The VLDB Journal manuscript No.
(will be inserted by the editor)

Wenjie Zhang · Xuemin Lin · Ying Zhang · Jian Pei · Wei Wang

Threshold-based Probabilistic Top-k Dominating Queries

Received: date / Accepted: date

Abstract Recently, due to intrinsic characteristics in many underlying data sets, a number of probabilistic queries on uncertain data have been investigated. Top-k dominating queries are very important in many applications including decision making in a multidimensional space. In this paper, we study the problem of efficiently computing top-k dominating queries on uncertain data. We first formally define the problem. Then, we develop an efficient, threshold-based algorithm to compute the exact solution. To overcome some inherent computational deficiency in an exact computation, we develop an efficient randomized algorithm with an accuracy guarantee. Our extensive experiments demonstrate that both algorithms are quite efficient, while the randomized algorithm is quite scalable against data set sizes, object areas, k values, etc. The randomized algorithm is also highly accurate in practice.

Keywords Uncertain Objects · Top k · Dominating Relation

1 Introduction

Managing uncertain data has been studied ever since the eighties of the last century by the database society [1,4, 24,29]. A great deal of research attention has been drawn in the field recently as a result of many emerging, important applications related with data uncertainty, including sensor data analysis, economic decision making, market surveillance and trends predication, etc. Uncertainty is inherent in such applications due to various factors such as data randomness and incompleteness, limitation of equipment, and delay or loss in data transfer. A number of issues have been recently addressed; these include modeling uncertainty [2,36], query evaluation [10,13,14, 37], indexing [11,41], top-k queries [22,35,39,42], skyline queries [34], joins [26,27], nearest neighbor query [5, 9,27], clustering [28,30], etc.

Top-k dominating queries and skyline are shown as useful tools in decision making [6,33,40,43] to rank certain data. A *top-k dominating* query retrieves the k objects with the highest dominating ability, that is, the k objects that dominate the largest number of other objects. It is formally defined as follows [43]. Suppose that $\mathcal{X}$ is a set of d-dimensional points. For a point $x \in \mathcal{X}$, the score function is defined as the number of points dominated by x, namely, $score(x) = |\{x' \in \mathcal{X} | x \prec x'\}|$. Here, $x \prec x'$ if the coordinate value of x is not greater than that of x' at each dimension with at least one dimension at which the coordinate value of x is smaller than that of x'. $score(x)$ is a useful ranking function due to the following ordering property [43]: $\forall x,\ x' \in \mathcal{X},\ x \prec x' \Rightarrow score(x) > score(x')$. A top-$k$ dominating query retrieves the k points in $\mathcal{X}$ with the highest scores. The *skyline* operator retrieves all objects from $\mathcal{X}$ which are not dominated by other objects.

The skyline operator and top-k dominating queries rank objects in different ways: skyline ranks objects in a "defensive" way and outputs the objects which are not worse than any other objects in a given dataset, while a top-k dominating query ranks objects in an "assertive" way and provides the objects that are better than the largest number of other objects. As pointed out in [43], the benefit of using top-k dominating queries is to assimilate the advantages of top-k queries and the skyline operator. That is, the result size in a top-k dominating query is strictly controlled by k, while like skyline operators, top-k dominating queries do not require a specific ranking function and are not affected by potentially different scales at different dimensions.

Figure 1 shows the average performance of 3 popular NBA players from 3 selected games in their rookie seasons with respect to two statistics aspects, number of assists (AST) and number of points (PTS). To retain the preference of smaller values, we record $(30 - PTS)$

Wenjie Zhang, Xuemin Lin, Ying Zhang, Wei Wang
The University of New South Wale & NICTA
E-mail: {zhangw, lxue, yingz, weiw}@cse.unsw.edu.au

Jian Pei
Simon Fraser University
E-mail: jpei@cs.sfu.ca

and $(6-AST)$ in Figure 1, while the corresponding three game statistics are depicted in Figure 2. According to the aggregate information (average performance), the skyline consists of Shaquille O'neal and Elton Brand and the top-2 dominating query also returns O'neal and Brand in this example. Both dominate Brown but there is no dominating relationship between O'neal and Brand.

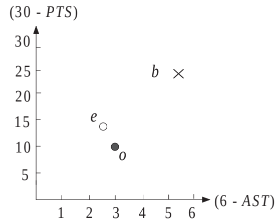


Fig. 1 Average

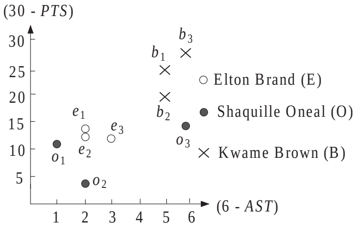


Fig. 2 NBA Players.

Motivating Example. Take NBA players as an example. NBA players may be ranked in various ways. Dominating queries provide an effective way to rank a player according to the number of other players whom this player outperforms. Using aggregates, such as AVERAGE per game, to summarize game statistics and then to count dominating relationships by the top-k dominating computation techniques in [43] is an option. While aggregates such as AVERAGE per game is useful to summarize the statistic information, they do not quite reflect the actual game-by-game performances and may be potentially affected by "outliers".

As depicted in Figure 2, O'neal's overall performance is affected by a bad outlier - o_3. Consequently, O'neal ties with Brand if we choose the top-1 dominating player according to the aggregate information in Figure 1. However, intuitively O'neal should be the winner based on the game-by-game statistics in Figure 2. The examples depicted in Figures 1 and 2 are quite representative.

We have conducted an evaluation on the fourteen 1st picks from 1991 to 2004 regarding their rookie seasons. To conduct a fair evaluation, we use the first 54 games (i.e. their rookie season games) against 3 kinds of game-by-game statistics, scores, rebounds, and assists since the year 1997 only has 54 games in the regular season. The 2nd column of Table 1 illustrates the ranks (bold number) of these players based on the number (the number in the bracket) of players dominated by them, respectively, using the average statistics per player, where Duncan is ranked first, Johnson is ranked 2nd, Webber and Brand are tied at 3rd, and O'neal is ranked 5th. Note that it is commonly believed that O'neal has the best rookie season among those players especially comparing to Brand's rookie season [1]. Thus, the top-k dominating queries against aggregates (average) may not provide right semantics for the applications where each object has multiple "instances" to occur.

Name	Ranks				
	agg	2%	5%	10%	20%
O'neal, S	**5**(4)	**5**(4)	**3**(5)	**5**(4)	**5**(4)
Johnson, L	**2**(6)	**1**(7)	**2**(6)	**1**(8)	**1**(10)
Duncan, T	**1**(7)	**1**(7)	**1**(7)	**2**(7)	**2**(7)
Webber, C	**3**(5)	**3**(5)	**3**(5)	**3**(5)	**3**(5)
Brand, E	**3**(5)	**3**(5)	**3**(5)	**3**(5)	**3**(5)
James, L	**6**(2)	**6**(2)	**6**(2)	**6**(2)	**6**(2)
Robinson, G	**10**(1)	**10**(1)	**10**(1)	**10**(1)	**10**(1)
Smith, J	**6**(2)	**6**(2)	**6**(2)	**6**(2)	**6**(2)
Iverson, A	**10**(1)	**10**(1)	**10**(1)	**10**(1)	**10**(1)
Ming, Y	**6**(2)	**6**(2)	**6**(2)	**6**(2)	**6**(2)
Howard, D	**6**(2)	**6**(2)	**6**(2)	**6**(2)	**6**(2)
Martin, K	**10**(1)	**10**(1)	**10**(1)	**10**(1)	**10**(1)
Olowokandi, M	**13**(0)	**13**(0)	**13**(0)	**13**(0)	**13**(0)
Brown, K	**13**(0)	**13**(0)	**13**(0)	**13**(0)	**13**(0)

Table 1 Ranks of NBA 1st Picks after removing outliers

Conducting an aggregate (e.g. average) after removing outliers and then applying the top-k dominating computation technique in [43] is a possible paradigm. To verify the affect of such a paradigm, we conduct the experiment on the above rookie data. We first employ one of the most popular clustering algorithms, DBSCAN [17], to remove 2%, 5%, 10% and 20% of instances as outliers from each player by choosing the distance and density parameters. Then, we calculate the average performance over remaining data for each player and then do the domination counting against the average performance. The result is depicted in Table 1 where $x\%$ for $x = 2, 5, 10, 20$ means $x\%$ of outliers have been removed. Table 1 shows that removing outliers does not quite affect the above rankings; this is because that there are bad outliers and good outliers. Therefore, the paradigm of removing outliers and then applying the top-k dominating computation may suffer from the following issues.

- The actual distributions of multiple instances are not addressed.
- Since 'bad" and "good" performance outliers have different affects, the contributions of "outliers" are not evaluated.

Probabilistic Dominating Queries. To address the applications where an object has multiple instances (e.g. game statistics of a NBA player), in this paper we develop a probabilistic model to measure the dominating ability of each object. Unlike conventional dominating queries, from probabilistic point of view each object could dominate any number of objects even with a very small probability (say 0, or close to 0). Therefore, we use the probability q by which an object dominates *at least* l objects to measure the dominating ability; that is, the dominating ability of an object U is measured by two parameters (q, l). Generally, the larger l, the smaller q. Consequently, there are two ways to model a probabilistic dominating query.

[1] See wikipedia and also `http://armchairgm.wikia.com/Top_No._1_Overall_NBA_Draft_Picks`

Name	Ranks									
	agg	0.9	0.8	0.7	0.6	0.5	0.4	0.3	0.2	0.1
O'neal, S	**5** (4)	**1** (1)	**1** (2)	**1** (2)	**1** (3)	**1** (4)	**1** (5)	**1** (5)	**1** (6)	**3** (7)
Johnson, L	**2** (6)	**2** (0)	**2** (1)	**1** (2)	**1** (3)	**1** (4)	**1** (5)	**1** (5)	**1** (6)	**1** (8)
Duncan, T	**1** (7)	**2** (0)	**2** (1)	**1** (2)	**1** (3)	**3** (3)	**3** (4)	**1** (5)	**1** (6)	**1** (8)
Webber, C	**3** (5)	**2** (0)	**2** (1)	**4** (1)	**4** (2)	**3** (3)	**3** (4)	**4** (4)	**4** (5)	**3** (7)
Brand, E	**3** (5)	**2** (0)	**5** (0)	**4** (1)	**4** (2)	**5** (2)	**5** (3)	**5** (3)	**5** (4)	**5** (6)
James, L	**6** (2)	**2** (0)	**5** (0)	**6** (0)	**6** (1)	**6** (1)	**6** (2)	**5** (3)	**5** (4)	**6** (5)
Robinson, G	**10** (1)	**2** (0)	**5** (0)	**6** (0)	**6** (1)	**6** (1)	**6** (2)	**7** (2)	**7** (3)	**8** (4)
Smith, J	**6** (2)	**2** (0)	**5** (0)	**6** (0)	**8** (0)	**6** (1)	**6** (2)	**7** (2)	**7** (3)	**8** (4)
Iverson, A	**10** (1)	**2** (0)	**5** (0)	**6** (0)	**8** (0)	**6** (1)	**9** (1)	**7** (2)	**7** (3)	**8** (4)
Ming, Y	**6** (2)	**2** (0)	**5** (0)	**6** (0)	**8** (0)	**6** (1)	**9** (1)	**7** (2)	**7** (3)	**6** (5)
Howard, D	**6** (2)	**2** (0)	**5** (0)	**6** (0)	**8** (0)	**11** (0)	**9** (1)	**11** (1)	**11** (2)	**12** (2)
Martin, K	**10** (1)	**2** (0)	**5** (0)	**6** (0)	**8** (0)	**11** (0)	**9** (1)	**11** (1)	**11** (2)	**11** (3)
Olowokandi, M	**13** (0)	**2** (0)	**5** (0)	**6** (0)	**8** (0)	**11** (0)	**13** (0)	**13** (0)	**13** (1)	**13** (1)
Brown, K	**13** (0)	**2** (0)	**5** (0)	**6** (0)	**8** (0)	**11** (0)	**13** (0)	**13** (0)	**14** (0)	**14** (0)

Table 2 Ranks of NBA 1st Picks

1. Given a probability threshold q, for each object U compute the maximum l such that U dominates at least l other objects with probability not smaller than q.
2. Given a threshold l, for each object U compute the maximum q such that U dominates at least l objects with probability not smaller than q.

In the second model, q could be very small. To control the value of q, in this paper we focus on the first model. Nevertheless our techniques can be immediately applied to the second model; we will discuss this in Section 7. In Table 2, we show the ranking results according to the 1st model where we assign each game statistic by the same occurrence probability. The 3rd to 11th columns show the ranks based on different probability thresholds, respectively. For example, in the column headed by the threshold 0.5, O'neal dominates at least 4 (the number in bracket) other players with at least the probability 0.5; thus he is ranked 1st (bold number). Clearly, O'neal is the winner against each of those probability thresholds except when the probability threshold is 0.1 (worse than Duncan and Johnson). The probabilistic rankings catch the common perception better. It is also interesting to note that Duncan dominates 7 players with a probability between 0.1 to 0.2, while according to the average (aggregate) game statistics Duncan actually dominates 7 players. Clearly, the domination counting regarding a small q is largely biased towards to "good" outliers. On the other hand, the phenomenon for a very large q (close to 1) is not very meaningful since q is always 1 if $l = 0$. The most interesting part of q is towards the middle of $(0, 1]$; these values will show the dominating ability among majority instances of objects, respectively. In addition, our probabilistic model provides a tool for us to "drill down" information against different probabilistic threshold values to provide the breakdown information like that in Table 2.

Probabilistic Top-k Dominating Queries. In this paper, we will study the problem of retrieval of k objects with the maximum values of l for a given probability threshold q (i.e., based on the 1st model). We will adopt the assumption that the probability distribution of the object is independent to each other due to the following reasons. Firstly, it is a common model currently adopted in probabilistic query processing. Secondly, handling dependence among a large number of objects is not only complex but also expensive, while applications with the assumption of independent distributions exist. For example, regarding the above example there is no reason to believe a dependence among those 1st picks' performance in their rookie season across different years, given the game rules are the same and the other players in each year have similar talents. Similarly, we could also evaluate the top-k all-round gymnastics players with the same gender by treating each competition record as an instance of a player where each competition record consists of scores for each individual programs. In male competitions, scores from Vault, Floor, Parallel bars, Rings, Pommel horse and Horizontal bar are recorded in each competition, respectively, as a 6-dimensional instance. While the performances of each female player in four programs (Vault, Floor, Uneven bars and Balance beam) are recorded per each competition. Clearly, the performances of the players are independent with each others.

Contributions. As shown above, dominating relationships among uncertain objects are quite complex and probabilistic distribution dependent. Moreover, due to the nature of uncertain data and dominating queries, an expensive computation will be involved in exactly computing "probabilistic" scores of objects; consequently, it is too expensive to compute such scores for all objects.

