k-Selection Query over Uncertain Data

Xingjie Liu[1†] Mao Ye[2†] Jianliang Xu[3‡] Yuan Tian[4†] Wang-Chien Lee[5†]

[†]Department of Computer Science and Engineering, The Pennsylvania State University, University Park, PA 16801, USA
[‡]Department of Computer Science, Hong Kong Baptist University, Kowloon Tong, Hong Kong
{[1]xzl106,[2]mxy177,[4]yxt144,[5]wlee}@cse.psu.edu [3]xujl@comp.hkbu.edu.hk

Abstract. This paper studies a new query on uncertain data, called *k-selection query*. Given an uncertain dataset of N objects, where each object is associated with a preference score and a presence probability, a k-selection query returns k objects such that the expected score of the "best available" objects is maximized. This query is useful in many application domains such as entity web search and decision making. In evaluating k-selection queries, we need to compute the expected best score (EBS) for candidate k-selection sets and search for the optimal selection set with the highest EBS. Those operations are costly due to the extremely large search space. In this paper, we identify several important properties of k-selection queries, including *EBS decomposition*, *query recursion*, and *EBS bounding*. Based upon these properties, we first present a dynamic programming (DP) algorithm that answers the query in $O(k \cdot N)$ time. Further, we propose a *Bounding-and-Pruning* (BP) algorithm, that exploits effective search space pruning strategies to find the optimal selection without accessing all objects. We evaluate the DP and BP algorithms using both synthetic and real data. The results show that the proposed algorithms outperform the baseline approach by several orders of magnitude.

1 Introduction

Data uncertainty is pervasive in our world. A web search engine returns a set of pages to a user, but cannot guarantee all the pages are still available. An on-line advertisement site lists many products with nice discounts, but some of them are already sold out in store. A GPS navigator may display nearest restaurants, but some of them may be already full. In presence of data uncertainty, effective queries that facilitate the retrieval of desired data items are urgently needed. In the past few years, several Top-k query semantics [14, 10, 11, 7] for uncertain data have been proposed, trying to capture the possibly "good" items. However, these proposals do not address a very common problem, i.e., a user is typically only interested in the "best" item that is "available". For example, a used car shopper buys the car of his preference that is still for sale. Based on this observation, in this study, we present a new and novel query operator over uncertain data, namely, *k-selection query*. Given that the uncertain availability of data items is captured as a probability, the k-selection query returns a set of k candidate items, such that the expected score of the best available item in the candidate set is optimized.

To illustrate the k-selection query semantics, let us consider a scenario where John plans to purchase a used car. Given an on-line used car database $D = \{d_1, d_2, \cdots, d_N\}$

where each d_i $(1 \leq i \leq N)$ represents a car and d_i ranks higher than d_{i+1}.[1] A top-k query may help by returning k cars based on John's preference. However, contacting all these sellers to find the best available car is time consuming since popular cars with a good deal may be sold quickly. Assuming that the available probabilities of vehicles can be obtained (e.g., a freshly posted used car has a higher available probability than a car posted weeks ago), a k-selection query on D takes into account the scores and availability of the cars to return an *ordered* list of k candidates that maximizes the expected preference score of the *best available* car. Thus, John can be more efficient in finding the best available car by following the list to contact sellers.

Data Object	Scores	Probability
d_1	4	0.3
d_2	3	0.4
d_3	2	0.8
d_4	1	0.6

(a) Dataset Example

Strategies	Results	Expected Best Score
Top-2 Scores	$\{d_1, d_2\}$	2.04
Weighted Score	$\{d_2, d_3\}$	2.16
2-Selection	$\{d_1, d_3\}$	2.32

(b) Query Result Example

Fig. 1. A 2-Selection Query Example.

