $G_j(s) \cap$ DDR$(G_i(t)_{min}) \neq \phi$
  (9)          $UB\_score(t) = UB\_score(t) + G_i(t).agg \cdot G_j(s).agg$
  (10) else // intermediate node
  (11)   for each entry $N_j \in N$
  (12)     if there exists a group $G_i(t)$ such that $N_j$ partially intersects with
               DDR$(G_i(t)_{max})$ or DDR$(G_i(t)_{min})$
  (13)       $[lb, ub]$ = Bounding_Score $(q, t, N_j)$;
  (14)       $LB\_score(t) = LB\_score(t) + lb$
  (15)       $UB\_score(t) = UB\_score(t) + ub$
  (16)     else for each group $G_i(t)$
  (17)            if $N_j \subseteq$ DDR$(G_i(t)_{max})$
  (18)              $LB\_score(t) = LB\_score(t) + G_i(t).agg \cdot N_j.agg$
  (19)            if $N_j \subseteq$ DDR$(G_i(U)_{min})$
  (20)              $UB\_score(t) = UB\_score(t) + G_i(t).agg \cdot N_j.agg$
}
```

Figure 6: **PTD Baseline Algorithm**

In particular, we address how to calculate the lower and upper bounds of the score for an uncertain object t. As shown in line 3 of procedure PTD_Baseline, we invoke procedure Bounding_Score (also in Fig. 6) and obtain the score bounds by traversing the aR-tree index $\mathcal{I}$.

Specifically, procedure Bounding_Score starts from the root (i.e. $root(\mathcal{I})$) of index $\mathcal{I}$, and accesses its children iteratively. When we encounter a leaf node N, for each entry $G_j(s)$ (i.e. one group for an uncertain object s), we check whether or not $G_j(s)$ is fully contained in the dynamic dominance region of group $G_i(t)$, that is, DDR$(G_i(t)_{max})$. In case the answer is yes, we update the lower bound $LB\_score(t)$ by adding $G_i(t).agg \cdot G_j(s).agg$ (Lemma 4.3), where $G_j(s).agg$ stores the summed appearance probabilities of all instances in group $G_j(s)$ in the aR-tree (lines 3-7). Similarly, if $G_j(s)$ and DDR($G_i(t)_{min}$) have non-empty intersection, we increase $UB\_score(t)$ by $G_i(t).agg \cdot G_j(s).agg$ (i.e. overestimated by assuming all the instances in $G_j(s)$ are contained within the region DDR($G_i(t)_{min}$); lines 8-9). On the other hand, when N is a non-leaf node, we process each entry N_j in N as follows. If there exists a group $G_i(t)$ such that N_j partially intersects with DDR($G_i(t)_{max}$) or DDR($G_i(t)_{min}$), we need to recursively access its children by invoking procedure Bounding_Score itself with node N_j, in order to obtain accurate lower/upper bound of scores (lines 12-13). The returned bounds lb and ub are added to $LB\_score(t)$ and $UB\_score(t)$, respectively (lines 14-15). If there is no partial intersection between any $G_i(t)$ and DDR, similar to the case of handling leaf nodes, we check the relationship between N_j and DDR($G_i(t)_{max}$) (or DDR($G_i(t)_{min}$)) and update the lower (upper) bound $LB\_score(t)$ ($UB\_score(t)$) accordingly (lines 16-20).

4.4.2 PTD Query Procedure

Although procedure PTD_Baseline can correctly answer the PTD query, as mentioned in the last subsection, it invokes procedure Bounding_Score and visits the aR-tree for *every* uncertain object t in the database $\mathcal{D}$, which is not that efficient in terms of both computation and I/O costs. This is because the calculation of score bounds for all objects in $\mathcal{D}$ has to traverse the tree nodes multiple times. Observing this, in the sequel, we propose a more efficient method to process PTD queries by accessing each node of the aR-tree at most once. Intuitively, this can be achieved by updating the score bounds of other nodes/instances whenever a node is visited. Thus, if a visited node is pruned, we do not need to access it some time later to calculate the score bounds of other nodes.

```
Procedure PTD_Processing {
  Input: aR-tree $\mathcal{I}$ constructed over $\mathcal{D}$, and query point $q$
  Output: the answer to the PTD query
  (1) initialize a max-heap $\mathcal{H}$ accepting entries in the form $(N, key)$
  (2) $S_{cand} = \phi$;
  (3) for each entry $N_i \in root(\mathcal{I})$
                  // $LB\_score(N_i)$ and $UB\_score(N_i)$ are initially 0
  (4)     for each entry $N_j \in root(\mathcal{I})$ such that $i \neq j$
  (5)         if $N_j \subseteq$ DDR($N_{i,max}$)
  (6)             $LB\_score(N_i)$=$LB\_score(N_i)$ + $N_j.agg$
  (7)         if $N_j \cap$ DDR($N_{i,min}$) $\neq \phi$
  (8)             $UB\_score(N_i)$=$UB\_score(N_i)$ + $N_j.agg$
  (9)     insert $(N_i, UB\_score(N_i))$ into heap $\mathcal{H}$
  (10) let $k\_lb\_score$ be the $k$-th largest score lower bound for $N_i$
            $\in root(\mathcal{I})$
  (11) while $\mathcal{H}$ is not empty and $top(\mathcal{H}).key \geq k\_lb\_score$
  (12)     $(N, key)$ = de-heap $\mathcal{H}$
  (13)     if $N$ is a leaf node
  (14)         for each entry $G_i(t) \in N$
  (15)             update score lower/upper bounds for nodes in heap $\mathcal{H}$
  (16)             if $G_i(t)$ is the first group of $t$ in $S_{cand}$
  (17)                 if $score(G_i(t)_{min})$ + $(1 - G_i(t).agg)$ ·$top(\mathcal{H}).key$
                           $\geq k\_lb\_score$
  (18)                     create a new entry in $S_{cand}$ for $G_i(t)$ in the form
                           $\langle G_i(t).agg, score(G_i(t)_{max}), score(G_i(t)_{min})\rangle$
  (19)             else // update entry $\langle cur\_agg, cur\_lb, cur\_ub\rangle$ in $S_{cand}$