In this paper, we investigate the problem of efficiently computing the top-k dominating queries against uncertain objects where object PDFs are not available; that is, we deal with *discrete cases*. To the best of our knowledge, this is the first work addressing the top-k dominating query over uncertain data. Our contributions may be summarized as follows.

- We formally define a top-k dominating query on uncertain data with a given probability threshold imposed to support different confidence requirements.
- An efficient, threshold-based exact algorithm is proposed to take an advantage of the threshold-based paradigm [18]. Based on a novel application of laws of large numbers [20] and mathematic characterizations, a set of novel, effective pruning techniques have been proposed to pursue efficiency.
- We develop an efficient randomized algorithm with an accuracy guarantee. Novel processing techniques and data structures are developed in our randomized techniques.

An extensive experimental study over synthetic and real data shows that our exact algorithm performs well, while our randomized algorithm is not only highly accurate and more efficient than the exact algorithm but also quite scalable against data sizes, object uncertain areas, k values, etc.

Organization of the paper. The rest of this paper is organized as follows. Section 2 formally defines probabilistic top-k dominating queries and presents preliminaries. Section 3 briefly outlines the framework of our exact and randomized algorithms. In Section 4, we present our exact algorithm. Following the framework of exact algorithm, a novel randomized algorithm is presented in Section 5. Our experiment results are reported in Section 6. This is followed by the discussions regarding the model where a threshold of a domination counting is given and the general cases where probabilistic distributions may be correlated. The related work is presented in Section 8. We conclude our paper in section 9.

2 Background Information

We first model the problem and then, present the preliminaries of the paper. For reference, notations used in this paper are summarized in Table 3.

2.1 Problem Statement.

Our investigation in the paper will focus on discrete cases. An *uncertain* object U is represented by a set of *instances* such that each instance $u \in U$ is a point in a d-dimensional numeric space $D = \{D_1, ..., D_d\}$ with the probability $P(u)$ to occur where $0 < P(u) \leq 1$ and $\sum_{u \in U} P(u) = 1$.

Given a set of uncertain objects $\mathcal{U} = \{U_1, \cdots, U_n\}$, a *possible world* $W = \{u_1, \cdots, u_n\}$ is a set of n instances - one instance per uncertain object. The probability of W to appear is $P(W) = \prod_{i=1}^{n} P(u_i)$. Let Ω be the set of all possible worlds; that is, $\Omega = U_1 \times U_2 \cdots \times U_n$. Then, $\sum_{W \in \Omega} P(W) = 1$.

Notation	Definition
$\mathcal{U}$	set of uncertain objects
U, V	uncertain objects
u, v	instances of uncertain objects
E	entry in an aR-tree of objects, instances, and samples
$MBB_U(MBB_E)$	minimum bounding box of $U(E)$
$\mu_U(\mu_E)$	upper-right corner of $MBB_U(MBB_E)$
$\imath_U(\imath_E)$	lower-left corner of $MBB_U(MBB_E)$
$P(\tau)$	probability of τ to occur
q	probability threshold of a query
$pscore(\upsilon)$	probabilistic score of υ ($\upsilon = U$ or u)
$pscore^+$	upper bound of $pscore$
$P_{=l}(U)(P_{=l}(u))$	probabilities to dominate l objects
$P_{\geq l}(U)(P_{\geq l}(u))$	probabilities to dominate $\geq l$ objects
γ_k	minimum $pscore$ of the top-k objects
λ_k	minimum $pscore$ of the current top-k objects
$P(u \prec V)$	probability of u dominating V
Ω	set of possible worlds
$L_i(U)$	ith level entries in an aR-tree of U
P^{upper}	upper bound of probability P
$PD(U)$ $(FD(U))$	set of objects partially (fully) dominated by U
$PD(E)$ $(FD(E))$	set of entries (objects) partially (fully) dominated by E
$\overrightarrow{\mathcal{U}}$	partially ordered list of uncertain objects

Table 3 The summary of Notations.

$\Omega_{\ell,U}$ denotes the set of possible worlds in each of which the instance $u \in U$ dominates exactly ℓ other instances. Clearly, the probability $P_{=\ell}(U)$ of U dominating exactly ℓ objects is:

$$P_{=\ell}(U) = \sum_{W \in \Omega_{\ell,U}} P(W). \tag{1}$$

Example 1 Regarding the example in Figure 2, we treat every player as an uncertain object and each game statistic as an instance of the object. Unless specified otherwise, the occurring probability of each instance is $1/3$. $\Omega_{1,O} = \{\{o_3, e_1, b_3\}, \{o_3, e_2, b_3\}, \{o_3, e_3, b_3\}, \}$. Dominating probabilities of each player are as follows.
$P_{=0}(O) = 2/9$, $P_{=1}(O) = 3/27$, $P_{=2}(O) = 2/3$;
$P_{=0}(E) = 0$, $P_{=1}(E) = 2/3$, $P_{=2}(E) = 1/3$;
$P_{=0}(B) = 1$, $P_{=1}(B) = 0$, $P_{=2}(B) = 0$.

As mentioned earlier, unlike dominating queries on certain objects, an uncertain object can dominate any number of objects with some probability. Nevertheless, such dominating probabilities could be very small (even zero); results with a small probability to occur are not very interesting. To resolve this, in our problem definition we enforce a probability threshold, and we model probabilistic dominating queries in an accumulative way; that is, we look for the objects that dominate at least ℓ other objects with at least probability (confidence) q. We assign a probabilistic score, $pscore_q(U)$, to each uncertain object U as follows.

Let $P_{\geq \ell}(U)$ denote the probability of U dominating at least ℓ other objects. Clearly,

$$P_{\geq \ell}(U) = \sum_{i=\ell}^{n} P_{=i}(U). \tag{2}$$

Definition 1 ($pscore_q$) $pscore_q(U)$ is the maximum ℓ such that $P_{\geq \ell}(U) \geq q$.

Note that for notation simplification, $pscore_q$ is hereafter abbreviated to $pscore$ whenever there is no ambiguity.

Definition 2 (PtopkQ) Given a probability threshold q, an integer k, and a set $\mathcal{U}$ of uncertain objects, PtopkQ retrieves the k objects with the highest $pscore$ values. Ties are broken arbitrarily.

Example 2 Regarding the example in Figure 2 when $q = 2/3$, $pscore_q(O) = 2$, $pscore_q(E) = 1$ and $pscore_q(B) = 0$; that is, O'neal is the top dominating player.

In this paper, we will develop efficient exact algorithms as well as efficient and effective randomized algorithms to compute PtopkQ.

2.2 Preliminaries

Dominating Relationships. A pair U, V of uncertain objects may have three relationships as illustrated in Figure 5.

Let MBB_U denote the minimum bounding box of the instances of an uncertain object U. μ_U and $\imath_U$ are the upper-right and lower-left corner of MBB_U, respectively. An object U *fully dominates* another object V if $\mu_U \prec \imath_V$, and *partially dominates* V if $\imath_U \prec \mu_V$ but $\mu_U \not\prec \imath_V$, including $\mu_U = \imath_V$. Otherwise, U does *not dominate* V. As depicted in Figure 5, U does not dominate V_1, partially dominates V_3, and fully dominates V_2.

Centroid. The dominating ability of an object is determined by the distribution of its instances and its relationships to the distributions of instances of other objects. The centroid $\omega(U)$ of instances will be used in our algorithms to approximately represent the distribution of instances. Formally, $\omega(U) = \sum_{u \in U} P(u) \times u$.

aR-tree. An aggregate R-tree (aR-tree) [31] is an extension of R-tree [21] where each entry keeps the number of objects contained. Figure 3 illustrates 9 data points indexed by an aR-tree, bounded by 3 MBBs at the leaf level.

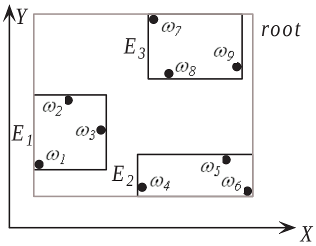


Fig. 3 Certain Data

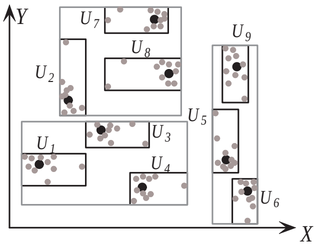


Fig. 4 Uncertain Data

Top-k Dominating Query on Certain Data. Given a k and a set of points, the CBT (cost-based traversal) algorithm in [43] selects the k points with the highest dominating $score$ values. Recall that $score(x)$ of a point is the number of other points dominated by x. Below we briefly introduce CBT, to be used as a black-box in the preprocessing of our algorithm in Section 4.1.

In CBT, an aR-tree is used. The algorithm CBT traverses the aR-tree level by level to calculate a lower bound $score^-(E)$ and an upper-bound $score^+(E)$ of the number of points dominated by a point in an entry E of aR-tree. An entry E is *pruned* if $score^+(E)$ is not greater than the current kth largest $score^-$ and the points in E are the solution if $score^-(E)$ is not smaller than the current kth largest $score^+$; otherwise E will be drilled down to the lower level; these are conducted by taking the consideration of the number of points in intermediate entries of aR-tree. In our preprocessing, we will make use of the entries that are either pruned or stayed in the job queue when the algorithm CBT terminates. Clearly, these entries are disjoint and cover all points. Note that these entries can be either points or intermediate entries. Below is an example.

Example 3 Regarding the example in Figure 3, if k=2, the algorithm terminates with the following entries in the job queue:

$$\{\omega_1.[6,6], \omega_4.[4,4], \omega_2.[3,3], \omega_3.[3,3], \omega_5.[0,3], \omega_6.[0,0]\},$$

while $E_3.[0,2]$ is pruned. In each entry representation, the left-end in the bracket is $score^-$ and the right-end is $score^+$. Top-2 dominating results retrieved is thus ω_1 and ω_4.

Efficient Computation of Dominating Probabilities. $P(u \prec V)$ denotes the probability that an instance $u \in U$ dominates an uncertain object V; that is, the sum of the probabilities of the instances in V which are dominated by u. For instance regarding the example in Figure 2, $P(o_3 \prec B) = 1/3$ (recall each instance takes the probability 1/3 to occur).

Let $P_{=\ell}(u)$ denote the probability that an instance $u \in U$ dominates ℓ other objects. $\Omega_\ell^{\mathcal{U}-U}(u)$ denotes the subset of possible worlds in $\prod_{V \in \mathcal{U}-U} V$ in each of which u dominates exactly ℓ instances. Clearly, $P_{=\ell}(u) = \sum_{W \in \Omega_\ell^{\mathcal{U}-U}(u)} P(W)$.

Example 4 Regarding Figure 2, let $U = E$, and $\ell = 2$. Then, $\Omega_\ell^{\mathcal{U}-U}(e_1) = \{(o_3, b_1), (o_3, b_2), (o_3, b_3)\}$. $P_{=2}(\{e_1\}) = 1/3{*}1/3 + 1/3 * 1/3 + 1/3 * 1/3 = 1/3$.

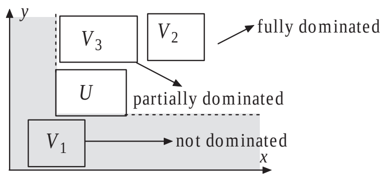


Fig. 5 Dominating Relationships.

We can immediately verify that (1) can be re-written as follows.

$$P_{=\ell}(U) = \sum_{u \in U} P(u) P_{=\ell}(u). \tag{3}$$

Based on (2) and (3), $P_{\geq \ell}$ can be rewritten as:

$$P_{\geq \ell}(U) = \sum_{u \in U} (P(u) \cdot (1 - \sum_{i=0}^{\ell-1} P_{=i}(u))). \tag{4}$$

According to Equation 3, a key to compute $pscore(U)$ and $P_{=\ell}(U)$ is to efficiently compute $P_{=\ell}(u)$ for each $u \in U$. Suppose that we already computed $P(u \prec V)$ for every $V \in \mathcal{U} - U$. The dynamic programming based techniques in [42] can be immediately used to compute $P_{=\ell}(u)$ ($\forall u \in U$) with time complexity $O(|\mathcal{U} - U| \times \ell)$ for a given u. Assume that uncertain objects in $\mathcal{U} - U$ are represented by $\{V_i : 1 \leq i \leq n-1\}$; note that objects in $\mathcal{U} - U$ can follow any order. We use p_i to denote $P(u \prec V_i)$. For $0 \leq n_1 \leq n_2$, let P_{n_1,n_2} denote the probability that u exactly dominates n_1 objects from the first n_2 objects of $\mathcal{U} - U$. It is shown [42] that $\forall 0 \leq i \leq j$ ($P_{0,0} = 1$),

$$\begin{aligned} P_{0,j} &= P_{0,j-1} \cdot (1 - p_j) = \Pi_{k=1}^{j} (1 - p_k) \\ P_{i,j} &= p_i \cdot P_{i-1,j-1} + (1 - p_i) \cdot P_{i,j-1} \end{aligned} \tag{5}$$

Let $FD(u)$ denote the set of objects fully dominated by u; that is, $\forall U \in FD(u)$, $P(u \prec U) = 1$. Let $PD(u)$ denote the set of objects partially dominated by u. It can be immediately verified that:

$$P_{=\ell}(u) = P_{=(\ell - |FD(u)|)}|_{PD(u)}(u \prec PD(u)). \tag{6}$$

Here, $P_{=(\ell-|FD(u)|)}(u)|_{PD(u)}$ denotes the probability that u dominates exactly $(\ell - |FD(u)|)$ objects in $PD(u)$ since the probability for u to dominate each object in $FD(u)$ is always 1. Consequently, in our techniques for each u we apply the dynamic programming technique on objects in $PD(u)$ only. Whenever there is no ambiguity, $P_{=l}(u)$ (or $P_{\geq l}(u)$), thereafter, always refers to the dominating probability against $PD(u)$ and $l = \ell - |FD(u)|$ where $\ell > |FD(u)|$ since all objects in $FD(u)$ are dominated by u with the probability 1.

Example 5 Regarding Figure 2, $p_1 = P(e_1 \prec O) = 1/3$, and $p_2 = P(e_1 \prec B) = 1$. By the above dynamic programming based algorithm, $P_{0,1} = 1 - p_1 = 2/3$, $P_{0,2} = P_{0,1} * (1 - p_2) = 0$, $P_{1,1} = p_1 = 1/3$, $P_{1,2} = P_{0,1} * p_2 + P_{1,1} * (1 - p_2) = 2/3$. Thus, $P_{=1}(e_1) = 2/3$.

2.3 Challenges

1. A solution to PtopkQ highly depends on the probability distribution of objects even if spatial locations of the instances are fixed.