Finding the optimal k-selection to maximize the expected preference score of the *best available* object is not trivial. Consider a toy example in Fig. 1(a), where a set of 4 cars $D = \{d_1, d_2, d_3, d_4\}$ along with their available probabilities are shown. Suppose a user is interested in obtaining an ordered set of two candidates, $\{d_i, d_j\}$, where d_i is ranked higher than d_j. Since d_i has a higher preference score than d_j, the best object of choice is d_i as long as it is available. Only if d_i is unavailable while d_j is available, d_j will become the best choice. Based on the above reasoning, we use *expected best* score (which stands for expected score of the best candidate) to measure the goodness of the returned candidates. First, let us consider $\{d_1, d_2\}$, the candidate set obtained based on the highest scores. Its expected best score is $4 \cdot 0.3 + 3 \cdot (1 - 0.3) \cdot 0.4 = 2.04$. Next, consider $\{d_2, d_3\}$ which is obtained based on the highest weighted score. Its expected best score is $3 \cdot 0.4 + 2 \cdot (1 - 0.4) \cdot 0.8 = 2.16$. The above two strategies look reasonable but they do not yield the best selection because the first strategy does not consider the availability while the second strategy does not consider the ranking order in their selecting processes. As shown, the $\{d_1, d_3\}$, returned by the proposed 2-selection query, yields the highest expected best score = $4 \cdot 0.3 + 2 \cdot (1 - 0.3) \cdot 0.8 = 2.32$.

Accordingly, the *expected best score (EBS)* for a selection $S \subseteq D$ can be expressed as in Eq. (1).

$$\texttt{EBS}(S) = \sum_{d_i \in S} f(d_i) \cdot P(d_i \text{ is the best available object in } S) \tag{1}$$

where $f(d_i)$ and $P(d_i)$ denote the preference score and available probability of object d_i, respectively.

Therefore, a k-selection query $Q(k, D)$ aims at returning an ordered subset $S^* \subseteq D$, $|S^*| = k$ such that the EBS of S^* is maximized. S^* can be expressed as shown in Eq. (2):

$$S^* = \arg \max_{S \subseteq D, |S|=k} \texttt{EBS}(S) \tag{2}$$

The k-selection query is a new type of rank queries on uncertain dara that, to our best knowledge, has not been reported in the literature. Evaluating the k-selection query is

[1] For simplicity, we assume that the data items have been sorted in the order of John's preference.

very challenging because the candidate objects can not be selected individually to form the optimal solution. As a result, the search space for optimal k-selection is as large as $\binom{N}{k}$, which is significant as N increases. Efficient algorithms for the k-selection query are needed to tackle the challenge.

The contributions made in this paper are summarized as follows:

- We present a new and novel rank query, called k-selection, for uncertain databases.
- Based on the possible world model for uncertain data, we formalize the presentation of *expected best score (EBS)* and propose decomposing techniques to simplify the calculation of EBS.
- We develop a dynamic programming algorithm that solves the k-selection query over sorted data in $O(k \cdot N)$ time (where N is the dataset size).
- A bounding-and-pruning (BP) algorithm is developed based on the EBS bounds and the relationship in their preference scores. Its computational cost is even lower than the DP algorithm for large datasets.
- We conduct a comprehensive performance evaluation on both the synthetic data and real data. The result demonstrates that the proposed DB and BP algorithms outperform the baseline approach by several orders of magnitudes.

The rest of the paper is organized as follows. In Section 2, we review the existing work and formally formulate the problem. Section 3 addresses the problem by decomposing the EBS calculation and identifying the query recursion. Section 4 introduces the dynamic programming algorithm and the bounding-and-pruning algorithm that efficiently processes the k-selection query. Section 5 reports the result obtained from an extensive set of experiments. Section 6 concludes this paper.

2 Preliminaries

In this section, we first briefly review the previous works related to our study. Then, we present the formal definition of the k-selection problem.

2.1 Related Work

The related work involves two major research areas: 1) uncertain data modeling and 2) top-k query processing on uncertain data.

Uncertain data modeling. When inaccurate and incomplete data are considered in a database system, the first issue is how to model the uncertainty. A popular and general model to describe uncertain data is the *possible world semantic* [1, 13, 9]. Our study in this paper adopts this model. The possible world semantic models the uncertain data as a set of possible instances $D = \{d_1, d_2, \cdots, d_N\}$, and the presence of each instance is associated with a probability $P(d_i)$ $(1 \leq i \leq N)$. The set of possible worlds $\mathcal{PW} = \{W_1, W_2, \cdots, W_M\}$ enumerates all possible combinations of the data instances in D, that may appear at the same time (i.e., in a same possible world). Each possible world W_i has an *appearance probability* which reflects W_i's probability of existence. In this paper, we assume that the appearance of an object is independent from any other objects. Thus, the appearance probability of a possible world W_i can be derived from the membership probabilities of the uncertain objects:

$$P(W_i) = \prod_{d \in W_i} P(d) \cdot \prod_{d \notin W_i} \overline{P(d)} \tag{3}$$

where the first term represents the probability that all the objects belong to W_i exist, and the second term represents the probability that all objects not in W_i do not exist, with $\overline{P(d)} = 1 - P(d)$.