  (20)                 $cur\_agg = cur\_agg + G_i(t).agg$
  (21)                 $cur\_lb = cur\_lb + G_i(t).agg \cdot score(G_i(t)_{max})$
  (22)                 $cur\_ub = cur\_ub + G_i(t).agg \cdot score(G_i(t)_{min})$
  (23)                 if $cur\_ub + (1 - cur\_agg) \cdot top(\mathcal{H}).key < k\_lb\_score$
  (24)                     mark $t$ as a false alarm
  (25)         update $k\_lb\_score$
  (26)     else // intermediate node
  (27)         for each entry $N_i \in N$
  (28)             update score lower/upper bounds for nodes in heap $\mathcal{H}$
  (29)             if $UB\_score(N_i) \geq k\_lb\_score$ // pruning
  (30)                 add $(N_i, UB\_score(N_i))$ to heap $\mathcal{H}$
  (31) refine candidates in $S_{cand}$ by calculating the exact scores
  (32) return $k$ uncertain objects with the highest scores
}
```

Figure 7: PTD Query Processing

Specifically, Fig. 7 illustrates the detailed query procedure, namely PTD_Processing, which traverses the aR-tree $\mathcal{I}$ by maintaining a maximum heap $\mathcal{H}$ accepting entries in the form (N, key), where N is a tree node and key is defined as the upper bound of score for any possible instance in N (line 1). We also maintain a PTD candidate set S_{cand} (initially empty, line 2) for storing possible query answers in the form $\langle cur\_agg, cur\_lb, cur\_ub\rangle$, where cur_agg is the summed appearance probability of instances in an uncertain object t that we have seen so far, and cur_lb and cur_ub are the score lower and upper bounds for instances in t that we have seen. For example, assume we have seen groups $G_1(t)$, $G_2(t)$, ..., and $G_{cur}(t)$ so far. Then, according to Lemma 4.3 (Eq. (5)), we have entry $\langle \sum_{i=1}^{cur} G_i(t).agg, \sum_{i=1}^{cur} G_i(t).agg \cdot score(G_i(t)_{max}), \sum_{i=1}^{cur} G_i(t).agg \cdot score(G_i(U)_{min})\rangle$, where aggregate $G_i(t).agg$ is the summed appearance probability of all instances in group $G_i(t)$. Note that, here the entry in S_{cand} may not see all the instances of uncertain object t. Thus, we can only obtain lower bound $score(t)$ by cur_lb assuming instances of t that we have not seen would not increase the score (i.e. not dynamically dominated by any instance of t). Similarly, score $score(t)$ can be upper bounded by $cur\_ub + (1 - cur\_agg) \cdot top(\mathcal{H}).key$, assuming those instance of t that have not been seen have scores $top(\mathcal{H}).key$, which is the largest possible key in the current heap $\mathcal{H}$ (i.e. the largest score upper bound for any node that we have not seen). As mentioned in Section 4.3, such obtained score bounds can be used to help prune the search space.

The query procedure PTD_Processing starts from root $root(\mathcal{I})$ of aR-tree and computes lower/upper bounds of scores for root entries which are inserted into the heap $\mathcal{H}$ (lines 3-9). Let threshold k_lb_score be the k-th largest lower bound of score for $N_i \in root(\mathcal{I})$ (line 10). Each time we pop out an entry (N, key) from heap $\mathcal{H}$ (lines 11-12). When N is a leaf node, for each entry $G_i(t)$, we update lower/upper bound of nodes in heap $\mathcal{H}$ and, moreover, decide whether or not to include t in the candidate set S_{cand} (lines 16-25). In particular, if uncertain object t does not have an entry in S_{cand} and the upper bound of $G_i(t)$ (i.e. $score(G_i(t))_{min}$ $+(1 - G_i(t).agg) \cdot top(\mathcal{H}).key)$ is greater than or equal to k_lb_score, then t is a possible candidate and we create an entry for uncertain object t in S_{cand} (lines 16-18); otherwise, we update the existing entry $\langle cur\_agg, cur\_lb, cur\_ub\rangle$ in S_{cand} considering instances in $G_i(U)$ (lines 19-22). In case the updated score upper bound is less than threshold k_lb_score, t can be safely pruned by marking it as a false alarm (Lemma 4.4; lines 23-24). Furthermore, we update threshold k_lb_score with the k-th largest cur_lb in S_{cand}.

When N is a non-leaf node, for each entry N_i in N, we update lower/upper score bounds for nodes in heap $\mathcal{H}$ (lines 27-28). If the score upper bound $UB\_score(N_i)$ is greater than or equal to k_lb_score, it indicates that N_i may contain candidates that have their scores higher than k_lb_score. Thus, we need to add N_i to heap $\mathcal{H}$ with key $UB\_score(N_i)$. Otherwise, node N_i can be safely pruned. Note that, for any uncertain object t whose exact upper bound is greater than k_lb_score, there must exist at least one group $G_i(t)$ such that its score upper bound is greater than k_lb_score. Thus, we will not introduce false dismissals if we prune N_i with upper bound lower than k_lb_score. Finally, the iteration terminates until either heap $\mathcal{H}$ is empty or the largest upper bound in the heap is smaller than threshold k_lb_score (line 11).

5. APPROXIMATE PROBABILISTIC TOP-K DOMINATING SEARCH

In this section, instead of exactly answering PTD queries, we propose an approximate approach that can achieve higher efficiency. Section 5.1 presents the basic idea of our approximate solutions. Section 5.2 discusses details of the approximate version of PTD.