   Example 6 Regarding the example of Figure 2, if we fix the spatial locations of these 9 instances but change the probability of instances from O'neal as follows, $P(o_1) = 1/6$, $P(o_2) = 1/6$ and $P(o_3) = 2/3$. The occurrence probability of every other instance remains 1/3. Then, we can immediately verify that regarding $q = 2/3$, $pscore(O) = 1$, $pscore(E) = 2$ and $pscore(B) = 0$. In this case, the top-1 dominating query retrieves Brand instead of O'neal (the top-1 result in Example 2).

2. Techniques developed solely on aggregate information cannot provide a correct solution to PtopkQ. It should be very straightforward to construct two different scenarios with the same aggregate information as depicted in Figure 1 such that they lead to different solutions towards PtopkQ.
3. The computation of $pscore(U)$ for an uncertain object U takes $O(|U| \times pscore(U) \times |PD(U)|)$ time as shown above. Trivially computing $pscore(U)$ for all $U \in \mathcal{U}$ and then choosing k objects with the highest $pscore$ values is computationally very expensive and slow.

3 Framework

Our exact and randomized algorithms both follow the threshold-based paradigm by using a combination of two thresholds based on q and $pscores$, respectively, to efficiently prune away objects not in PtopkQ as early as possible. Below, Algorithm 1 is an outline of the framework to be adopted in the exact and randomized algorithms. It follows three steps, pre-ordering, initial computation and final computation.

Algorithm 1 Exact Algorithm

```
Step 1: Pre-ordering. For all uncertain objects $\mathcal{U}$, generate
    an ordered list $\overrightarrow{\mathcal{U}}$ of $\mathcal{U}$.
Step 2: Initial Computation. Choose the first $k$ objects $\{U_i :$
    $1 \leq i \leq k\}$ in $\overrightarrow{\mathcal{U}}$ and compute their $pscore$ (for exact
    algorithm) or $pscore^r$ (for randomized algorithm) values.
Step 3: Final Computation. Determine the solution of
    PtopkQ in a "level-by-level" fashion.
```

Using $\overrightarrow{\mathcal{U}}$ resulted in Step 1, score values for the first k objects are computed in Step 2. Such values serve as thresholds in Step 3.

3.1 Data Structures

In the exact and randomized algorithms, we maintain an aR-tree on centroids to run CBT algorithm [43] as preprocessing (Step 1). We also maintain an *aR*-tree on the MBBs of uncertain objects to speed-up our pruning techniques at the object level.

Moreover, in the exact algorithm, for each object U, we build a local data structure, *aR*-tree, to organize its instances to efficiently support a level-by-level pruning computation in Step 3. However, the randomized algorithm indexes the sampled instances of each uncertain object using a novel data structure *gCaR-tree* for efficiency.

3.2 Monotonic Property

The following monotonic property will be effectively used to terminate our algorithm as early as possible. It immediately derives from Equation(2).

Monotonic Property: For an uncertain object U and two integers ℓ_1 and ℓ_2, if $\ell_1 \geq \ell_2$, $P_{\geq \ell_1}(U) \leq P_{\geq \ell_2}(U)$.

3.3 Efficient Level-by-level Computation

In the exact algorithm, for each uncertain object U in $\mathcal{U}$, instances in U are indexed using an aR-tree. Suppose that $E \in U$ is at the ith level of the aR-tree. Let $L_i(U)$ denote the set of entries in the ith level of local aR-tree of U. The equation (4) can be rewritten as:

$$P_{\geq l}(U) = \sum_{E \in L_i(U)} P_{\geq l}(E) \tag{7}$$

It will be too expensive to compute $P_{\geq l}(E)$ in our level-by-level computation. Instead, we use upper-bound techniques to bound $P_{\geq l}(E)$ for efficiency.

Let $(\mathcal{U} - U)_i$ denote the objects in $\mathcal{U} - U$ with the following modification regarding level i. For each object $V \in \mathcal{U} - U$ and each entry E_V at the ith level of the local aR-tree of V, we move all the instances contained by E_V to the upper-right corner μ_{E_V} of E_V. Let $\imath_E$ denote the lower-left corner of E. Let $P_{\geq \lambda}(\imath_E \prec (\mathcal{U} - U)_i)$ denote the probability that $\imath_E$ dominates at least λ objects in $(\mathcal{U} - U)_i$.

Theorem 1 $P_{\geq \lambda}(E) \leq P_{\geq \lambda}(\imath_E \prec (\mathcal{U} - U)_i) \sum_{u \in E} P(u)$.

Proof It can be immediately verified that for each possible world in the original case where an instance u from E dominates at least λ instances from different objects, its corresponding instance as modified above retains such a property. □

It is immediate that an application of the dynamic programming based algorithm in Section 2.2 leads to the time complexity $O(m_1 \times C \times \lambda)$ to compute $P_{\geq l}(E)$ where m_1 is the number of instances in E and C is the average cost to compute dominating probability between an instance and an object, while the computation of the upper-bound in Theorem 1 only takes $O(\lambda \times m_2)$ time where m_2 is the number of entries partially dominated by E. Clearly, m_2 is much smaller than C.

Example 7 In Figure 4, assume that we want to compute $P_{\geq \lambda}(U)$. Theorem 1 states that we can get an upper-bound of $P_{\geq \lambda}(U)$ at the root level of local aR-trees of objects. Let $\imath_3$ be the lower left corner of the MBB of U_3 and μ_i (for $1 \leq i \leq 9$) be the upper right corner of U_i.

Then, $(\mathcal{U} - U_3)_1 = \{\mu_i | 1 \leq i \leq 9 \ \& \ i \neq 3 \ \& \ P(\mu_i) = 1\}$. Theorem 1 states that $P_{\geq \lambda}(U) \leq P_{\geq \lambda}(\imath_3 \prec (\mathcal{U} - U_3)_1)$ since $\sum_{u \in U_3} P(u) = 1$.

4 Exact Algorithm

We present detailed techniques developed based on the framework in Section 3. The first step and the second step are quite straightforward and mainly based on the techniques in [43,42]. The third step is the most important step in Algorithm 1 to prevent as many objects as possible from an exact computation of *pscore*; novel, effective, efficient pruning techniques are developed.

4.1 Step 1: Pre-ordering Objects

Step 1 aims to generate such an access order so that the maximal possible threshold value regarding *pscore* can be reached as soon as possible. Clearly, the maximum possible threshold value regarding *pscore* should be the minimum value of the *pscore*s of the top-k objects. Nevertheless, this is infeasible to achieve without conducting an exact computation of PtopkQ. The following heuristic is developed to resolve this.

The centroid $\omega(U)$ ($\forall U \in \mathcal{U}$) is used to approximately represent the probabilistic distribution of an uncertain object U with the aim to use $score(\omega(U))$ to approximately reflect the rank of $pscore(U)$. Note that it is quite expensive to compute $score(\omega(U))$ for each object U. Instead, we apply the CBT algorithm (briefly introduced in section 2.2) to generate an approximately ordered list $\overrightarrow{\mathcal{U}}$ as follows.

In $\overrightarrow{\mathcal{U}}$, we keep the scored entries of the aR-tree of centroids, generated by CBT; that is, the entries pruned by CBT or the entries remained in the job queue once it terminates (as described in Section 2.2). Then, we sort entries in $\overrightarrow{\mathcal{U}}$ non-increasingly according to their accompanied $score^{+}$ values. When a centroid $\omega(U)$ and the intermediate entry E have the same $score^{+}$ value, we always rank $\omega(U)$ before E in $\overrightarrow{\mathcal{U}}$. Then, if two $score^{+}$ values from two centroids are the same, we always rank a centroid with the exact $score$ value higher. In other cases, entries with the same score are ranked randomly among them. Note that in an entry, each contained centroid $\omega(U)$ corresponds to the object U; we use U to replace $\omega(U)$ in $\overrightarrow{\mathcal{U}}$.

Example 8 Regarding the example in Figure 3 and Figure 4, Figure 3 shows the centroids of uncertain objects in Figure 4. As shown in Example 3 when $k = 2$,

$$\{\omega_1.[6,6], \omega_4.[4,4], \omega_2.[3,3], \omega_3.[3,3], \omega_5.[0,3], \omega_6.[0,0]\},$$

remain in job queue, while $E_3.[0,2]$ is pruned by CBT. Consider that ω_i (for $1 \leq i \leq 9$) corresponds to the uncertain object U_i. Therefore, $\overrightarrow{\mathcal{U}} = \{U_1, U_4, U_2, U_3, U_5, E_3, U_6\}$ when $k = 2$. Their $score^+$ values are 6, 4, 3, 3, 3, 2, and 0, respectively.

4.2 Step 2: Initial Computation

Our algorithm to calculate the *pscore*s for each U of the first k objects in $\overrightarrow{\mathcal{U}}$ is outlined below in Algorithm 2.

Algorithm 2 Calculate *pscore*

```
Step 2.1: Traverse the $aR$-tree of objects' MBBs to obtain
    the number of objects that $U$ fully dominates $|FD(U)|$,
    and the set $PD(U)$ of objects that $U$ partially dominates.
Step 2.2: Do a synchronous traversal [7,32] of the local $aR$-
    tree of $U$ against the local $aR$-trees of the objects in
    $PD(U)$ to calculate $P(u \prec V)$ for each $V \in PD(U)$ and
    each instance $u \in U$.
Step 2.3: Calculate the $pscore(U)$.
```

We conduct step 2.1 by window query techniques [21] by using $\imath_U$ to get all objects that U dominates (fully or partially) and then use μ_U to check the full dominance.

We conduct Step 2.2 by the well known synchronous traversal paradigms [7,32] to compute $P(u \prec V)$ ($\forall u \in U$ and $\forall V \in PD(U)$) since the synchronous traversal paradigm has been shown effective in join computation. Moreover, [43] shows that on average the synchronous traversal strategy is the most cost effective way to count the dominance relationships. Finally, our techniques can be extended to cover any traversal strategies.

Note $P_{\geq l}(U) = \sum_{u \in U} P(u) P_{\geq l}(u)$. In Step 2.3, to calculate $P_{\geq l}(U)$ we apply the dynamic programming based algorithm in Section 2.2 to calculate $P_{\geq l}(u)$ ($\forall u \in U$) **restricted to the objects in** $PD(U)$. Based on the monotonic property in Section 3.2, when $P_{\geq l}(U) \geq q$ and $P_{\geq (l+1)}(U) < q$, the computation stops and $(l + |FD(U)|)$ is the *pscore* for U. To avoid any redundant computation, we conduct the calculation in Equation (4) iteratively from $l = 0$. After the completion of calculation of $P_{\geq l}(u)$ for each $u \in U$ for the current l, we examine if $P_{\geq l}(U) \geq q$ to determine whether we should stop a further calculation of such probability. We can immediately verify that the time complexity of Step 2.3 is $O(l \times |PD(U)| \times |U|)$ for each U.

4.3 Step 3: Final Computation

The final computation is conducted by **bounding-pruning-refining**. This will be based on a threshold of *pscore* and the given confidence q. Clearly, the best available threshold of *pscore* is the minimum value, denoted by λ_k, of *pscore*s of the current top-k objects. To pursue efficiency, for each remaining U the Step 3 will be conducted level-by-level in a synchronous traversal fashion among the local aR-trees of U and the objects in $PD(U)$;[2] nevertheless, our techniques can be extended to any traversal strategies. Our algorithm for Step 3 is outlined in Algorithm 3.

Algorithm 3 Final Computation

```
$T_k :=$ {the first $k$ objects from $\overrightarrow{\mathcal{U}}$}; $\overrightarrow{\mathcal{U}} := \overrightarrow{\mathcal{U}} - T_k$;
WHILE $\overrightarrow{\mathcal{U}} \neq \emptyset$ DO
  Step 3.1 - Pruning at Object Level: Dequeue the first entry
      $E$ from $\overrightarrow{\mathcal{U}}$; Use window queries to check if objects in $E$
      can be completely pruned away - if not, then go to Step
      3.2.
  Step 3.2 - Level-by-Level Pruning: For each remaining $U$, do
      a level-by-level synchronous traversal among the local $aR$-
      tree of $U$ and the local $aR$-trees of the objects in $PD(U)$
      to conduct a level-by-level pruning.
  Step 3.3 - Compute $pscore$: For each remaining object $U$ af-
      ter Step 3.2,
      – calculate the $pscore(U)$;
      – if $pscore(U) > \lambda_k$, then replace an object $V$ in $T_k$
        with $pscore(V) = \lambda_k$ by $U$, and Update $\gamma_k$.
ENDWHILE
Return $T_k$.
```

While Steps 3.1 and 3.3 are relatively straightforward, Step 3.2 is critical in Algorithm 3; it can significantly speed-up the algorithm by avoiding as many objects as possible to enter into the expensive Step 3.3; our experiment results demonstrate that our pruning techniques can speed-up the computation by orders of magnitude. We show the basic idea of our algorithm of Step 3.2 in Example 9. Suppose that E is an entry, at the ith level, of the local aR-tree of U, let $PD(E)$ denote the set of *entries* at the ith level of the local aR-trees of other objects, which are partially dominated by E. $\#obj(PD(E))$ denotes the number of distinct objects containing the entries in $PD(E)$, while $FD(E)$ denotes the set of *objects* fully dominated by E.

Example 9 In Figure 6, the 3 local aR-trees of U_1, U_2, and U_3 have 3 levels, respectively, with one intermediate level E_j ($\forall 1 \leq j \leq 9$). Assume that $\lambda_k = 1$ and Step 3.2 is conducted against U_1.

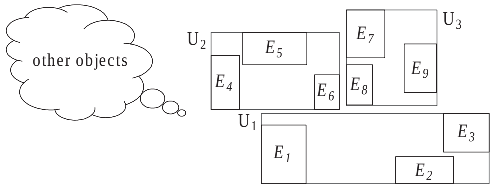


Fig. 6 Level-by-level computation.

[2] Note that if local aR-trees have different height, the one that reaches the bottom level first will stay at the bottom, while others traverse down to the lower levels.

Note that $PD(U_1) = \{U_2, U_3\}$ and $FD(U_1) = \emptyset$. As $pscore^+(U_1) = |PD(U_1)| + |FD(U_1)| \geq \lambda_k$, we expand U_1, U_2, and U_3 synchronously to the next level. The following is immediate where each E_j (for $1 \leq j \leq 9$) is at level 2.

– $PD(E_1) = \{U_2.(E_5, E_6)\}$[3] and $FD(E_1) = \{U_3\}$. Note that $\#obj(PD(E_1)) = 1$.
– $PD(E_2) = \{U_3.(E_9)\}$ and $FD(E_2) = \emptyset$. Note that $\#obj$ $(PD(E_2)) = 1$.
– $PD(E_3) = \emptyset$ and $FD(E_3) = \emptyset$.

Since $pscore^+(E_3)(\triangleq \#obj(PD(E_3)) + |FD(E_3)|) = 0$ ($< \lambda_k$), we can exclude E_3 from a further consideration. We only need to check E_1 and E_2 by the following bounding-pruning techniques to determine whether or not they need to be expanded to the next level.