In [9, 8], query processing over independent data presence probabilities is studied, with SQL-like queries on probabilistic databases supported in [8]. Furthermore, since the presence of one object may depend on that of another, this presence dependency is modeled as *generation rules*. The rules define whether an object can present when some others exist and thus model the correlations between data objects. Query processing over correlated objects has been discussed in [17, 16].

Top-k query processing over uncertain data. Following the possible world semantics, lots of queries defined in certain databases are revisited in uncertain scenario, such as Top-k queries [14, 10, 11, 7], nearest neighbor queries [6, 12, 2, 5, 4, 3] and skyline queries [15, 18].Among different queries over uncertain data, top-k queries have received considerable attention [14, 10, 11, 7]. Like the k-selection query, all top-k queries assume a scoring function that assigns a preference score for each object. Because of the data uncertainty, various top-k query semantics and definitions have been explored, including *U-Top$k$* and *U-k$Ranks$* [14], *PT-Top$k$* [10], *PK-Top$k$* [11], and *Expected-k$Ranks$* [7]. However, their goals are essentially different from k-selection query.

The U-Top$k$ introduced in [14] catches the k-object set having the highest accumulated probability of being ranked as top-k objects. Taking the same example in Fig. 1(a), the U-top2 result is $\{d_2, d_3\}$ with top2 probability 0.224. For U-kRanks, the query tries to find the object with the highest probability to be ranked exactly at position $1, 2, \cdots, k$, respectively. Therefore, the object d_3 has the highest probability of being ranked first as well as the second. Thus, a U-2Rank returns the result $\{d_3, d_3\}$. The PT-Top$k$ [10] and PK-Top$k$ [11] define the top-k probabilities for individual objects. Specifically, the top-k probability measures the chance that an object ranks *within* the first k objects. The PT-Top$k$ returns the objects having top-k probability no less than a threshold, and PK-Top$k$ returns the k objects having the largest top-k probabilities. Based on the top-2 probabilities calculated for each object, we can find that, given a threshold of 0.3, PT-Top2 will return objects $\{d_3, d_2, d_1\}$; and PK-Top2 will return $\{d_3, d_2\}$. Most recently, [7] proposed the expected ranking semantics, where each individual object is sorted by its expected rank over all the possible worlds. Based on this definition, the top-2 objects are $\{d_3, d_2\}$. In summary, although existing top-k queries catch the top scored objects with various semantics, because their optimization problems are essentially different with k-selection, none of them can be used to answer the k-selection query.

2.2 Problem Formulation

Given the possible world model, we consider a k-selection query with some possible world W_i. Assume the selection set is S ($S \subseteq D$), since a user will only pick objects from S, the best available object is from the intersection of S and W_i. The *expected best score (EBS)* of S is therefore defined as follows:

Definition 1. ***Expected Best Score:*** *The EBS of a candidate answer set S, $EBS(S)$, is defined as the expected best score of S over each possible world $W_i \in \mathcal{PW}$:*

$$EBS(S) = \sum_{W_i \bigcap S \neq \emptyset} \max_{d \in W_i \bigcap S} f(d) \cdot P(W_i) \tag{4}$$

Definition 2. ***k*-Selection Query:** *The k-selection query over uncertain dataset D, $Q(k, D)$, is defined as finding the optimal answer set S^* consisting of k objects from D such that the EBS of the k selected objects is maximized (see Eq.(2))*

3 Analysis for *k*-Selection

To process a k-selection query, one straightforward way is to enumerate all possible selection sets, and for each selection set, enumerate all possible worlds to calculate its EBS. Finally, a set with maximum EBS is returned. This solution is clearly inefficient because it involves a lot of unqualified selection sets, and the EBS calculation accesses a large number of possible worlds one by one. To facilitate the EBS calculation and develop efficient query processing algorithm, in this section, we identify a set of useful properties of EBS and the query.