5.1 Rationale

Recall from query procedure PTD_Processing (Fig. 7), that the major cost of our PTD query processing is with computing the lower/upper bounds. This is a bottleneck of the cost, since for each expansion of the tree node, we need to update the bounds for other nodes. Motivated by this, in the sequel we aim to efficiently estimate the lower and upper bounds of scores during the PTD query processing. This way, we can significantly reduce the computation cost.

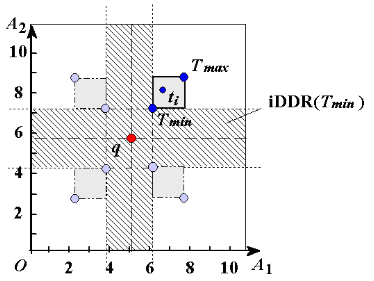


Figure 8: Illustration of Inverse Dynamic Dominance Region

Fig. 8 illustrates the same example as Fig. 4(a). The only difference is in the area with sloped lines, which is the inverse region of dynamic dominance regions DDR(T_{min}) (given in Fig. 4(a)). Formally, let the entire data space be $\mathcal{A}$. The *inverse dynamic dominance region* (iDDR) of a point t_i is defined as iDDR(t_i) = $\mathcal{A}$ - DDR(t_i). In the example, we observe that instances can either fall into regions iDDR(T_{min}) or DDR(T_{min}), while the total number of instances is fixed (assuming the database is given). Therefore, we can conceptually transform our PTD query that retrieves k uncertain objects with the highest scores in DDR to the problem of obtaining k uncertain objects with the lowest scores in iDDR. Note that, such conceptual transformation can greatly help the score estimation during our PTD query processing.

5.2 Approximate Query Procedure

As mentioned above, after the conceptual transformation, instead of computing lower/upper bounds of scores in DDR, now we can aggregate the appearance probabilities of instances that fall into iDDR. Although the computation costs for obtaining the exact PTD answer are similar in these two cases, we can speed up the latter case with approximations, with the help of sketches.

As shown in the 2D example of Fig. 8, region iDDR(T_{min}) (filled with sloped lines) can be considered as a union of two areas, one between lines $A_1 = 3.9$ and $A_1 = 6.1$, and the other between lines $A_2 = 4.2$ and $A_2 = 7.2$. In a generic d-dimensional space, iDDR(t) can be obtained by a union of d smaller regions, that is iDDR(t) = $\cup_{i=1}^{d}$ iDDR$^{(i)}(t)$, where iDDR$^{(i)}(t)$ is a region such that its i-th dimension is constrained by interval $[q.A_i - |q.A_i - T_{min}.A_i|, q.A_i + |q.A_i - T_{min}.A_i|]$. This way, the problem of finding the summed appearance probability of instances in iDDR(q, t) can be further reduced to the one that obtains the expected number of *distinct* uncertain objects (weighted by their summed appearance probability of instances) that fall into these d regions iDDR$^{(i)}(t)$.

Formally, we have d sets of instances falling into d regions, that is, iDDR$^{(i)}(t)$, respectively, where each instance is associated with its appearance probability $t_i.p$ or aggregate $G_i(t).agg$ for group $G_i(t)$. Now we want to estimate the summed appearance probability of all the distinct instances (groups) in these regions.

There are many previous work on estimating distinct values using *FM-sketch*, which however assumes equal importance of each value (or id). Recently, the *probabilistic FM-sketch* (pFM sketch) [8] has been proposed to estimate the distance values with different appearance probability. Specifically, the pFM algorithm maintains an array pFM[] comprising real-valued probability entries. A family of hash functions $h : [M] \rightarrow \{1, 2, ..., logM\}$ are employed such that any value $x \in [M]$ is hashed to the i-th position with probability 2^{-i}. For each incoming value t with appearance probability p, we update pFM sketch with pFM[$h(t)$] = pFM[$h(t)$] $\cdot (1 - p) + p$. Therefore, the expected number of distinct ids in a pFM sketch is given by $\sum_{j=1}^{logM} 2^j \cdot$ pFM[j] $\prod_{k=j+1}^{logM}(1-$ pFM[k]).

In our problem, for each dimension j, we divide the domain along this dimension into l bins of equal size. For example, in Fig. 9, we can divide the domain into 11 bins of size 1 for each dimension (i.e. $Bin_{[0,1]}$, $Bin_{[1,2]}$, ..., and $Bin_{[10,11]}$). In each bin, we maintain a pFM sketch which summarizes all the instances (with ids t and appearance probabilities $t_i.p$) that have their j-th dimension falling into this bin. The basic idea of our estimation for region iDDR(t) is as follows. Given a region iDDR(t), we can obtain all the bins that intersect with region iDDR$^{(i)}(t)$ along each dimension $1 \leq i \leq d$ (boundary effect should be considered; details are omitted due to space limit). Then, we retrieve all the pFM sketches in these bins combine them together into one pFM sketch, and finally estimate the score bound, $E_{iDDR}(t)$, in iDDR(t). The expected score in DDR(t) can be given by C- $E_{iDDR}(t)$, where C is a constant defined as the summation of appearance probabilities for all instances in the database $\mathcal{D}$.

Next, one remaining issue to be addressed is how to combine two pFM-sketches into one. We give the lemma below.

LEMMA 5.1. *Given two pFM sketches pFM$_1$ and pFM$_2$, we can combine them into one pFM sketch, pFM$_0$, by letting:*

$$pFM_0[j] = 1 - (1 - pFM_1[j]) \cdot (1 - pFM_2[j]), \tag{8}$$

for all $1 \leq j \leq logM$.