The key in Step 3 is to develop efficient and effective bounding-pruning techniques for pruning purposes. They will be conducted based on the following two principles.

1. **probability-based:** Efficiently and effectively computing an upper-bound $P^{upper}_{\geq\lambda_k}(U)$ of $P_{\geq\lambda_k}(U)$ so that U can be pruned if $P^{upper}_{\geq\lambda_k}(U) \leq q$.
2. **score-based:** Efficiently and effectively computing a $pscore^+(U)$ such that U can be pruned if $pscore^+(U)$ $< \lambda_k$.

4.3.1 Efficient and Effective Bounding Techniques

In Theorem 1, for each entry E, we use $P_{\geq\lambda}(\imath_E \prec (\mathcal{U} - U)_i)$, multiplied by $\sum_{u \in E} P(u)$, as an upper bound of $P_{\geq\lambda}(E)$. This takes $O(\lambda_k \times |PD(E)|)$ time for each entry E. To further speed-up the computation, the following two upper-bounds of $P_{\geq\lambda}(\imath_E \prec (\mathcal{U} - U)_i)$ are developed; they reduce the costs from $O(\lambda_k \times |PD(E)|)$ to $O(|PD(E)|)$. This is significant when λ_k is large.

1. Chernoff-Hoeffding Bound based Upper-bound. For an uncertain object V and an instance u in another uncertain object U, we can regard the event that u dominates V as a random variable. Consequently, we can employ the probabilistic bounds to compute the upper bound of the pscore of an uncertain object, which is very time efficient. Due to the independence assumption, we apply a strong version of Chernoff-Hoeffding Bound [16] in the paper.

Chernoff-Hoeffding Bound [16]. Let X_1, X_2, X_3, ..., X_n be independent random variables with values in [0, 1], $X = \sum_{i=1}^{n} X_i$ and $\epsilon > 0$. Then,

$$P(X > (1 + \epsilon)E(X)) < \exp^{-E(X)\epsilon^2/3} \tag{8}$$

Recall $\imath_E$ is the lower-left corner of an entry E. $\imath_E$ partially dominates l objects V_1, V_2, ... V_l. Since each V_i ($1 \leq i \leq l$) is an uncertain object, the probability of $\imath_E$ dominating a V_i can be treated as the expected value of the following random variable.

$$X_{V_i} = \begin{cases} 1 \text{ if } \imath_E \text{ dominates one instance of } V_i \\ 0 \text{ otherwise.} \end{cases} \tag{9}$$

We can view the number of objects, dominated by $\imath_E$, as the sum of following random variables.

$$X_{\imath_E} = X_{V_1} + X_{V_2} + ... + X_{V_l} \tag{10}$$

Clearly, $E(X_{V_i}) = P(\imath_E \prec V_i)$, and $E(X_{\imath_E}) = \sum_{i=1}^{l} P(\imath_E \prec V_i)$. Since all V_is are mutually independent, we can apply the above Chernoff-Hoeffding bound with $\epsilon = \frac{(\gamma - E(X_{\imath_E}))}{E(X_{\imath_E})}$ to get Lemma 1, where $\gamma = \gamma_k - |FD(\imath_E)|$ and $|FD(\imath_E)|$ is the number of objects fully dominated by $\imath_E$.

Lemma 1 *If $E(X_{\imath_E}) < \gamma$, then $P_{\geq\gamma}(\imath_E \prec (\mathcal{U} - U)_i) \leq \exp^{-\frac{(\gamma - E(X_{\imath_E}))^2}{3E(X_{\imath_E})}}$.*

In our pruning technique, we will use $\exp^{-\frac{(\gamma - E(X_{\imath_E}))^2}{3E(X_{\imath_E})}}$ as an upper-bound of $P_{\geq\gamma}(\imath_E \prec (\mathcal{U} - U)_i)$. This will reduce the complexity of calculation from $O(\gamma \times l)$ to $O(l)$ when $\gamma > E(X_{\imath_E})$. This is significant when γ is large. Below, we present another upper-bound estimation of $P_{\geq\gamma}(\imath_E)$ when γ is relatively small — $\gamma \leq E(X_{\imath_E})$; in this case, Chernoff-Hoeffding Bound does not yield interesting results.

2. Bisection-based Upper-bound. Due to the above limitation when applying the Chernoff-Hoeffding Bound based Upper-Bound, we further develop a more general Upper-Bound called Bisection-Based Upper-bound. Following theorem is the key to obtain this upper bound. Without loss of generality, suppose that the l objects, partially dominated by the lower-left corner $\imath_E$ of an entry E, are sub-indexed such that $P(\imath_E \prec U_i) \leq P(\imath_E \prec U_j)$ if $i < j$. Let $p_i = P(\imath_E \prec U_i)$ for $1 \leq i \leq l$.

Theorem 2 *Suppose that we replace p_i by p_i^* for $1 \leq i \leq l$ such that $p_i \leq p_i^*$. Then, the probability that u dominates at least λ objects (for $1 \leq \lambda \leq l$) regarding $\{p_i : 1 \leq i \leq l\}$ is not greater than that regarding $\{p_i^* : 1 \leq i \leq l\}$.*

Theorem 2 is quite intuitive, but the proof is lengthy. Please refer to the appendix for the detailed proof.

Now, we can divide the probabilities of those partially dominated objects (by $\imath_E$) into two groups $\mathcal{G}_1 = \{p_1, p_2, ..., p_j\}$ and $\mathcal{G}_2 = \{p_{j+1}, p_{j+2}, ...p_l\}$ such that we replace each probability value in $\mathcal{G}_1$ by p_j and replace each probability value in $\mathcal{G}_2$ by p_l. The following Lemma is immediate.

Lemma 2 *Without loss of generality, we assume that $j \leq (n - j)$, let $y_0 = max\{0, \lambda - l + j\}$*

$$P_{\geq\gamma}(\imath_E \prec (\mathcal{U} - U)_i) \leq \sum_{y=y_0}^{j} C_j^y p_j^y (1 - p_j)^{j-y} \times \left(\sum_{x=\lambda-y}^{l-j} C_{l-j}^x p_l^x (1 - p_l)^{l-j-x} \right) \tag{11}$$

[3] Note that E_6 is fully dominated by E_1; consequently we no longer need to expand E_6 regarding E_1 but just add $P(E_6)$ to calculate the probabilities and scores of the children of E_1.

Proof Suppose that the instance u dominates l objects with the probabilities, $\overbrace{p_j, p_j \cdots, p_j}^{j}$, $\overbrace{p_l, p_l, \cdots, p_l}^{l-j}$. It can be immediately verified that the probability that $\imath_E$ dominates at least λ objects among these l objects is as what is stated on the right hand-side of the inequality of (11). The lemma immediately follows from Theorem 2. □

Lemma 2 states that we can bisect the set of partially dominated objects into two groups such that in each group, we use the largest probability value as a representative. Then, we use the right-side part of the inequality in (11) as an upper-bound. Clearly, it can be calculated in $O(l)$ time if we accumulatively compute the part, $\sum_{x=\lambda-y}^{l-j} C_{l-j}^{x} p_l^x (1-p_l)^{l-j-x}$, from the tail.

The key to deliver a good upper-bound is to choose a p_j such that the value of upper-bound can be minimized. This problem can be trivially solved in time $O(l^2)$ by enumerating all possible cases; nevertheless, such costs are even more expensive than the costs $O(\lambda \times l)$ of the dynamic programming based algorithm to produce the exact probability value.

In our computation, we choose the median to divide the set into two groups. It is clear that the median can be calculated in $O(l)$ time [12]. Therefore, the whole computation of upper-bound can be executed in time $O(l)$.

Remark 1 It seems hard to find an efficient algorithm with costs lower than $O(\lambda l)$ to divide l probability values into more than 2 groups; consequently we settle for a bisection. The bisection-based upper-bound can also be used in case when $\gamma > E(X_{\imath_E})$. However, our experiments, in Section 6, demonstrate that the above Chernoff-Hoeffding bound based upper-bound is tighter than the bisection-based upper-bound. Therefore, in our implementation we only use the Chernoff-Hoeffding bound for the case where $\gamma > E(X_{\imath_E})$. These two bounds will be used to calculate the upper-bounds of $P_{\geq\lambda}(\imath_E \prec (\mathcal{U} - U)_i)$ in our level-by-level computation.

We also examined Markov's inequality [20] and Chebyshev's inequality [20]; the upper-bounds generated by them are not as tight as the above two upper-bounds.

3. Utilizing Existing Computation Results. Below we show two upper-bounds by utilizing the existing computation results. One is dominating probability based, while another is *pscore* based.

Theorem 3 *Suppose that u is a point (or an instance of U_1) and u fully dominates an uncertain object, say, U_2. Then, $P_{\geq\gamma}(u) \geq P_{\geq\gamma}(U_2)$ ($\forall\gamma \geq 1$).*

The proof of Theorem 3 is quite lengthy and we leave it to Appendix.

Note that Theorem 3 will be used to prune away objects, fully dominated by u, if $P_{\geq\lambda}(u) < q$. The following Theorem is immediate.

Theorem 4 *Suppose that a point u (partially or fully) dominates λ' objects in total, and u dominates the lower-left corner of the MBB of an entry E of the local aR-tree of an object U at the level i. Then, $pscore^+(E) \leq \lambda'$.*

Note that in Theorem 4, level $i = 1$ means an object.

4.3.2 Effective Pruning Rules

The pruning rules below can be immediately verified from the definitions; thus we omit the proofs.

Pruning Rule 1 Score-Based: *$\forall U$, if $pscore^+(U) \leq \lambda_k$, then U can be excluded from the solution of PtopkQ.*

Let $L_i^+(U)$ denote the subset of entries of $L_i(U)$ with the property that $\forall E \in L_i^+(U)$, the captured $P_{\geq\lambda_k}^{upper}(E) \neq 0$. Based on equation (7), the following pruning rule is immediate.

Pruning Rule 2 Level's Probability-based: *Suppose that $\sum_{E \in L_i^+(U)} P_{\geq\lambda_k}(E) \leq q$. Then, U can be excluded from the solution of PtopkQ.*

Note that when $i = 1$, $L_i^+(U)$ in Pruning Rule 2 only contains the root entry: U.

In our computation if instances or entries in U are found with 0 probability to dominate λ_k objects, we mark and exclude them in further computation. For each entry E of a local aR-tree, let $\mathcal{I}_E^+$ denote the set of instances each of which is not yet detected with 0 probability to dominate at least λ_k objects, and $P(\mathcal{I}_E^+)$ denotes the sum of probabilities of instances in $\mathcal{I}_E^+$. The Pruning Rule 3 below is also immediate if we make the upper-bound of probability for an instance in $\mathcal{I}_E^+$ to dominate at least λ_k objects be 1.

Pruning Rule 3 Drilling-down based: *At the level i (for an i), if $\sum_{E \in L_i^+(U)} P(\mathcal{I}_E^+) < q$, then U can be excluded from the solution of PtopkQ.*

Pruning rules 2 and 3 are fundemental to a level-by-level computation (details in Section 4.3.3).

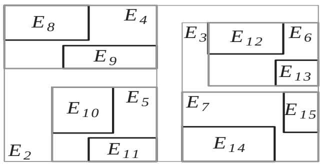


Fig. 7 Entry Distribution

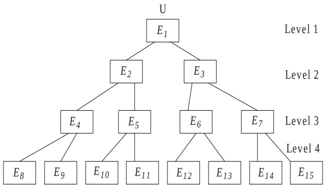


Fig. 8 Tree Structure Map

Remark 2 Note that in our level-by-level algorithm, an $\mathcal{I}_E^+$ may change when levels progress down. For instance, regarding the example in Figures 7 and 8, E_{13} and E_{15} are initially detected with $P_{\geq\lambda_k}(E_{15}) = 0$ and $P_{\geq\lambda_k}$ $(E_{13}) = 0$ because they are fully dominated by one point that (partially or fully) dominates no more than the current λ_k objects (formally stated in Theorem 4); consequently, $\mathcal{I}_{E_6}^+$ contains the instances contained by E_{12}.

Nevertheless, once progress to level 2, we may find that the total number of objects (fully or partially) dominated by E_6 is less than threshold λ_k; consequently, in $\mathcal{I}^+_{E_6}$ we replace E_{12} by $\varnothing$. Thus $\mathcal{I}^+_{E_6}$ is empty.

4.3.3 Algorithm Details

Step 3.1. Objects corresponding to the centroids in an entry E of the aR-tree on centroids may be spread to different entries of the aR-tree on object MBBs.

Example 10 Regarding the centroids ω_1, ω_2, and ω_3 in Figures 3, their corresponding objects U_1, U_2, and U_3 are spread to 2 entries in the local aR-tree on object MBBs.

In each entry E of the local aR-tree on object MBBs, we record the lower left corner of the MBB encompassing the objects that correspond to the contained centroids in E, denoted by $\jmath_E$. Note that $\jmath_E$ is not the lower left corner of E. Below is the algorithm presented in Algorithm 4.

Algorithm 4 Step 3.1

```
Description:
1: get $pscore^+(E)$;
2: if $pscore^+(E) \le \lambda_k$ (Pruning Rule 1) then
3:    prune other objects dominated by $\jmath_E$
4: else
5:    if $E$ is an object $U$ then
6:       record $PD(U)$ and goto Step 3.2;
7:    else
8:       for each child $E'$ of $E$ do
9:          call Algorithm 4 regarding $E'$;
```

To compute $pscore^+(E)$ - an upper-bound of the maximum number of objects (partially or fully) dominated by an object in E, we use window query techniques by the "half-open" window with $\jmath_E$ as the lower-left corner to probe the aR-tree on MBBs of objects and then count the number of objects overlapping with the window as $pscore^+(E)$.

If the condition in line 2 holds, then objects corresponding to the centroids in E will be excluded from a further consideration. In this case, we can prune other objects by $\imath_E$ by using the above window to probe the aR-tree of objects to get the objects fully dominated by $\jmath_E$. These objects will be removed from $\overrightarrow{\mathcal{U}}$ or from an entry in $\overrightarrow{\mathcal{U}}$. Note that when objects removed from an entry E of $\overrightarrow{\mathcal{U}}$, we need to update $\jmath_E$ and the corresponding information in its descendants. Moreover, if an entry in the aR-tree of objects is detected to be fully dominated by $\jmath_E$, then it is marked so that the entry can be skipped when another $\jmath_{E'}$ is used to prune away objects.

Example 11 In Step 3.1, suppose the current λ_k is 3. When the entry containing ω_7, ω_8, and ω_9 is selected, we use the recorded lower-left corner (with this entry) of the MBB of objects U_7, U_8, and U_9 to do the window query on the aR-tree of objects. The window query does not intersect any object. Consequently, the entry containing ω_7, ω_8, and ω_9 will be removed from candidates, and U_7, U_8, and U_9 are excluded from the candidates of PtopkQ.