3.1 Expected Best Score (EBS)

One key step to find the optimal k-selection is to reduce the computation of EBS. From Eqn. (4), we can group the possible worlds based on their best available object. In other words, instead of enumerating all the possible worlds to find EBS of a selection S, we enumerate all the data object $d_i^S \in S$ and then accordingly identify those possible worlds that have the best available object as d_i^S. Therefore, continuing from Eqn. (4), we have:

$$\begin{aligned} \texttt{EBS}(S) &= \sum_{i=1}^{k} f(d_i^S) \cdot \sum_{W_i \in \mathcal{PW}} P(W_i \mid f(d_i^S) \text{ is the largest in } W_i \cap S) \\ &= \sum_{i=1}^{k} f(d_i^S) \cdot P(d_i^S) \prod_{j=1}^{i-1} \overline{P(d_j^S)} \end{aligned} \tag{5}$$

The first step of the above equation eliminates the maximum operator of Eqn. (4) by considering each selected objects one by one. Therefore, for each object d_i^S, the probability that d_i^S is the best available object is the sum of possible world probabilities that d_i^S happens to be the available object with the largest score. In the second step, we further simplify the best available probability as subject to two conditions: 1) d_i^S is available; 2) all the objects within S that have a higher score than d_i^S are unavailable. (Note that objects within S are also numbered in score decreasing order.)

3.2 Query Recursion

For a dataset with N objects, there are a total of $\binom{N}{k}$ possible subsets. Thus, to exhaustively enumerate all possible selections is prohibitively expensive for a large dataset. To develop an efficient algorithm to find S^*, we explore some nice properties of the k-selection query under the data independence assumption.

Theorem 1. ***EBS Decomposition:*** *Consider a candidate selection S with objects $\{d_1^S, d_2^S, \cdots, d_k^S\}$ such that d_1^S is the top scored object. We define a partition of selection S as $S = S_{i^-} \cup S_{(i+1)^+}$, with $S_{i^-} = \{d_1^S, d_2^S, \cdots, d_i^S\}$ and $S_{(i+1)^+} = \{d_{i+1}^S, d_{i+2}^S, \cdots, d_k^S\}$. The expected best score of S is decomposable as follows:*

$$EBS(S) = EBS(S_{i^-}) + \prod_{j=1}^{i} \overline{P(d_j^S)} \cdot EBS(S_{(i+1)^+}). \tag{6}$$

Proof. From Eq. (5), we have:

$$\texttt{EBS}(S) = \sum_{j=1}^{i}\prod_{l=1}^{j-1}\overline{P(d_l^S)}P(d_j^S)\cdot f(d_j^S) + \prod_{l=1}^{i}\overline{P(d_l^S)}\cdot\sum_{j=i+1}^{k}\prod_{l=i+1}^{j-1}\overline{P(d_l^S)}P(d_j^S)\cdot f(d_j^S)$$
$$= \texttt{EBS}(S_{i^-}) + \prod_{l=1}^{i}\overline{P(d_l^S)}\cdot\texttt{EBS}(S_{(i+1)^+}).$$

From Theorem 1, we find that any partition of S can split its EBS into a linear relation as $b_0 + b_1 x$, with b_0 and b_1 only depending on the head partial selection of S_{i^-}. And the tail selection $S_{(i+1)^+}$ would affect the x value.

Theorem 2. ***Query Recursion:*** *For any dataset sorted in descending order of the preference score, $D_{i^+} = \{d_i, d_{i+1}, \cdots, d_N\}$, the optimal k-selection set has the maximum EBS as $\texttt{Opt}(k, D_{i^+})$. Then, the optimal EBS can be derived recursively as follows:*

$$\texttt{Opt}(k, D_{i^+}) = \max\begin{cases} P(d_i)f(d_i) + \overline{P(d_i)}\cdot\texttt{Opt}(k-1, D_{(i+1)^+}), \\ \texttt{Opt}(k, D_{(i+1)^+}). \end{cases} \tag{7}$$

Proof. Consider the optimal answer set S^* for k-selection query over D_{i^+} and $\forall d_i$, it is either included in S^* or not. If $d_i \in S^*$, then because the slope of Eq. (6) is non-negative, S_{i+1^+} must also be maximized. By Theorem 1, the corresponding EBS for S^* is $P(d_i)f(d_i) + \overline{P(d_i)}\cdot\texttt{Opt}(k-1, D_{(i+1)^+})$. Similarly, if $d_i \notin S^*$, then S^* must also be the optimal set of k-selection query for $D_{(i+1)^+}$, with EBS $\texttt{Opt}(k, D_{(i+1)^+})$. Thus, the EBS of S^* takes the maximum of the above two cases, as in Eq. (7).