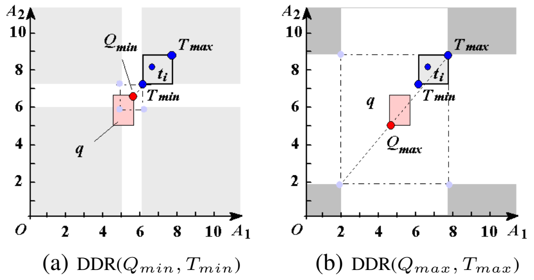
Proof Sketch. In [8], we have pFM[j] $= 1 - \Pi_{\forall t_i}(1 - t_i.p)$. Thus, by applying this formula, the correctness of Eq. (8) is obvious. □

In summary, by approximating the lower/upper bounds of scores with pFM sketches, we can avoid the costly computation with pairwise tree nodes, which has the time complexity $O(|\mathcal{H}| \cdot |N|)$, where $|\mathcal{H}|$ and $|N|$ are the numbers of entries in the heap $\mathcal{H}$ and node N, respectively. In particular, in procedure PTD_Processing, the places that need to compute bounds are replaced by our pFM sketch estimation with only $O(1)$ cost, with lines 4-8, 17-18, and 21-22. Further, the costly sentences, lines 15 and 28, can be removed, since the score bounds can be estimated directly from pFM sketch and we do not need to maintain the score bounds by checking the dynamic dominance relationship for each nodes in the heap.

6. PTD QUERIES WITH UNCERTAIN QUERY OBJECT

Up to now, we always assume that the query object for PTD is a precise point. In this section, we consider the problem where the PTD query object is also an uncertain object q containing instances q_1, q_2, ..., and $q_{|q|}$. In this case, we define the score of an uncertain object as the expected score with different query instances q_i. Formally, $score(t) = \sum_{i=1}^{|q|} q_i.p \cdot score(q_i, t)$, where $q_i.p$ is the appearance probability of query instance q_i, and $score(q_i, t)$ is the score given by Eq. (2) considering q_i as static query point.

In order to answer PTD queries with uncertain query object q, our main concern is on how to obtain the upper and lower bounds of score, denoted as $score(q_i, t)$, for all $q_i \in q$. Once the score lower bounds are available, we can easily apply the pruning condition given in Lemma 4.4 to prune the PTD search space and retrieve PTD candidates.



(a) DDR(Q_{min}, T_{min}) (b) DDR(Q_{max}, T_{max})

Figure 9: Bounding Scores for PTD with Uncertain Query Object

In the sequel, we use a 2D example shown in Fig. 9 to illustrate how to bound the scores in the case of uncertain query object. Without loss of generality, assume instances of query object q can be bounded by an MBR. As illustrated in Fig. 9(a), given an uncertain object t, let T_{min} and Q_{min} be the two closest points in MBRs of uncertain objects t and q, respectively. The shaded regions in Fig. 9(a) correspond to (conservative) dynamic dominating regions between uncertain query object q and uncertain object t, denoted as DDR(Q_{min}, T_{min}), assuming query instance q_i resides at Q_{min}. Interestingly, we observe that the summed appearance probability for all the instances falling into the shaded regions is exactly an upper bound of $score(q_i, t)$. This is because, the union of regions DDR(q_i, T_{min}) for different locations of query instance q_i is exactly the shaded region.

Similarly, as illustrated in Fig. 9(b), let T_{max} and Q_{max} be two farthest points in MBRs of uncertain objects t and q, respectively. The shaded regions in the figure correspond to dynamic dominating regions DDR(Q_{max}, T_{max}), assuming query object locates at the position of Q_{max}. Similarly, we find that the intersection of DDR(q_i, T_{max}) for all possible positions of q is exactly the shaded regions, which indicates that the summed appearance probability for all the instances falling into the shaded regions would be a lower bound of score $score(q_i, t)$.

Based on the observation in the example above, we can calculate the lower/upper bounds for scores and then use them to effectively filter out false alarms of the PTD query. Finally, the remaining candidates are refined by computing their actual scores and the ones with k highest scores are returned.

7. EXPERIMENTAL EVALUATION

In this section, we demonstrate the efficiency and effectiveness of our proposed approaches to answer *probabilistic top-k dominating* (PTD) queries. Specifically, we test the performance of PTD queries over both real and synthetic data sets with different parameter settings. For synthetic data sets, in order to generate an uncertain object t for a d-dimensional uncertain database, we first determine a region (without loss of generality, we consider it as a hypersphere) centered at location C_t and with proximity radius $r_t \in [r_{min}, r_{max}]$ in a data space $[0, 100]^d$, and then we randomly generate instances t_i of t within this region. Here, we denote as lU (lS) the data sets having center *location* C_t of *Uniform* (*Skew* with skewness 0.8) distribution; moreover, denote as rU (rG) the data sets containing uncertain objects with *radius* r_t of *Uniform* (*Gaussian* with mean $\frac{r_{min}+r_{max}}{2}$ and variance $\frac{r_{max}-r_{min}}{5}$) distribution. Thus, we can obtain 4 types of data sets, namely $lUrU$, $lUrG$, $lSrU$, and $lSrG$. For the real data set, we use 300K 3D sensory data (including attributes temperature, humidity, and light) collected from 54 sensors deployed in Intel lab, which are available at [*http://db.csail.mit.edu/labdata/labdata.html*]. Similar to synthetic data, we generate uncertainty regions for each data p with radius $r_p \in [r_{min}, r_{max}]$ following Uniform or Gaussian distribution and obtain $sensor\_rU$ and $sensor\_rG$ data sets, respectively. Note that, in real applications, r_{min} and r_{max} can be provided by the uncertainty model of data. For data sets with other data distributions or parameters (like skewness, mean, or variance), the experimental results are similar, and we would not present all of them here due to space limit. We construct an aggregate R-tree (aR-tree) [15] for each tested data set above, where the page size is set to $4K$ and the number of groups for each uncertain object is about 2-4 in our experiments.