Remark 3 At the object level, we also use Pruning Rule 3 to check (line 2 of Algorithm 4) if an object should be removed from the candidates of PtopkQ.

Step 3.2. For each remaining object U, we synchronously traverse the local aR-trees of U and objects in $PD(U)$ level-by-level such that at each internal level i, we conduct the following two substeps.

Step 3.2a. Use Pruning Rule 3 to check if U should be removed. If U cannot be removed, then go to Step 3.2b.

Step 3.2b. For each $E \in L^+_i(U)$, we compute $PD(E)$ and $|FD(E)|$. Then, based on Theorem 1 we use Chernoff-Hoeffding bound based upper bound or Bisection based upper bound to bound $P_{\geq\lambda}(\imath_E \prec (\mathcal{U} - U)_i)$, which is multiplied by $\sum_{u \in E} P(u)$ to give an upper bound $P^{upper}_{\geq\lambda_k}(E)$ of $P_{\geq\lambda_k}(E)$. Then, we use Pruning Rule 2 to check if U should be excluded or goto the next level. Note that when applying Pruning Rule 2, we replace $P_{\geq\lambda_k}(E)$ by $\min\{P^{upper}_{\geq\lambda_k}(E), P(\mathcal{I}^+_E)\}$.

To efficiently execute Pruning Rule 3, for each entry E we record the summation $p^0(E)$ of occurrence probabilities of detected instances that have 0 probability to dominate at least λ_k objects. Once an entry E is detected to have every instance with 0 probability dominating at least λ_k objects, this information is propagated to all ancestors as follows if E is the first time, (i.e. $full(E) = 0$), detected. Let $full(E) = 1$ denote the situation that every instance in E has already been detected to be with 0 probability dominating at least λ_k objects.

Algorithm 5 Propagation to Ancestors

```
Description:
if $full(E) = 0$ then
   $full(E) = 1$; $p' := p^0(E)$; $p^0(E) := P(E)$;
   for each ancestor $E'$ of $E$ do
      if $full(E') = 0$ then
         $p^0(E') := p^0(E') + P(E) - p'$;
         if $P(E') = p^0(E')$ then
            $full(E') = 1$
      else
         Terminate
```

Example 12 Regarding the example in Figures 7 and 8, suppose that E_{15} is detected to be fully dominated by a point that has zero probability to dominate at least λ_k objects. Then, $P_{\geq\lambda_k}(E_{15}) = 0$. Further suppose that each entry at the bottom level has instances with the total probability 1/8. Thus, we record $full(E_{15}) = 1$, $P^0(E_{15}) = P^0(E_7) = P^0(E_3) = P^0(E_1) = 1/8$.

Assume that another such point is found to fully dominate E_3. Then, update $full(E_3)$ to be 1, and $P^0(E_3) = 1/2$ and $P^0(E_1) = 1/2$. If we find the third such point that fully dominates E_{15}, the search of E_{15} will stop at E_3 since $full(E_3) = 1$.

Remark 4 Once the lower-left corner $\imath_E$ of an entry E is detected to have 0 probability to dominate at least λ_k objects, we use window query techniques to check if entries from other objects are fully dominated by $\imath_E$. For any entry fully dominated by $\imath_E$, we apply Algorithm 5 to propagate to ancestors of the entry. Moreover, when an object is processed as a candidate in Step 3.2, we do not need to expand its entries E with $full(E) = 1$.

Step 3.3. We use the dynamic programming method to calculate *pscore*(U) as what is described in Step 2. Note that when an instance u is detected $P_{\geq \lambda_k}(u) < q$, we can apply Theorem 3; that is, we do window queries, in the same way as described in the above step, by excluding all objects fully dominated by u, and update $\overrightarrow{U}$ accordingly.

5 Randomized Algorithm

The basic idea of our randomized algorithm is to sample all possible worlds, $\prod_{i=1}^{n} U_i$ from $\mathcal{U} = \{U_i | 1 \leq i \leq n\}$, by a small number m of possible worlds $\mathcal{S}_i$ $(1 \leq i \leq m)$, where each $\mathcal{S}_i$ consists of n instances - one instance per object. An instance u *homo-dominates* another instance v if u dominates v, and they are in one sample $\mathcal{S}_i$. Let $u_{i,j}$ denote an instance in sample S_i from object U_j. $pscore^r(u_{i,j})$ is defined as the number of instances in sample S_i that are dominated by $u_{i,j}$; that is, the number of instances homo-dominated by $u_{i,j}$. For $1 \leq j \leq n$, $pscore^r(U_j)$ is the $(q*m)$th largest in $\{pscore^r(u_{i,j}) | 1 \leq i \leq m\}$.

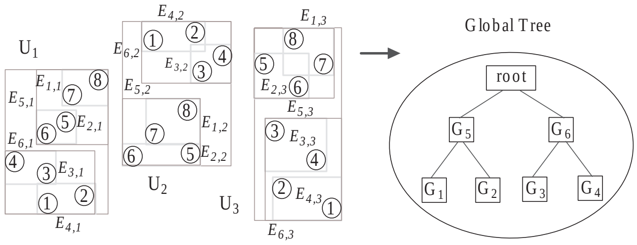


Fig. 9 Samples

Example 13 Regarding the example in Figure 9, suppose that $m = 8$, $k = 2$, and $q = 0.5$. A circled number j in object U_i means the sampled instance (from U_i) is in the sample j. The $pscore^r$ of object U_1 is 2. This is because that the samples 1, 2, 3 and 4 homo-dominate two other samples respectively (i.e. samples with the same sub-indexes) from U_2 and U_3, while samples 5, 6, 7, and 8 homo-dominate 1 sample, respectively.

Similarly, we obtain that $pscore^r(U_2) = 1$ and $pscore^r(U_3) = 0$. Therefore, Algorithm 6 returns U_1 and U_2 as the top-k objects.

Below, Algorithm 6 outlines our randomized algorithm.

In Algorithm 6, Calculating-$pscore^r$ $(\{\mathcal{S}_i : 1 \leq i \leq m\}, q)$ returns the k objects with the highest $pscore^r$s. A naive way of Calculating-$pscore^r$ is to compute the dominating number for each sampled instance in $\mathcal{S}_i$ for $1 \leq i \leq m$; consequently, we need to perform such computation m times if there are m samples. Our experiments demonstrate such a naive algorithm is very expensive, slow, and not scalable against m. Below, we present a novel, efficient algorithm for Calculating-$pscore^r$ with the aim to share the computation among samples and to effectively prune away objects. First, we show an accuracy guarantee of the algorithm.

Algorithm 6 Randomized Algorithm

```
Input: {$\mathcal{S}_i : 1 \leq i \leq m$}; $0 < q \leq 1$.
Output: $\mathcal{T}_k$ : the $k$ objects with the largest $pscore^r$.
Description:
1: $\mathcal{T}_k$ := Calculating-$pscore^r$ ($\{\mathcal{S}_i : 1 \leq i \leq m\}, q$);
2: return $\mathcal{T}_k$
```

5.1 Accuracy Guarantee

For each object U_j, the events whether in sample $\mathcal{S}_i$, the randomly selected instance $u_{i,j}$ dominates at least l other instances may be described by the following totally independent random variables.

$$X_{\geq l,i,j} = \begin{cases} 1 \text{ if } u_{i,j} \text{ dominates at least } l \text{ instances in } \mathcal{S}_i \\ 0 \text{ otherwise} \end{cases}$$

Clearly, $E(X_{\geq l,i,j}) = \sum_{u \in U_j} P(u) P_{\geq l}(u) = P_{\geq l}(U_j)$. Let

$$X_{\geq l,j} = \frac{\sum_{i=1}^{m} X_{\geq l,i,j}}{m}.$$

It is immediate that $E(X_{\geq l,j}) = P_{\geq l}(U_j)$. Theorem 5 immediately follows the Hoeffding's inequality [20] (Theorem 6).

Theorem 5 *If* $m = O(\frac{1}{\epsilon^2} \log \frac{1}{\delta})$ *where* $0 < \delta, \epsilon < 1$, *then* $P(|X_{\geq l,j} - P_{\geq l}(U_j)| \geq \epsilon) < \delta$.

Theorem 6 Hoeffding's Inequality: *Suppose that* Y_1, Y_2, ... , Y_m *are independent random variables such that* $0 \leq Y_i \leq 1$ *for* $1 \leq i \leq m$. *Let* $Y = \sum_{i=1}^{m} Y_i$. *Then, we have that:*

$$\begin{aligned} P(Y - E(Y) \geq \epsilon m) &\leq \exp^{(-2\epsilon^2 m)} \\ P(E(Y) - Y \geq \epsilon m) &\leq \exp^{(-2\epsilon^2 m)} \end{aligned} \tag{12}$$

Theorem 5 implies that $P_{\geq l}(U_j) - \epsilon \leq X_{\geq l,j} \leq P_{\geq l}(U_j) + \epsilon$ with confidence $1 - \delta$.

In our randomized algorithm — Algorithm 6, we use $X_{l,j}$ to approximately represent $P_{\geq l}(U_j)$; consequently, $(q*m)$th greatest number $(pscore^r(U_j))$ of homo dominated instances is used to approximately represent $pscore_q$ (U_j). Below is a theoretical guarantee of our randomized algorithm.

Lemma 3 *In Algorithm 6, suppose that we replace q by* $(1-\epsilon)q$ *in Algorithm 6, replace* ϵ *by* ϵq *in Theorem 5, and change m from* $O(\frac{1}{\epsilon^2} \log \frac{1}{\delta})$ *to* $O(\frac{1}{\epsilon^2 q^2} \log \frac{n}{\delta})$. *Then the top-k objects retrieved by Algorithm 6 have the following properties with confidence* $1 - \delta$. *For* $1 \leq i \leq k$ $(\forall k \leq n)$,

Property 1: the $pscore^r$ of the top ith object U_i is not smaller than the pscore of the top ith object to PtopkQ (regarding q);
Property 2: $P_{\geq pscore^r}(U_i) > (1-2\epsilon)q$.

Proof It can be immediately verified that for each object U_i,

$$X_{pscore^r(U_i)+1,i} > (1-\epsilon)q. \tag{13}$$

Consequently, we have $P_{\geq pscore^r(U_i)+1}(U_i) \leq q$ with very small probability $\frac{\delta}{n}$ applying Theorem 5. The Property 1 immediate follows.

From Theorem 5, Property 2 immediately follows. □

Note that the sample size in Theorem 5 and Lemma 5 is irrelevant to the number of instances in an object; thus, the randomized algorithm has a potential to support the applications where a large number of instances is involved.

While Theorem 5 and Lemma 3 provide theoretical performance guarantee, our experiments demonstrate that Algorithm 6 is quite accurate when m is up to 1000 and q, instead of $(1-\epsilon)q$ used in Algorithm 6.

5.2 Efficient Algorithm

We present an efficient algorithm to execute Calculating-$pscore^r$. It follows the framework of 3 Steps in Section 3, Pre-ordering, Initial Computation, and Final Computation. We first present a novel data structure to replace local aR-trees.

***gC*aR-tree.** The sampled instances of each object are organized into an R-tree like structure, *gCaR*-tree (*Global Constrained* aR-tree). Different from a conventional R-tree, n gCaR-trees for n objects (one gCaR-tree per object) follow a global tree structure as follows.

> Corresponding to each node G_i in the global tree, for all $1 \leq j, j' \leq n$, entries $E_{i,j}$ and $E_{i,j'}$ of U_j and $U_{j'}$ contain the instances from the same samples, respectively. For example, in Figure 9, corresponding to G_2, $E_{2,1}$, $E_{2,2}$, and $E_{2,3}$ contain the sampled instances from the samples 5 and 6, respectively.

In a gCaR-tree, the number of sampled instances in each entry is also recorded. Given a global tree structure, we aim to minimize the sum of areas of gCaR-tree for all uncertain objects. It can be immediately shown that this optimization problem is NP-hard since a special case of the problem (i.e., when $n = 1$) is the area minimization problem of an R-tree which is NP-hard.

We build n gCaR-trees following the techniques of building an R-tree except that we enforce the constraint of a global tree structure as above. gCaR-trees have the advantage that in level-by-level computation, only the *homo-dominating* relationships among objects need to be checked. An entry E (fully or partially) *homo-dominates* another entry E' if E and E' correspond to the same entry in the global tree and E (fully or partially) dominates E'.[4] Consequently, for each entry E we only need to check one entry per object to determine if there is a homo-dominating relationship. Thus, the total costs to compute all entries, at the ith level of gCaR-trees, homo-dominated by an entry in U takes $O(n')$ where n' is the number of objects partially dominated by U. This is much lower than $O(N)$ in the exact algorithm where N is the number of total entries at the ith level.

However, such a global constraint may also bring a disadvantage — sizes of MBBs may be too large. Clearly, traversing gCaR-trees from a parent E to a child E' does not bring much extra geometric information if the MBB of E' has a similar size to that of E. To resolve this, we introduce a post-processing as follows while building gCaR-trees.

Post-processing a *gC*aR-tree. We enforce the constraint that for each group G_i,

$$\frac{area(G_i)}{area(G_i^p)} \leq \rho. \tag{14}$$

Here, $area(G_i)$ denotes the total area of the MBBs from n gCaR-trees corresponding to G_i, while G_i^p denotes such total area corresponding to the parent of G_i. If a G_i does not satisfy the inequality (14), then we go to the children of G_i and check the children of G_i one by one (still against G_i^p), so on and so forth. Below is the algorithm.

Algorithm 7 gCaR post-processing

```
Input: n gCaR-trees; the root $Gr$ of the gobal tree; $0 < \rho \leq 1$.
Output: n gCaR-trees following the inequality (14).
Description:
 1: $\mathcal{Q} := \{$children of $Gr\}$;
 2: while $\mathcal{Q} \neq \emptyset$ do
 3:     get a $G$ from $\mathcal{Q}$;
 4:     $\mathcal{Q} := \mathcal{Q} - \{G\}$;
 5:     if $\frac{area(G)}{area(Gr)} < \rho$ then
 6:         if $G$ is not children of $Gr$ then
 7:             modify the global tree (thus $n$ gCaR-trees) by using $Gr$ as the parent of $G$;
 8:         call Algorithm 7 with $G$ as the root if $G$ is not a data point;
 9:     else
10:         add children of $G$ to $\mathcal{Q}$;
```

Example 14 In Figure 9, suppose that we choose $\rho = 1/3$. G_6 does not follow the inequality in line 5; thus we link the root to G_3 and G_4 (thus, remove G_6). However, G_5 follows the inequality; consequently G_5 is used as the root to call Algorithm 7. None of G_1 and G_2 follows the inequality (14). Therefore, the final result is that in these three gCaR-trees, the root has three children corresponding to G_5, G_3, and G_4, respectively; the next level contains all sampled instances.

[4] in Figure 9, $E_{6,1}$ fully homo-dominates $E_{6,2}$ and $E_{6,1}$ partially homo-dominates $E_{6,3}$.