Theorem 2 unleashes the recursion of the k-selection query. Armed with this recursion, we can reduce any k-selection query on D_{i^+} to queries with equal to or smaller than k over a smaller data set $D_{(i+1)^+}$.

3.3 Bounding Property

The above recursion property helps us to relate the k-selection query with smaller scale queries. It also indicates the dependency of the query to smaller scale queries. In other words, to find the optimal selection for query $Q(k, D_{i^+})$, all the queries with smaller k and fewer objects than D_{i^+} have to be solved. Since there are still many of these queries, we further explore the bounding property of the k-selection query.

Theorem 3. ***EBS Bounding of Optimal Selection:*** *For any dataset D_{i^+} with d_i as the top object, the optimal EBS of a k-selection query, $\texttt{Opt}(k, D_{i^+}), k > 0$, is bounded by $[P(d_i)f(d_i), f(d_i)]$.*

Proof. Because all the object scores are positive, any $\texttt{Opt}(k-1, D_{(i+1)^+})$ is non-negative. Therefore, from the first case in recursion Eq. (7), we must have $\texttt{Opt}(k, D_{i^+}) \geq P(d_i)f(d_i)$. Furthermore, because $f(d_i)$ is the maximum score in D_{i^+}, it is also the bound of the maximum score in any set $S \subseteq D_{i^+}$. Thus, based on Eq. (4), $\texttt{Opt}(k, D_{i^+}) \leq \max_{d\in S} f(d)\cdot\sum_{W_i \bigcap S\neq\emptyset} P(W_i) \leq \max_{d\in S} f(d) = f(d_i)$.

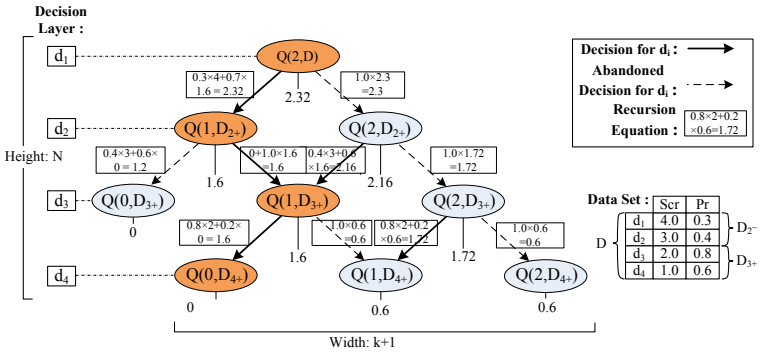


Fig. 2. Example of Dynamic Programming Algorithm (k=2, N=4).

4 Query Processing Algorithms

In the previous section, we analyze a set of properties for k-selection queries. Now we present two efficient k-selection processing algorithms based on these properties.

4.1 Dynamic Programming (DP) Algorithm

According to Theorem 2, the k-selection query can be decomposed into queries with smaller dataset size and query size. Specifically, a query $Q(k, D_{i^+})$ can be answered in constant time if the sub-queries $Q(k-1, D_{(i+1)^+})$ and $Q(k, D_{(i+1)^+})$ are solved. Thus, if we link these sub-queries as $Q(k, D_{i^+})$'s children, an acyclic recursion graph can be constructed. A sample recursion graph for a 2-selection query over 4 data objects is shown in Fig. 2, with the root node representing $Q(k, D)$. In this graph, there are two scenarios in which the query can be trivially solved: 1) the query size $k = 0$, thus the optimal EBS is also 0 because no object can be selected; 2) the remaining dataset size is 1 (i.e., D_{N^+}), which means that the only object must be chosen to maximize the EBS for query size $k > 0$. We call these queries the *base-case queries*. Consider these base-case queries as leaves, the acyclic graph is similar to a tree with height N and width $k+1$ (Fig. 2). Furthermore, the recursion graph can be structured into layers, with the root query on layer 1 and $Q(t, D_{i^+})$ $(0 \leq t \leq k)$ queries on layer i. Based on Theorem 2, all the sub-queries in layer i need to decide whether to accept or reject d_i.