In order to evaluate the PTD query, we produce 100 query points (or query regions for the dynamic PTD with uncertain query object), which follows the same distribution as center locations C_t in the data set. To our best knowledge, no previous work has studied the problem of the PTD query in the context of uncertain databases. In our experiments, we compare our proposed PTD query procedure, PTD_Processing (in Fig. 7), with the baseline algorithm, PTD_Baseline (in Fig. 6). Furthermore, we also investigate the performance of our PTD query processing in the case where query object is also uncertain (discussed in Section 6), as well as the approximate PTD query processing (discussed in Section 5). For brevity, we denote the baseline algorithm, PTD method, approximate PTD, and PTD with uncertain query object as Baseline, PTD, A-PTD, and UQ-PTD, respectively.

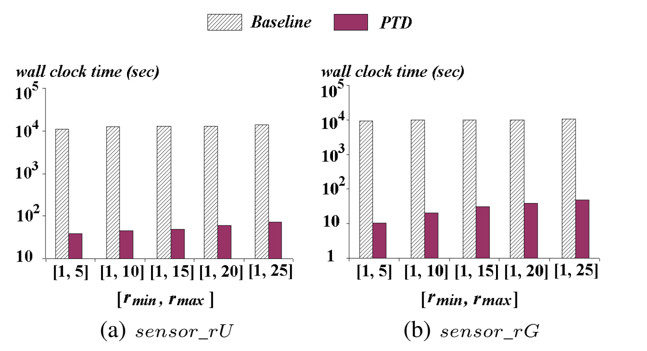
We measure the performance of our PTD query procedure in terms of *wall clock time*, which takes into account (sums up) both CPU time and I/O cost of the query processing. In particular, we incorporate the cost of each page access into the wall clock time by penalizing $10ms$ (i.e. 0.01 second) [30, 31, 17]. We conduct all the experiments on a Pentium IV 3.2GHz PC with 1G memory, and the reported experimental results are the average of 100 queries.

7.1 PTD Queries with Precise Query Point

In the first set of experiments, we evaluate the PTD query performance with precise query point over both real and synthetic data sets, by varying parameters (e.g. radius range $[r_{min}, r_{max}]$, parameter k, dimensionality d, and data size N).

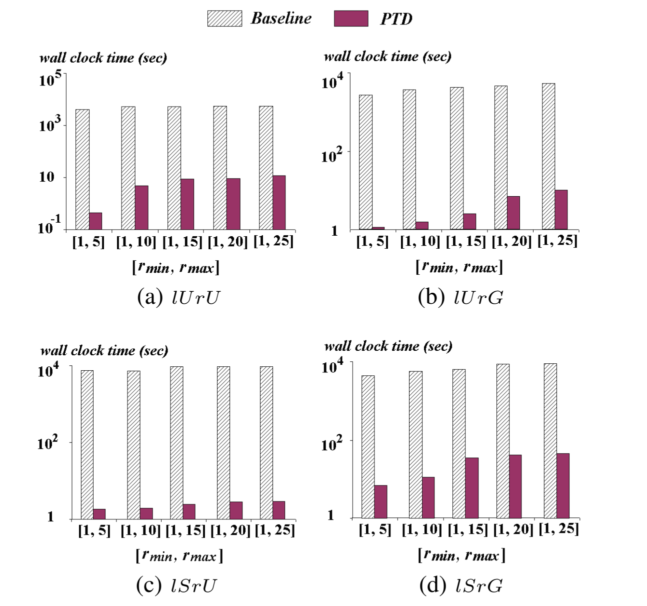
7.1.1 *Performance vs. Radius Range* $[r_{min}, r_{max}]$

Fig. 10 and Fig. 11 illustrate the experimental results of PTD query processing with different radius range $[r_{min}, r_{max}]$, over real and synthetic data sets, respectively. In particular, we vary $[r_{min}, r_{max}]$ with intervals $[1, 5]$, $[1, 10]$, $[1, 15]$, $[1, 20]$, and $[1, 25]$,



(a) $sensor\_rU$ (b) $sensor\_rG$

Figure 10: Performance vs. Radius Range $[r_{min}, r_{max}]$ (Real Data)



(a) $lUrU$ (b) $lUrG$

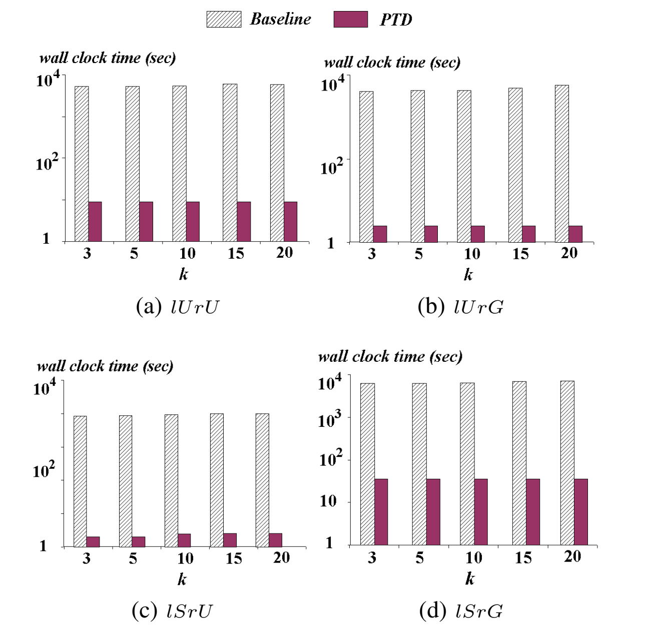
(c) $lSrU$ (d) $lSrG$

Figure 11: Performance vs. Radius Range $[r_{min}, r_{max}]$ (Synthetic Data)

where other parameters of real/synthetic data sets are fixed, that is, parameter $k = 10$, dimensionality $d = 3$, and data size $N = 300K$. Note that, when the uncertainty size $(r_{max} - r_{min})$ of objects becomes large, the overlap of score intervals is expected to be heavy (i.e., lower/upper score bounds are loose), which indicates that many objects are comparable to each other. Thus, higher cost is needed to obtain the PTD candidates, in terms of wall clock time, which is confirmed in figures. For all the data sets, our PTD query processing approach always shows better performance than the baseline algorithm by orders of magnitude, which confirms the effectiveness of our pruning method via score bounds, as well as the efficiency of our PTD query procedure.