In our algorithm, those gCaR-trees are pre-computed. We assign 1/4 to ρ since it leads to a very good performance according to our initial experiments.

Calculating-*$pscore^r$*. Our algorithm closely follows the framework of the exact algorithm with the following modifications. Let λ_k denote the smallest $pscore^r$ value of the current top-k candidates.

1: Pruning Rules.

For each object, we search for the $(q * m)$th greatest homo-dominating number $pscore^r$ among the sampled instances. Below are the pruning rules that we will use in our randomized algorithm.

Pruning Rule 4 *$\forall U$, if $pscore^r(U) \leq \lambda_k$, then U can be excluded from the solution of PtopkQ.*

Note that at the object level, we use the number of objects (totally or partially) dominated by U as an upper-bound of $pscore^r(U)$ for applying Theorem 4. Let $\mathcal{M}_{\leq \lambda_k, U_i}$ denote the number of sampled instances, from U_i, with their $pscore^r \leq \lambda_k$, respectively.

Pruning Rule 5 *An object U_i can be excluded from the solution of PtopkQ (against the probability threshold q) if $\mathcal{M}_{\leq \lambda_k, U_i} \geq (1-q) \times m + 1$.*

Note that Pruning rule 5 will be used to replace Pruning Rules 2 and 3 in the exact algorithm.

2: Step 1 and 2 - Pre-ordering and Initial computation

While the Step 1 (pre-ordering objects) in our randomized algorithm is the same as Step 1 in the exact algorithm, Step 2 for computing scores of the first k objects is conducted differently. We need to compute $pscore^r$ for each object instead of *pscore*. Below is the algorithm, Algorithm 8, to calculate the $pscore^r$ for one object.

Algorithm 8 Calculating $pscore^r$

```
Input: $U$; $PD(U)$; $FD(U)$; $0 < q \leq 1$; $m$ samples.
Output: $pscore^r$ of $U$;
Description:
for each sampled $u \in U$ do
    compute $pscore^r(u)$ against $PD(U)$;
    $\delta :=$ the $(q \times m)$th largest value of $pscore^r(u)$;
$pscore^r(U) := |FD(U)| + \delta$;
```

Note that the computation of $pscore^r(u)$ ($\forall u \in U$) is conducted within the sample that u belongs to. We incrementally maintain a min-heap [12] against the current top-$(q \times m)$ instances (i.e., with the largest $pscore^r$s) or a max-heap against the current bottom-$[(1-q)m+1]$ objects depending on whether $q \leq 0.5$. Clearly, Algorithm 8 runs in time $O(m|PD(U)| + m \log(q * m))$ for each object U.

3: Step 3 - Final Computation.

In this step, we use the same bounding-pruning-refining framework as in the exact algorithm by effectively using the following Theorem 7 in combination with Pruning Rules 4 and 5. Let $\nu_{i,j}$ denote the largest number of instances homo-dominated by an instance contained by an entry $E_{i,j}$ of a gCaR-tree of object U_j. For example, regarding the example in Figure 9, $\nu_{6,1} = 2$.

Theorem 7 *Suppose that an $E_{i,j}$ fully homo-dominates l_1 entries and partially homo-dominates l_2 entries. Further suppose that $\imath_{E_{i,j}}$ dominates $\imath_{E_{i,j'}}$. Then,*

1. *$\nu_{i,j} \leq l_1 + l_2$,*
2. *$\nu_{i,j'} \leq l_1 + l_2$.*

Theorem 7 is immediately based on the definitions, and is used in level-by-level computation. Below we present our algorithm details. It also consists of 3 steps: Step 3.1, Step 3.2, and Step 3.3.

Step 3.1: pruning at the object level. It is the same as Step 3.1 in the exact computation (Algorithm 4).

Step 3.2: level-by-level pruning. The basic idea is to synchronously traverse the gCaR-trees of a U_j and the uncertain objects in $PD(U_j)$. For an object U_j, let $L^+_\kappa(U_j)$ denote the set of entries at the κ level of the gCaR-tree such that for each entry $E_{i,j}$ in $L^+_\kappa(U_j)$, $\mu_{i,j}$ is not captured less than λ_k. In our algorithm, we initialize each object U_j by assigning 0 to $\mathcal{M}_{\leq \lambda_k, U_j}$ and the root entry of U_j to $L^+_1(U_j)$. The step proceeds as follows for each remaining object U_j.

At each level κ, for every entry $E_{i,j}$ in $L^+_\kappa(U_j)$ we compute l_1 and l_2. If $l_1+ \ l_2 \geq \lambda_k$, we add the child entries, which are not marked out, of $E_{i,j}$ to $L^+_{\kappa+1}(U_j)$ for the computation at the next level.

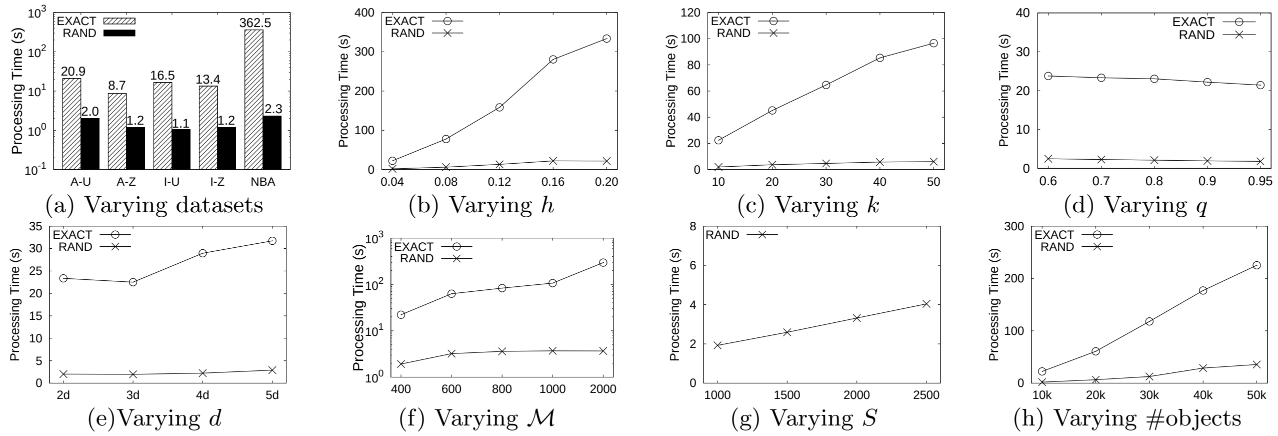
Otherwise ($l_1 + l_2 \leq \lambda_k$), according to Theorem 7 we do the following two things.

1. For every entry $E_{i,j'}$ ($\forall U_{j'} \in PD(U_j)$) such that $U_{j'}$ has not been processed in Step 3 and $E_{i,j'}$ is not marked out, if $\imath_{E_{i,j}} \prec \imath_{E_{i,j'}}$, $\mathcal{M}_{\leq \lambda_k, U_{j'}} = \mathcal{M}_{\leq \lambda_k, U_{j'}} + a_{i,j'}$.[5] $U_{j'}$ will be marked out for further consideration if the updated $\mathcal{M}_{\leq \lambda_k, U'_j} \geq (1-q) \times m + 1$ (Pruning Rule 5). In case that $U_{j'}$ cannot be excluded, $E_{i,j'}$ is marked out by using Algorithm 5; that is, it will not be considered while processing $U_{j'}$.
2. Update $\mathcal{M}_{\leq \lambda_k, U_j}$ to $\mathcal{M}_{\leq \lambda_k, U_j} + a_{i,j}$. Then exclude U_j from the result set if $\mathcal{M}_{\leq \lambda_k, U_j} \geq (1-q) \times m + 1$ (Pruning Rule 5).

If U_j is not pruned in Step 3.2, then we invoke Step 3.3.

Step 3.3: final computation. At the leaf level, for all instances in the remaining entries of U_j we compute their actual values of $pscore^r$ and return the $(q \times m)th$ largest value as $pscore^r(U_j)$. If $pscore^r(U_j) > \lambda_k$, then we replace the object with the smallest $psocre^r$ (i.e., λ_k) among the current top-k objects by U_j and update λ_k.

[5] To avoid to over-count already marked-out entries, $a_{i,j'}$ is the number of instances in the subentries of $E_{i,j'}$ that have not been marked out. To efficiently record such $a_{i,j'}$ for each entry, we apply Algorithm 5.



(a) Varying datasets (b) Varying h (c) Varying k (d) Varying q

(e)Varying d (f) Varying $\mathcal{M}$ (g) Varying S (h) Varying #objects

Fig. 10 Runtime with respect to different parameters.

6 Experimental Study

In this section, we present a thorough performance evaluation of the efficiency and effectiveness of our algorithms. All algorithms are implemented in C++. Experiments are run on PCs with Intel P4 2.8GHz CPU and 2G memory under Debian Linux.

We refer to the exact algorithm in Section 4 as **EXACT**, and to the randomized algorithm in Section 5 as **RAND**.

Two types of datasets are used in our evaluation process.

Real dataset is extracted from NBA players' game-by-game statistics (http://www.nba.com), containing 339,721 records of 1,313 players. Performance of a player is treated as an uncertain object and the statistics of a player in a single game is treated as an instance of an uncertain object. For one player, all instances are assumed to take the same probability to appear. In our experiment, we use three attributes, points, assistances, and rebounds in an instance. Since larger values of those attributes are preferred, we adopt the corresponding negative values[6].

Synthetic datasets are generated using methodologies in [6] with respect to the following parameters. Dimensionality varies from 2 to 5 with default value **3**. Data domain along each dimension is $[0, 1]$. Number of objects varies from $10,000$ to $50,000$ where default value is **10, 000**. Number of instances per object follows a uniform distribution in $[1, \mathcal{M}]$ where $\mathcal{M}$ changes from 400 to $2,000$ with the default value **400**. Each MBB to bound an uncertain object is a hype-cube; and the average edge length of MBB of uncertain objects follows a normal distribution in the range [0,h] with the expectation value $h/2$ and standard deviation 0.025; the default value of h is 0.04 — 4% of the edge length of the whole data space. The value k in PtopkQ varies from 10 to 50 with default value 10. As for randomized algorithm, sample size varies from 1000 to $2,500$. Table 4 summarizes parameter ranges and default values (in bold font). Note that in the default setting, the total number of instances is about 2 millions.

Instances of an object follow either *uniform* (random) or *zipf* distribution. In *uniform* distribution, instances are distributed uniformly inside the uncertain range with the same occurrence probability. In *zipf* distribution, firstly an instance u is randomly generated and the distances from all other instances to u follow a zipf distribution with $z = 0.5$. The occurrence probability for each instance also follows zipf distribution with $z = 0.2$.

Centers of objects (objects' MBBs) follow either *anti-correlated* or *independent* distribution. So, in all we have four types of synthetic datasets combining object centers and instances distribution: Anti-Uniform, Inde-Uniform, Anti-Zipf and Inde-Zipf. These are abbreviated to **A-U**, **I-U**, **A-Z**, and **I-Z** in our experiment reports.

dimensionality d	2, **3**, 4, 5
number of objects	**10k**, 20k, 30k, 40k, 50k
edge length h	**0.04**, 0.08, 0.12, 0.16, 0.20
number of instances $\mathcal{M}$	**400**, 600, 800, 1k, 2k
k	**10**, 20, 30, 40, 50
q	0.6, 0.7, 0.8, **0.9**, 0.95
sample size S	**1k**, 1.5k, 2k, 2.5k
data types	**A-U**, A-Z, I-U, I-Z, NBA

Table 4 Parameter values.

6.1 Efficiency Evaluation

We evaluate our algorithms against the parameters in Table 4.

Overall Performance. Figure 10(a) reports the result of our performance evaluation over synthetic (with the

[6] Note that there might be correlations among the player statistics. We ignore the correlations so that NBA data can be used to test efficiency and effectiveness of our techniques.

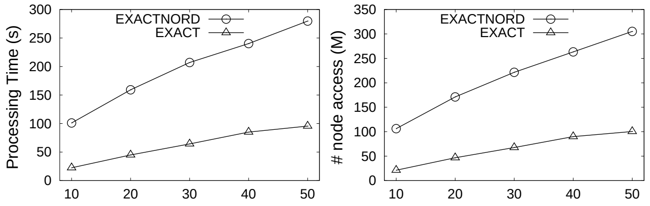
default setting) and real datasets. The experiment demonstrates that while EXACT is very efficient against various synthetic datasets with the default setting, it is slower against the NBA dataset. This is because in the NBA dataset, MBB sizes are large relative to the whole data space; this gives a very high overlapping degree among objects' MBBs. On the other hand, RAND very effectively deals with such situation. RAND has a very steady performance and is at least 10 times faster than EXACT against all these datasets. We run the trivial exact algorithm as discussed in Section 2.3; that is, compute *pscore* for each object and then choose the top-k. Our experiment results show that it is about 100 times slower than EXACT. We also implement the trivial randomized algorithm as discussed in Section 5; that is, compute $pscore^r$ for each instance in a sample. The costs are 1589(s), 1543(s), 3081(s), 3376(s), and 115(s), respectively; it increases to 6685(s) when 2500 samples are used. Consequently we omit the evaluation of both trivial algorithms in the rest of our experiments. Note that the trivial randomized algorithm runs fast against NBA data; this is because NBA data only have about 1000 objects.

Varying MBB sizes, k and q. Figure 10(b) reports our second experiment results, against synthetic datasets with different average MBB sizes. Figure 10(c) reports our performance evaluation against different k values. While the costs of EXACT linearly increase when k increases, the performance of RAND is quite steady. This is because that the costs of computing the scores, $pscore^r$, for objects in RAND are no longer as dominant as that in EXACT. The experiment results, depicted in Figure 10(d), show the impact from different q values is quite minor.

Varying other parameters. Figures 10(e) - (g) report the possible impacts against dimensionality, average instance numbers, and average sample size. It is interesting to note that the costs of EXACT generally increase with the increment of dimensionality but the costs in $3d$ are slightly less than that in $2d$; this is because the ratio of average MBB volume against the data space decreases with the increment of dimensionality. Nevertheless, the experiment demonstrates that an increment of dimensionality plays a dominant role in the costs from $3d$.

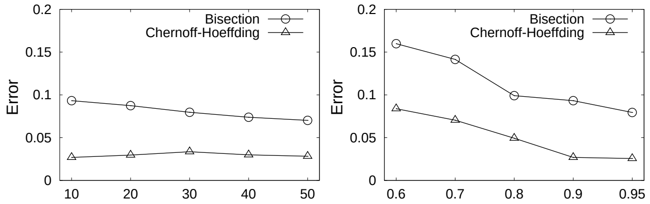
The impact of the number of objects is plotted in Figure 10(h). Although the processing time of two algorithms both increases as more uncertain objects are involved, RAND has overall better performance and also degrades much more slowly than EXACT.