Since the evaluation of each sub-query relies on the results of its descendants in the recursion graph, a dynamic programming algorithm is developed to process the queries in a bottom-up fashion in Algorithm 1. Firstly, two types of base-case queries are initialized in line 1. Then, the algorithm recursively determines the optimal selection for each subproblem from bottom up, with queries for smaller datasets evaluated first (lines 2 through 8). For each sub-query $Q(t, D_{i^+})$, the variable `accept` gets the optimal EBS assuming that d_i is included; otherwise, `reject` stores the EBS when d_i is excluded. By comparing `accept` and `reject`, the choice of whether to include d_i is stored in the variable `trace`(t, D_{i^+}). Finally, the algorithm traces back the `trace` array to find out all accepted objects to obtain the optimal selection (lines 9 through 11). It is not difficult to see that the running time of the dynamic programming algorithm is $O(k \cdot N)$ to traverse the entire recursion graph.

Algorithm 1: Dynamic Programming (DP) Algorithm

Input: Dataset D, query size k
Output: Optimal subset S^* for k-selection over D

Initialize $\texttt{Opt}(0, D_{(0:N)^+}) \leftarrow 0$, and $\texttt{Opt}(1:k, D_{N^+}) \leftarrow P(d_N)f(d_N) + 0$;
for *layer* $i \leftarrow N-1, N-2 \cdots 1$ **do**
  for *query size* $t \leftarrow 1, 2 \cdots k$ **do**
    $\texttt{accept} \leftarrow P(d_i)f(d_i) + \overline{P(d_i)} \cdot \texttt{Opt}(t-1, D_{(i+1)^+})$;
    $\texttt{reject} \leftarrow \texttt{Opt}(t, D_{(i+1)^+})$;
    $\texttt{Opt}(t, D_{i^+}) \leftarrow \max(\texttt{accept}, \texttt{reject})$;
    $\texttt{trace}(t, D_{i^+}) \leftarrow \texttt{accept} > \texttt{reject}$;
Initialize optimal subset $S^* \leftarrow \emptyset$, query size $t \leftarrow k$;
for *layer* $i \leftarrow 1, 2 \cdots N$ **do**
  if $\texttt{trace}(t, D_{i^+}) =$ *true* **then**
    $S^* \leftarrow S^* \cup \{d_i\}, t \leftarrow t-1$;
return $S^*, \texttt{Opt}(k, D)$

Take Fig. 2 as an example. At the beginning, the EBS values for the layer 4 nodes are initialized to $0, 0.6, 0.6$, respectively. Then, the recursion procedure starts from layer 3. Consider the sub-query $Q(1, D_{3^+})$ for instance, the left edge from the node represents the case of accepting d_3, with EBS as $0.8 \times 2 + 0.2 \times \texttt{Opt}(0, D_{4^+}) = 1.6$; the right edge represents the choice of rejecting d_3, and the corresponding EBS is $1.0 \times \texttt{Opt}(1, D_{4^+}) = 0.6$. Since accepting d_3 leads to a higher EBS, d_3 will be included in the optimal set for the sub-query $Q(1, D_{3^+})$ with solid edge. After the recursion procedure completes, all nodes get their optimal EBS values and decision edge identified. (d_i is selected if the solid edge goes left; otherwise to right). The optimal selection for the root query $Q(2, D)$ can be found by tracing back the decision result of each relevant node. For our running example, the decision path is $Q(2, D) \rightarrow Q(1, D_{2^+}) \rightarrow Q(1, D_{3^+}) \rightarrow Q(0, D_{4^+})$, and the corresponding decisions along the path are $\langle \texttt{accept}, \texttt{reject}, \texttt{accept} \rangle$ indicating $S^* = \{d_1, d_3\}$.