7.1.2 Performance vs. Parameter k

In this subsection, we evaluate the wall clock time of the two approaches, PTD and Baseline, with different values of query parameter k, as illustrated in Fig. 12. Note that, since the trends of real data sets are similar to that of synthetic data, in this and subsequent experiments, we will only present the experimental results on synthetic data due to space limit. Specifically, we vary k from 3 to 20, where we fix radius range $[r_{min}, r_{max}] = [1, 15]$, dimensionality $d = 3$, and data size $N = 300K$. The wall clock time of both



(a) $lUrU$ (b) $lUrG$

(c) $lSrU$ (d) $lSrG$

Figure 12: Performance vs. PTD Query Parameter k

methods is not very sensitive to the k values, when k varies. Similar to the previous results, our PTD approach outperforms Baseline by orders of magnitude.

7.1.3 Performance vs. Dimensionality d

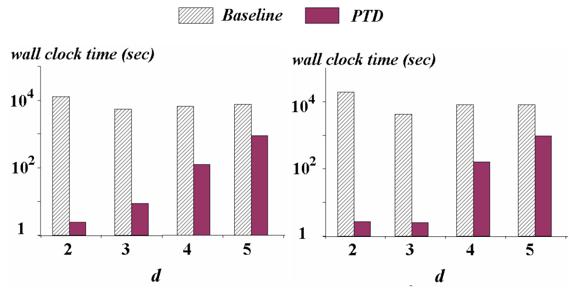
Fig. 13 studies the effect of dimensionality on the performance of our PTD query processing. In particular, we vary the dimensionality d of each type of data sets from 2 to 5, where PTD query parameter $k = 10$, radius range $[r_{min}, r_{max}] = [1, 15]$, and data size $N = 300K$. In figures, the required wall clock time for answering PTD queries increases with the increasing dimensionality d. We do not show the case where $d > 5$, but the performance is expected to degrade, since the query efficiency over the multidimensional index (including aR-tree) usually degrades with the increase of dimensionality [32]. However, we can see that our PTD method performs better than Baseline algorithm by order of magnitude.

7.1.4 Performance vs. Data Size N

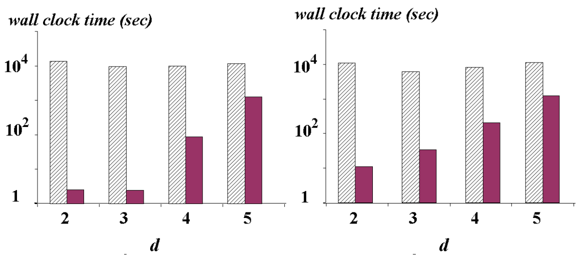
Fig. 14 conducts a scalability test of the two proposed methods with respect to the data size N. Specifically, we vary the data size N ranging from $100K$ to $500K$, and set radius range $[r_{min}, r_{max}] = [1, 15]$, parameter $k = 10$, and dimensionality $d = 3$. For all the tested data sets, the wall clock time increases when the data size N increases. This is due to the larger PTD candidate set with large data size. Furthermore, similar to previous results, wall clock time of our PTD method is by orders of magnitude better than the Baseline approach, which indicates a nice scalability of our PTD query procedure over large data set.

7.1.5 Approximate PTD Query Processing

Fig. 15 shows the efficiency and effectiveness of our approximate PTD query processing, A-PTD, over different types of data sets, compared with exact PTD approaches, PTD and Baseline, where radius range $[r_{min}, r_{max}] = [1, 15]$, parameter $k = 10$, dimensionality $d = 3$, and data size $N = 300K$. Specifically, the efficiency of A-PTD is measured by wall clock time, whereas the effectiveness of A-PTD is measured by the recall ratio, which is

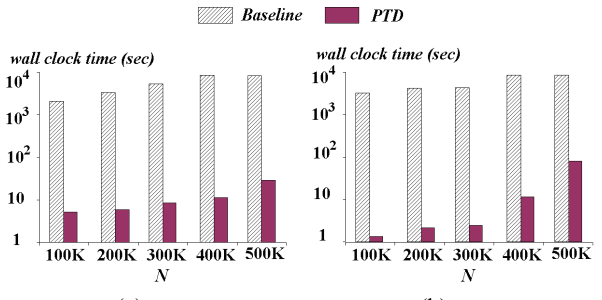


(a) $lUrU$ (b) $lUrG$

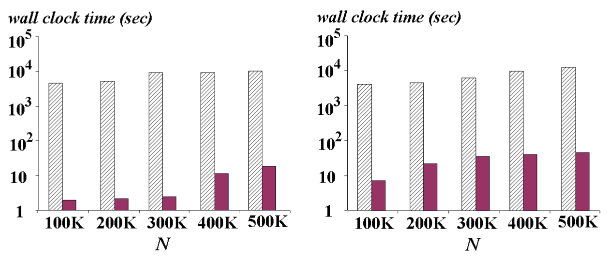


(c) $lSrU$ (d) $lSrG$

Figure 13: Performance vs. Dimensionality d



(a) $lUrU$ (b) $lUrG$



(c) $lSrU$ (d) $lSrG$

Figure 14: Performance vs. Data Size N

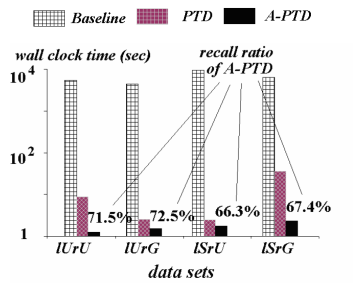
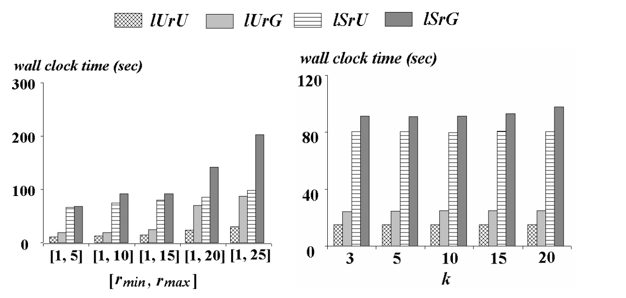
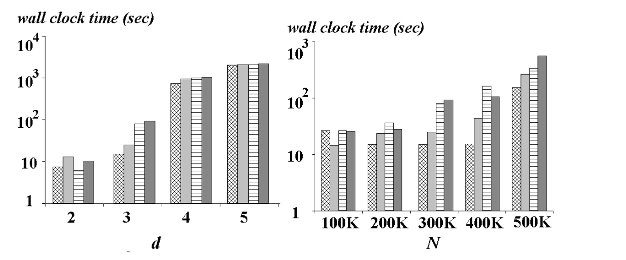


Figure 15: A-PTD Query Performance vs. Data Sets



(a) radius range $[r_{min}, r_{max}]$ (b) parameter k



(c) dimensionality d (d) data size N

Figure 16: Performance vs. Parameters $[r_{min}, r_{max}]$, k, d, and N (UQ-PTD)