Accessing order. In order to evaluate the effectiveness of the objects accessing order in Section 4.1, we also implement another version of the exact algorithm, named EXACTNORD, in which the objects are accessed with a random order. We evaluate the processing time as well as the number of node access of two algorithms with k varying from 10 to 50 in Figure 11. As depicted in Figure 11(a), the accessing order plays an important role for the computation as the EXACT algorithm significantly outperforms the EXACTNORD. We also use the *warm-buffer* paradigm to run our algorithms to evaluate I/O costs. In Figure 11(b), we record the number of node access for the aR-Trees of the uncertain objects during the computation. As expected, the number of node access of EXACT Algorithms is much less than that of EXACTNORD.



(a) *Processing Time vs k* (b) *# Node Access vs k*

Fig. 11 Performance vs diff. object access orders



(a) Varying k (b) Varying q

Fig. 12 Chernoff-Hoeffding based vs Bisection based

6.2 Pruning Powers

Chernoff-Hoeffding vs Bisection. We first evaluate the effectiveness of the Chernoff-Hoeffding-bound based upper bound and the Bisection-based upper bound. The experiment is conducted against the real data - NBA dataset. In our experiment, we first vary k values and then vary q values. We record the average value of

$$P^{upper}_{\geq\lambda_k}(U) - P_{\geq\lambda_k}(U)$$

during query processing where $P_{\geq\lambda_k}(U)$ is the actual probability and $P^{upper}_{\geq\lambda_k}(U)$ represents the Chernoff-Hoeffding-bound based upper bound and the Bisection-based upper bound, respectively. Note that for a fair comparison, we only record such average for the Bisection-based upper bounds when Chernoff-Hoeffding Bound based upper bound can be used. The results are reported in Figure 12(a) and Figure 12(b). They demonstrate that the Chernoff-Hoeffding Bound based upper bound is tighter than the Bisection-based upper bound. This is the reason that in our algorithm, we employ the Chernoff-Hoeffding Bound based upper bound whenever applicable.

Various Pruning Techniques. Figures 13 and 14 report our evaluation of the effectiveness of the pruning

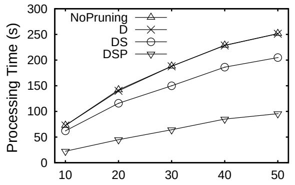


Fig. 13 Varying k

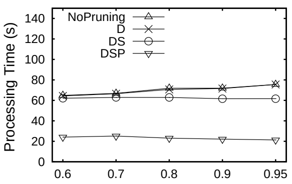
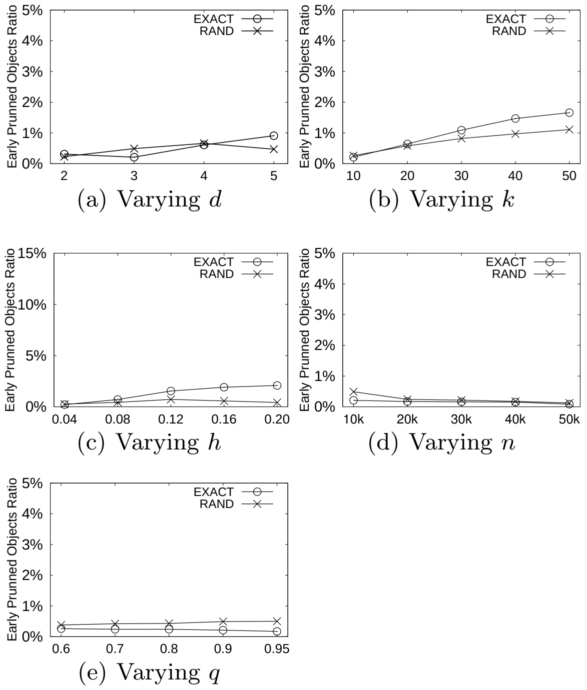


Fig. 14 Varying q



(a) Varying d

(b) Varying k

(c) Varying h

(d) Varying n

(e) Varying q

Fig. 15 Node calculated ratio with respect to different parameters.

rules presented in the paper with various k values and q values, respectively. NoPruning denotes the exact algorithm without applying any pruning rules in Section 4.3.2 at the instance levels, D denotes that we apply the *Drill-down*-based pruning rule, DS denotes that we apply the *Drill-down*-based pruning rule and the *Score* based pruning rule at each level, and DSP denotes that we apply *Level's Probability*-based pruning rule (i.e. Chernoff-Hoeffding Bound based upper-bound and the Bisection-based upper-bound) each level in addition to DS. They demonstrate that an application of the *Drill-down*-based pruning rule alone does not improve much efficiency since it basically still functions at the object level. While combining with level-by-level score based pruning rule does improve efficiency noticeably, adding Chernoff-Hoeffding Bound based upper-bound and the Bisection-based upper-bound significantly improves the performance. This is because the computation costs at each level are significantly reduced by using those upper-bounds and the upper-bounds are tight. Note that NoPruning is basically the combination of the techniques in [43] and techniques in [22,42] on the top of our pre-ordering techniques.

Effectiveness. We report our performance evaluation of pruning power of EXACT and RAND in Figure 15 against dimensionality, k values, edge lengths, object numbers, and q values. The experiment is conducted against syntectic data in order to evaluate all possible impacts. We record "early pruned object ratio" - the ratio of the number of objects, with entries of local aR-trees accessed from the 2nd level onwards, over the total number of objects. Our evaluation reports that the exact algorithm has a very powerful set of pruning techniques and up to 97% of objects have been pruned from the candidate sets even when MBB is large.

6.3 Accuracy Evaluation

We evaluate possible impacts of different parameters on the accuracy of RAND. Evaluation is based on average *relative errors* for a retrieved object's dominating probability with its $pscore^r$ computed using RAND regarding a given threshold q. We use the following relative error metrics to evaluate the ability of RAND to meet a given threshold q. Without loss of generality, U_i denotes the top-ith object returned by RAND.

$$err_i^p = \begin{cases} 0 & \text{if } P_{\geq pscore^r(U_i)} \geq q \\ \frac{|P_{\geq pscore^r}(U_i) - q|}{q} & \text{otherwise.} \end{cases}$$

Figure 16 reports our performance evaluation, regarding the average relative error ($\frac{\sum_{i=1}^{k} err_i^p}{k}$), against data types, number of objects, k values, q values, different average MBB sizes, dimensionality, and sample sizes. Our experiment results demonstrate that when sample size reaches 1000, the relative error is already very small. Moreover, the accuracy is not quite related to the dimensionality, object number, k values, or MBB sizes. Nevertheless, the accuracy decreases when q gets smaller; this is because when q is smaller, RAND requires more samples to retain the same accuracy according to our theoretic results in Section 5. It also shows that the accuracy increases when the sample size increases.

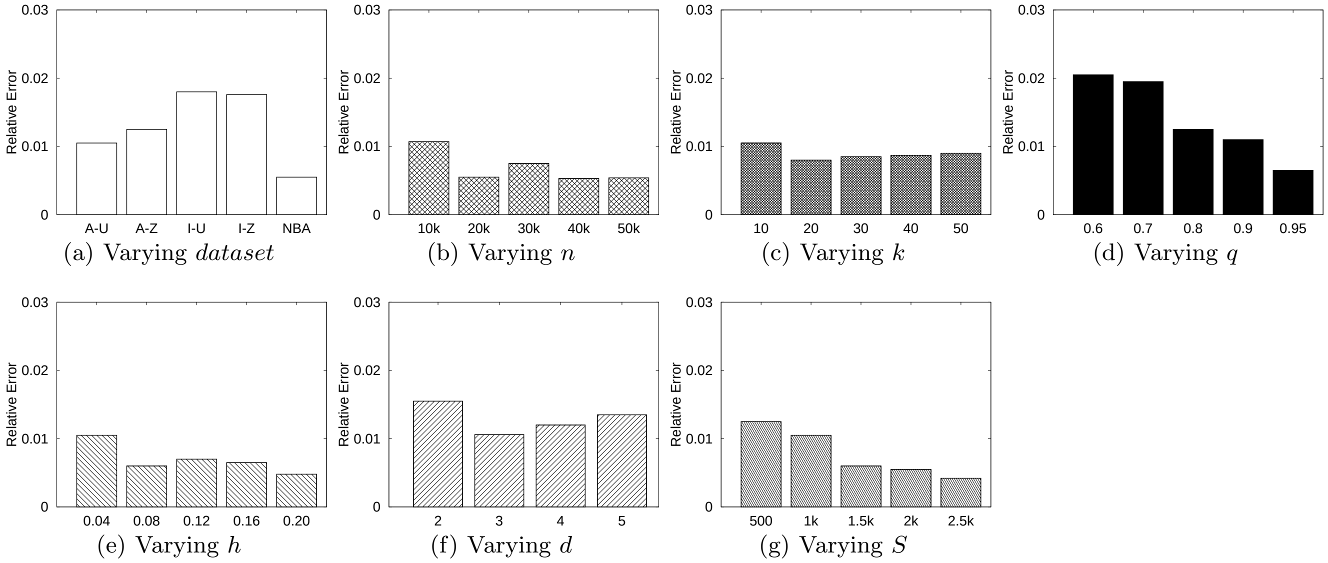
We also evaluate the accuracy in the top-k scores output by RAND using the average relative error metrics - $\frac{\sum_{i=1}^{k} err_i^l}{k}$ where

$$err_i^l = \begin{cases} 0 & \text{if } P_{\geq pscore^r}(U_i) \geq pscore_i \\ \frac{|pscore^r(U_i) - pscore_i|}{pscore_i} & \text{otherwise.} \end{cases}$$

As demonstrated in Figure 17, the performance of RAND is very accurate - the average relative error is less than 0.4%. It is interesting to note that such accuracy is not quite related to these parameters.

6.4 Summary

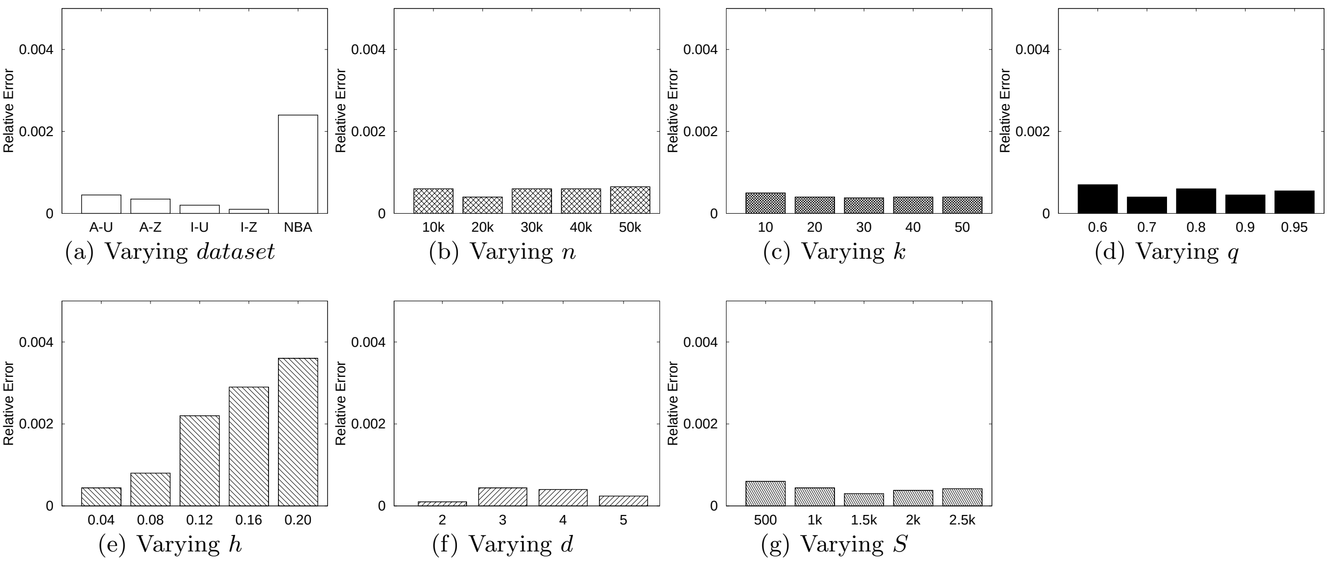
Both of EXACT and *RAND* are efficient when k is not very large (a typical case for a top-k query), the average



(a) Varying *dataset* (b) Varying n (c) Varying k (d) Varying q

(e) Varying h (f) Varying d (g) Varying S

Fig. 16 Relative error with respect to different parameters



(a) Varying *dataset* (b) Varying n (c) Varying k (d) Varying q

(e) Varying h (f) Varying d (g) Varying S

Fig. 17 Relative error of score with respect to different parameters

MBB size of uncertain objects is reasonable (say, upto 20% of the edge length of the data space), and the total data size is about a few millions. Nevertheless, our randomized algorithm is much more efficient and is also very scalable against dimensionality, k values, data sizes, and object MBB sizes; it is also highly accurate when the sample size reaches 1000.

7 Discussions

The techniques developed in this paper can be immediately used to the second model of the problem of top-k probabilistic dominating queries; that is, given a threshold l and a set of uncertain objects $\mathcal{U}$, find k uncertain objects with the highest $P_{\geq l}(U)$ where each $U \in \mathcal{U}$ and ties are broken arbitrarily. Recall that $P_{\geq l}(U)$ denotes the probability of U dominating at least l other uncertain objects. We can redefine the pscore of an uncertain object U as $P_{\geq l}(U)$. All of the upper bound techniques and pruning rules in the exact algorithm can be immediately applied. For instance, any uncertain objects with $P_{\geq l}^{upper} \leq q_k$ can be pruned during the computation where q_k denotes the minimal pscore of current top-k uncertain objects. Regarding the random algorithm, we redefine $pscore^r$ of an uncertain object U as $\frac{m_l(U)}{m}$ where m denotes the total number of sampled possible worlds and m_l represents the number of sampled possible worlds in which U dominates at least l other objects. Then, the gCaR-tree and pruning techniques in the random algorithm can be immediately applied as well.

In many applications, the instances of uncertain objects might be correlated with each other. While exact algorithms may be very expensive in processing correlations among a large number of objects, we can draw the sampled possible worlds from the correlated data with Markov Chain Monte Carlo(MCMC) methods [19], including several sampling techniques. For instance, the *Gibbs sampler* can be employed when the *univariate* conditional distributions of the uncertain objects are available. Then our random algorithm can be immediately applied to the sampled possible worlds. Moreover, the accuracy guarantee in the paper holds as long as the sampled possible worlds are independent with each other. However, because of the independence assumption, it is non-trivial to extend our exact algorithm to tackle the problem against dataset with correlations. As a possible future work, we will consider to develop efficient exact algorithm based on the graph model [38,15] which can effectively capture the correlations of the uncertain dataset.