defined as the number of actual PTD answers returned by A-PTD divided by the actual number of answers. In figures, the numbers over columns indicate the recall ratio of A-PTD, which is around 70%. However, A-PTD trades accuracy for efficiency. As confirmed by figures, the wall clock time of A-PTD is much smaller than both PTD and Baseline.

7.2 PTD Queries with Uncertain Query Object

Next, we consider the PTD query processing with uncertain query object. In particular, we generate the region for uncertain query object by first selecting a random point in the data space and then expand this point to a hyperrectangle with an extent $(100 \cdot \lambda)$ along each dimension, where 100 is the data domain of the data space on each dimension. The experimental results on both real and synthetic data sets are similar, and we will only report those on synthetic data due to space limit.

7.2.1 Performance vs. Parameters $[r_{min}, r_{max}]$, k, d, and N

In this subsection, we show the experimental results with the same settings as those in Section 7.1 (i.e. PTD queries with precise query point), by varying parameters $[r_{min}, r_{max}]$, k, d, and N, where the parameter λ for uncertain query object is set to 0.1 (shown in Fig. 16). Due to the uncertain region for query object, as mentioned in Section 6, the resulting score lower/upper bounds would be loose compared with the case of precise query point (since we need to overestimate the upper bound and underestimate the lower bound). Therefore, the wall clock time in this set of experiments is higher than that in the case with precise query point. However, the trends of these experiments are similar.

7.2.2 Performance vs. Parameter λ

Fig. 17 illustrates the effect of parameter λ on the performance of PTD query with uncertain query object. Since larger λ results

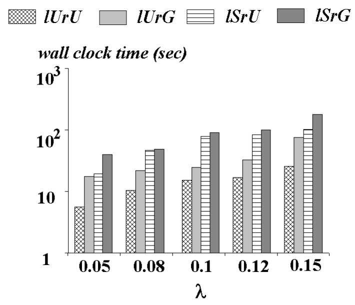


Figure 17: Performance vs. Parameter λ (UQ-PTD)

in large uncertain region of the query object, the score bounds become loose. Thus, the wall clock time of the PTD query processing increases with the increasing λ. In summary, we have demonstrated extensive experiments to show the effectiveness of the pruning methods, and verified the efficiency of our proposed PTD query processing, in terms of wall clock time.

8. CONCLUSIONS

Query processing in the uncertain database has become very hot recently due to the wide existence of uncertain data in many real applications. In this paper, we formulate and tackle the *probabilistic top-k dominating* (PTD) query in the uncertain database. In particular, we formalize the PTD query, which retrieves k uncertain objects in the database that *dynamically dominate* the highest number of instances for all possible instance combinations. Then, we propose an effective method to reduce the PTD search space, and avoid considering exponential number of instance combinations. Furthermore, approximate PTD query processing and the PTD query processing with uncertain query object are also discussed. Finally, extensive experiments have demonstrated the query performance of our proposed PTD query procedures, in terms of wall clock time.

9. ACKNOWLEDGMENT

Funding for this work was provided by Hong Kong RGC Grants under Project 611608, National Grand Fundamental Research 973 Program of China under Grant 2006CB303000, and NSFC Key Project Grant 60736013.

10. REFERENCES

[1] O. Benjelloun, A. Das Sarma, A. Y. Halevy, and J. Widom. ULDBs: Databases with uncertainty and lineage. In *Proc. 32nd Int. Conf. on Very Large Data Bases*, 2006.

[2] C. Böhm, A. Pryakhin, and M. Schubert. The Gauss-tree: efficient object identification in databases of probabilistic feature vectors. In *Proc. 22th Int. Conf. on Data Engineering*, 2006.