8 Related Work

Top-k Dominating Query in Multi-Dimensional Space. It is firstly investigated by Papadias *et al* in [33] as a variation of skyline queries. To enhance the efficiency, Yiu and Mamoulis [43] propose two techniques based on aR-tree index structure, counting-guided search, and priority-based traversal. The k-dominant skyline query is studied by Chan *et al* [8] where skylines in a k-dimensional subspace is retrieved.

Uncertain Data Management. Considerable research effort has been put into modeling and managing uncertain data in recent years due to many emerging applications. Sarma *et al* [36] models uncertain data using *possible world semantics* and a prototype of uncertain data management system, *Trio*, is developed by the Stanford Info Lab [2]. Many general issues in modelling and managing uncertain data have been addressed in [2,3]. Managing correlated uncertain data is investigated by Sen and Deshpande in [37]. Very recently Dalvi and Suciu [14] have shown that the problem of evaluating conjunctive probabilistic queries is either *PTIME* or *#P-complete*.

A number of problems in querying uncertain data have also been studied, such as indexing [41], similarity join [26], nearest neighbor query [27], skyline query [34], clustering [28,30], etc.

Top-k Query Processing over Uncertain Data. Top-k query is important in analyzing uncertain data since it captures the inherent imprecise nature of data. Unlike a top-k query over certain data which returns the k best alternatives according to a ranking function, a top-k query against uncertain data has inherently more sophisticated semantics. Soliman *et al* [39] first relate top-k queries with uncertain data. They define two types of important queries - *U-Topk* and *U-kRank*, as well as develop novel techniques to approach them. Based on novel observations, Yi *et al* [42] significantly improve the efficiency. Hua *et al* [22] investigate the problem of threshold-based probabilistic top-k uncertain objects. Re *et al* [35] deal with query evaluation on probabilistic database and results are ranked according to the probability of satisfying a given query.

To the best of our knowledge, the work presented in this paper is the first one to study top-k dominating queries in uncertain semantics. While the techniques in [43] and the techniques in [23,42] are applicable, our experiments demonstrate that the combination of them is much slower than our techniques even with the help of our pre-ordering technique.

9 Conclusion

In this paper, we formally define a probabilistic threshold top-k dominating query. To process such a query, we firstly propose an exact algorithm. The exact algorithm utilizes novel and efficient pruning techniques based on novel mathematic characterizations. While fairly efficient, it is quite sensitive to sizes of data set, uncertain object sizes, k values, etc. To trade-off between efficiency and accuracy, a randomized algorithm with an accuracy guarantee is proposed together with a new data structure, gCaR-tree; it is much more efficient than the exact algorithm. The efficiency and effectiveness of these two algorithms are extensively investigated in experimental study.

Note that our algorithms are main memory based. It can be immediately extended to external memory computation using warm buffer; that is, keep things in the buffer and use a buffer replacement policy once it is full. We have evaluated the I/O costs for such a paradigm. Moreover, our techniques developed in the paper can be immediately extended to cover the dual problem. That is, given a threshold about the number of objects to be dominated, find top-k objects with the maximum dominating probabilities. Finally, our randomized algorithm can also be immediately extended to continuous cases by sampling PDFs using Monte Carlo sampling [25]. As a possible future work, we will deal with the correlations among objects as discussed in Section 7.

Acknowledgement. We would like to thank the anonymous reviewers for their efforts and valuable comments to help us to improve the presentation of the paper. The work of Xuemin Lin is supported by Australian Research

Council Discovery Grants (DP0987557, DP0881035 and DP0666428) and Google Research Award. Wei Wang's research is supported by ARC Discovery Grants DP0987273 and DP0881779. Jian Pei's research is supported in part by a NSERC Discovery grant and a NSERC Discovery Accelerator Supplement grant.

References

1. S. Abiteboul, P. Kanellakis, and G. Grahne. On the representation and querying sets of possible worlds. In *SIGMOD 1987*.
2. P. Agrawal, O. Benjelloun, A. D. Sarma, C. Hayworth, S. Nabar, T. Sugihara, and J. Widom. Trio: a system for data, uncertainty, and lineage. In *VLDB 2006*.
3. L. Antova, C. Koch, and D. Olteanu. 10^{10^6} worlds and beyond: Efficient representation and processing of incomplete information. In *ICDE 2007*.
4. D. Barbara, H. Garcia-Molina, and D. Porter. The management of probabilistic data. *IEEE TKDE*, 4(5):487–502, 1992.
5. G. Bekales, M. A. Soliman, and I. F. Ilyas. Efficient search for the top-k probable nearest neighbors in uncertain databases. In *VLDB 2008*.
6. S. Börzsönyi, D. Kossmann, and K. Stocker. The skyline operator. In *ICDE 2001*.
7. T. Brinkhoff, H.-P. Kriegel, and B. Seeger. Efficient processing of spatial joins using r-trees. In *SIGMOD 1993*.
8. C. Chan, H. Jagadish, K.-L. Tan, A. K. Tung, and Z. Zhang. Finding k-dominant skylines in high dimensional space. In *SIGMOD 2006*.
9. R. Cheng, J. Chen, M. F. Mokbel, and C.-Y. Chow. Probabilistic verifiers: Evaluating constrained nearest-neighbor queries over uncertain data. In *ICDE 2008*.
10. R. Cheng, D. V. Kalashnikov, and S. Prabhakar. Evaluating probabilistic queries over imprecise data. In *SIGMOD 2003*.
11. R. Cheng, Y. Xia, S. Prabhakar, R. Shah, and J. S. Vitter. Efficient indexing methods for probabilistic threshold queries over uncertain data. In *VLDB 2004*.
12. T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to algorithms 2nd Edition*. The MIT Press, 2001.
13. N. Dalvi and D. Suciu. Efficient query evaluation on probabilistic databases. In *VLDB 2004*.
14. N. Dalvi and D. Suciu. Management of probabilistic data: foundations and challenges. In *PODS 2007*.
15. N. N. Dalvi and D. Suciu. Management of probabilistic data: foundations and challenges. In *PODS*, 2007.
16. D. Dubhashi and A. Panconesi. *Concentration of measure for the analysis of randomised algorithms, page 12*. http://citeseer.ist.psu.edu/old/ dubhashi98concentration.html, 1998.
17. M. Ester, H.-P. Kriegel, J. Sander, and X. Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *KDD*, 1996.
18. R. Fagin, A. Lotem, and M. Naor. Optimal aggregation algorithms for middleware. *JCSS*, 66:614–656, 2003.
19. W. Gilks, S. Richardson, and D. Spiegelhalter. *Markov chain Monte Carlo in practice*. Chapman & Hall, 1996.
20. O. Goldreich. *Randomized Methods in Computation, Lecture 2*. http://www.wisdom.weizmann.ac.il/˜oded/rnd.html, 2001.
21. A. Guttman. R-trees: A dynamic index structure for spatial searching. In *SIGMOD 1984*.
22. M. Hua, J. Pei, W. Zhang, and X. Lin. Ranking queries on uncertain data: A probabilistic threshold approach. In *SIGMOD 2008*.
23. M. Hua, J. Pei, W. Zhang, and X. Lin. Ranking queries on uncertain data: a probabilistic threshold approach. In *SIGMOD*, 2008.
24. T. Imielinski and W. Lipski. Incomplete information in relational databases. *JACM*, 31(4), 1984.
25. M. H. Kalos and P. A. Whitlock. *Monte Carlo Methods*. Wiley Interscience, 1986.
26. H. P. Kriegel, P. Kunath, M. Pfeifle, and M. Renz. Probabilistic similarity join on uncertain data. In *DASFAA 2006*.
27. H. P. Kriegel, P. Kunath, and M. Renz. Probabilistic nearest-neighbor query on uncertain objects. *DASFAA 2007*.
28. H. P. Kriegel and M. Pfeifle. Density-based clustering of uncertain data. In *KDD 2005*.
29. L. V. S. Lakshmanan, N. Leone, R. Ross, and V. S. Subrahmanian. Probview: a flexible probabilistic database system. *ACM TODS*, 22(3):419–469, 1997.
30. W. K. Ngai, B. Kao, C. K. C. R. Cheng, M. Chau, and K. Y. Yip. Efficient clustering of uncertain data. In *ICDM 2006*.
31. D. Papadias, P. Kalnis, J. Zhang, and Y. Tao. Efficient olap operations in spatial data warehouses. In *SSTD 2001*.
32. D. Papadias, N. Mamoulis, and Y. Theodoridis. Processing and optimization of multiway spatial joins using R-trees. In *PODS 1999*.
33. D. Papadias, Y. Tao, F. Greg, and B. Seeger. Progressive skyline computation in database systems. *ACM TODS*, 30(1):41–82, 2003.
34. J. Pei, B. Jiang, X. Lin, and Y. Yuan. Probabilistic skyline on uncertain data. In *VLDB 2007*.
35. C. Re, N. Dalvi, and D. Suciu. Efficient top-k query evaluation on probabilistic data. In *ICDE 2007*.
36. A. D. Sarma, O. Benjelloun, A. Halevy, and J. Widom. Working models for uncertain data. In *ICDE 2005*.
37. P. Sen and A. Deshpande. Representing and querying correlated tuples in probabilistic databases. In *ICDE 2007*.
38. P. Sen and A. Deshpande. Representing and querying correlated tuples in probabilistic databases. In *ICDE*, 2007.
39. M. A. Soliman, I. F. Ilyas, and K. C. Chang. Top-k query processing in uncertain databases. In *ICDE 2007*.
40. K.-L. Tan, P.-K. Eng, and B. C. Ooi. Efficient progressive skyline computation. In *VLDB 2001*.
41. Y. Tao, R. Cheng, X. Xiao, W. K. Ngai, B. Kao, and S. Prabhakar. Indexing multi-dimensional uncertain data with arbitrary probability density functions. *VLDB 2005*.
42. K. Yi, F. Li, G. Kollios, and D. Srivastava. Efficient processing of top-k queries in uncertain databases with x-relations. *IEEE Trans. Knowl. Data Eng.*, 20(12):1669–1682, 2008.
43. M. L. Yiu and N. Mamoulis. Efficient processing of top-k dominating queries on multi-dimensional data. In *VLDB 2007*.

A Appendix: PROOFS of THEOREMS

A.1 Proof of Theorem 2

Proof We use the possible world semantics to prove the theorem. Without loss of generality, we assume that there are n objects where u does not dominate any instance from U_i for $l+1 \leq i \leq n$. For each uncertain object U_i with $1 \leq i \leq l$, we divide its instances into 2 groups $U_{i,1}$ and $U_{i,2}$ such that the instances in $U_{i,1}$ are all dominated by u and none of the instances in $U_{i,2}$ is dominated by u. Clearly, $Pr(U_{i,1}) = p_i$ and $Pr(U_{i,2}) = (1 - p_i)$ for $1 \leq i \leq l$.

It can be immediately verified that the possible worlds in each of which u dominates at least λ instances from different objects can be expressed by the union of the following spaces.

$$\Omega_{\geq\lambda} = \bigcup_{\text{follows Condition } \lambda} (\prod_{i=1}^{l} \delta_i \times \prod_{i=l+1}^{n} U_n) \tag{15}$$

Here, Condition λ includes that for $1 \leq i \leq l$, $\delta_i \in \{U_{i,1}, U_{i,2}\}$ and there are at least λ δ_is with the form of $U_{i,1}$. Clearly,

$$Pr(\Omega_{\geq\lambda}) = \sum_{\text{Condition } \lambda} (\prod_{i=1}^{l} Pr(\delta_i) \times \prod_{i=l+1}^{n} Pr(U_i))$$

Now if we modify each instance in $U_{i,2}$ (for $1 \leq i \leq l$) by adding one instance at the position of $MBB^{+}.U_i$ with the occurrence probability $(p_i^{*} - p_i)$ and totally reduce the probabilities of the instances in the original $U_{i,2}$ by $(p_i^{*} - p_i)$; clearly, the total probabilities of $U_{i,2}$ remain unchanged. It can be immediately verified that each possible world from (15) correspond to a possible world after such a modification that dominates at least λ instances from different objects. □

A.2 Proof of Theorem 3

Proof We prove this theorem using the possible world semantics. There are two cases: case 1) u is not an instance of any object, and case 2) u is an instance of the object U_1.

Theorem 3 holds trivially for case 1) since in any possible world where an instance $v \in V$ dominates at least λ other instances, u always dominates at least $\lambda + 1$ instances.

Regarding case 2), for each $v \in U_2$ let $\Omega^{\mathcal{U}-U_1-U_2}_{\geq\lambda,v}$ ($\Omega^{\mathcal{U}-U_1-U_2}_{\geq\lambda-1,v}$) denote the subset of possible worlds from $\prod_{i=3}^{n} U_i$ such that v dominates at least λ ($\lambda - 1$) instances in each possible world. We use $U_{1,1,v}$ to denote the set of instances of U_1, dominated by v, and $U_{1,2,v}$ denotes the set of instances of U_1, not dominated by v.

Consequently, all the possible worlds where an instance $v \in U_2$ dominates at least λ other instances can be represented below.

$$\Omega_v = (U_{1,2,v} \times v \times \Omega^{\mathcal{U}-U_1-U_2}_{\geq\lambda,v}) \cup (U_{1,1,v} \times v \times \Omega^{\mathcal{U}-U_1-U_2}_{\geq\lambda-1,v})$$

Clearly, $Pr(U_{1,2,v}) + Pr(U_{1,1,v}) = 1$ and $\sum_{v \in U_2} Pr(v) = 1$. Note that $\Omega^{\mathcal{U}-U_1-U_2}_{\geq\lambda,v} \subseteq \Omega^{\mathcal{U}-U_1-U_2}_{\geq\lambda-1,v}$ and $Pr(U_2 \times \Omega^{\mathcal{U}-U_1-U_2}_{\geq\lambda-1,v})$ $= Pr(\Omega^{\mathcal{U}-U_1-U_2}_{\geq\lambda-1,v})$ for each $v \in U_2$. Thus,

$$\begin{aligned} \sum_{v \in U_2} Pr(\Omega_v) &\leq \max_{v \in U_2} \{\Omega^{\mathcal{U}-U_1-U_2}_{\geq\lambda-1,v})\} \\ &= \max_{v \in U_2} \{Pr(U_2 \times \Omega^{\mathcal{U}-U_1-U_2}_{\geq\lambda-1,v})\} \end{aligned} \tag{16}$$

Let $v*$ denote the instance in U_2 that makes

$$Pr(U_2 \times \Omega^{\mathcal{U}-U_1-U_2}_{\geq\lambda-1,v}) = \max_{v \in U_2} \{Pr(U_2 \times \Omega^{\mathcal{U}-U_1-U_2}_{\geq\lambda-1,v})\}.$$

Clearly, u dominates at least λ instances in any possible world in $U_2 \times \Omega^{\mathcal{U}-U_1-U_2}_{\geq\lambda,v}$. Therefore, $P_{\geq\lambda}(u) \geq P_{\geq\lambda}(U_2)$. □