[3] J. Chen and R. Cheng. Efficient evaluation of imprecise location-dependent queries. In *Proc. 23th Int. Conf. on Data Engineering*, 2007.

[4] L. Chen, M. T. Ozsu, and V. Oria. Robust and fast similarity search for moving object trajectories. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, 2005.

[5] R. Cheng, D. Kalashnikov, and S. Prabhakar. Querying imprecise data in moving object environments. In *IEEE Trans. Knowl. Data Eng.*, volume 16, 2004.

[6] R. Cheng, D. V. Kalashnikov, and S. Prabhakar. Evaluating probabilistic queries over imprecise data. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, 2003.

[7] R. Cheng, S. Singh, and S. Prabhakar. U-DBMS: A database system for managing constantly-evolving data. In *Proc. 31st Int. Conf. on Very Large Data Bases*, 2005.

[8] G. Cormode and M. Garofalakis. Sketching probabilistic data streams. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, 2007.

[9] E. Dellis and B. Seeger. Efficient computation of reverse skyline queries. In *Proc. 33rd Int. Conf. on Very Large Data Bases*, 2007.

[10] A. Faradjian, J. Gehrke, and P. Bonnet. Gadt: A probability space ADT for representing and querying the physical world. In *Proc. 18th Int. Conf. on Data Engineering*, 2002.

[11] M. Hua, J. Pei, W. Zhang, and X. Lin. Ranking queries on uncertain data: A probabilistic threshold approach. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, 2008.

[12] G. Kollios, K. Yi, F. Li, and D. Srivastava. Efficient processing of top-k queries in uncertain databases. In *Proc. 24th Int. Conf. on Data Engineering*, 2008.

[13] H.-P. Kriegel, P. Kunath, M. Pfeifle, and M. Renz. Probabilistic similarity join on uncertain data. In *Proc. Int. Conf. on Database Systems for Advanced Applications*, 2006.

[14] H.-P. Kriegel, P. Kunath, and M. Renz. Probabilistic nearest-neighbor query on uncertain objects. In *Proc. Int. Conf. on Database Systems for Advanced Applications*, 2007.

[15] I. Lazaridis and S. Mehrotra. Progressive approximate aggregate queries with a multi-resolution tree structure. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, 2001.

[16] M. Li and Y. Liu. Underground coal mine monitoring with wireless sensor networks. In *ACM Transactions on Sensor Networks*, 2009.

[17] X. Lian and L. Chen. Monochromatic and bichromatic reverse skyline search over uncertain databases. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, 2008.

[18] X. Lian and L. Chen. Probabilistic ranked queries in uncertain databases. In *Proc. 11th Int. Conf. on Extending Database Technology*, 2008.

[19] V. Ljosa and A. K. Singh. APLA: indexing arbitrary probability distributions. In *Proc. 23th Int. Conf. on Data Engineering*, 2007.

[20] V. Ljosa and A. K. Singh. Top-k spatial joins of probabilistic objects. In *Proc. 24th Int. Conf. on Data Engineering*, 2008.

[21] M. F. Mokbel, C.-Y. Chow, and W. G. Aref. The new casper: query processing for location services without compromising privacy. In *Proc. 32nd Int. Conf. on Very Large Data Bases*, 2006.

[22] D. Papadias, Y. Tao, G. Fu, and B. Seeger. An optimal and progressive algorithm for skyline queries. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, 2003.

[23] J. Pei, B. Jiang, X. Lin, and Y. Yuan. Probabilistic skylines on uncertain data. In *Proc. 33rd Int. Conf. on Very Large Data Bases*, 2007.

[24] J. Pei, X. Lin, M. Hua, and W. Zhang. Efficiently answering probabilistic threshold top-k queries on uncertain data. In *Proc. 24th Int. Conf. on Data Engineering*, 2008.

[25] C. Re, N. Dalvi, and D. Suciu. Efficient top-k query evaluation on probabilistic data. In *Proc. 23th Int. Conf. on Data Engineering*, 2007.

[26] A. D. Sarma, M. Theobald, and J. Widom. Exploiting lineage for confidence computation in uncertain and probabilistic databases. In *Proc. 24th Int. Conf. on Data Engineering*, 2008.

[27] S. Singh, C. Mayfield, R. Shah, S. Prabhakar, S. Hambrusch, J. Neville, and R. Cheng. Database support for probabilistic attributes and tuples. In *Proc. 24th Int. Conf. on Data Engineering*, 2008.

[28] M. A. Soliman, I. F. Ilyas, and K. C. Chang. Top-k query processing in uncertain databases. In *Proc. 23th Int. Conf. on Data Engineering*, 2007.

[29] Y. Tao, R. Cheng, X. Xiao, W. K. Ngai, B. K., and S. Prabhakar. Indexing multi-dimensional uncertain data with arbitrary probability density functions. In *Proc. 31st Int. Conf. on Very Large Data Bases*, 2005.

[30] Y. Tao, D. Papadias, and X. Lian. Reverse kNN search in arbitrary dimensionality. In *Proc. 30th Int. Conf. on Very Large Data Bases*, 2004.

[31] Y. Tao, D. Papadias, X. Lian, and X. Xiao. Multidimensional reverse kNN search. In *The VLDB Journal*, volume 16, 2005.

[32] Y. Theodoridis and T. Sellis. A model for the prediction of R-tree performance. In *Proc. ACM SIGACT-SIGMOD Symp. on Principles of Database Systems*, 1996.

[33] W. Xue, Q. Luo, L. Chen, and Y. Liu. Contour map matching for event detection in sensor networks. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, 2006.

[34] M. L. Yiu and N. Mamoulis. Efficient processing of top-k dominating queries on multi-dimensional data. In *Proc. 33rd Int. Conf. on Very Large Data Bases*, 2007.