4.2 Bounding and Pruning Heuristics

The dynamic programming algorithm proposed in the last subsection needs to access all the data objects to find the optimal k-selection. However, the optimal k-selection, after all, tends to include those objects with higher scores. This intuition leads us to consider an algorithm that can stop without solving all the sub-queries.

However, for any sub-query $Q(t, D_{i^+})$ ($0 \leq t \leq k$), finding pruning rules to stop solving it is not trivial. This is because any of its ancestors above layer i (including the root query) counts on the exact value of $\texttt{Opt}(t, D_{i^+})$ to make selection decisions. Thus, to develop efficient pruning heuristics, we start by investigating the relation between the subproblem $Q(t, D_{i^+})$ and the root k-selection query $Q(k, D)$. Considering a dataset partition for D as $D_{i^-} = \{d_1, d_2, \cdots, d_i\}$ and $D_{(i+1)^+} = \{d_{i+1}, d_{i+2}, \cdots, d_N\}$, then a k-selection query over D is also partitioned by selecting t objects from D_{i^-} and $k-t$ objects from $D_{(i+1)^+}$. We define conditional k-selection queries as follows:

Definition 3. ***Conditional k-Selection****: $Q(k, D \mid t, D_{i^-})$ is defined as a conditional k-selection query over D, by choosing t objects from D_{i^-} and the other $k-t$ objects from $D_{(i+1)^+}$. The EBS of the entire selection is maximized as $\texttt{Opt}(k, D \mid t, D_{i^-})$.*

Clearly, the conditional k-selection query is sub-optimal to $Q(k, D)$ because it is restricted to the condition that exactly t objects are selected from D_{i^-}. But, the global optimal k-selection can be found by solving a group of conditional k-selection queries.

To find the optimal conditional selection query $\texttt{Opt}(k, D \mid t, D_{i^-})$, t objects S_{t^-} are chosen from D_{i^-} (hereafter S_{t^-} is called *head selection*); the remaining $k-t$ objects $S_{(t+1)^+}$ will be from $D_{(i+1)^+}$ (hereafter $S_{(t+1)^+}$ is called *tail selection*). Recall from Theorem 1, the EBS of $S = S_{t^-} \cup S_{(t+1)^+}$ is a linear function as $b_0 + b_1 x$, where the intercept and slope depend on head selection ($b_0 = \texttt{EBS}(S_{t^-})$, $b_1 = \prod_{d \in S_{t^-}} \overline{P(d)}$); and x is the EBS of tail selection ($x = \texttt{EBS}(S_{(t+1)^+})$). Thus, to find an optimal conditional k-selection, x must be maximized because the slope b_1 is always nonnegative. Furthermore, after x is known, proper head selection shall also be chosen to maximize overall EBS of S.

Since we cannot know the optimal x value without solving $Q(k-t, D_{(i+1)^+})$, but from the bounding property, x's bounding can be found without much effort. According to Theorem 3, the optimal $x = \texttt{Opt}(k-t, D_{(i+1)^+})$ must fall within $[P(d_{i+1})f(d_{i+1}), f(d_{i+1})]$. Combining this value range with all the possible head selection linear functions, we developed two pruning heuristics.

Theorem 4. ***Intra-Selection Pruning:*** *For a head selection S_{t^-} with EBS function represented by $L(S_{t^-}, x) = b_0 + b_1 x$. For any value x within the bounding range $[P(d_{i+1})f(d_{i+1}), f(d_{i+1})]$, if there always exists another head selection $S^a_{t^-}$ having $L(S_{t^-}, x) < L(S^a_{t^-}, x)$, S_{t^-} will not result in the optimal conditional selection.*

Proof. Suppose the optimal tail selection is $S^*_{(t+1)^+}$. By combining the alternative head selection $S^a_{t^-}$, we have the overall EBS as $\texttt{EBS}(S^a_{t^-} \cup S^*_{(t+1)^+}) = L(S^a_{t^-}, x) > L(S_{t^-}, x) = \texttt{EBS}(S_{t^-} \cup S^*_{(t+1)^+})$. Thus, S_{t^-} cannot result in the optimal selection, and is subject to pruning.

Theorem 5. ***Inter-Selection Pruning:****For a head selection S_{t^-} and any value x within the bounding range of $[P(d_{t+1})f(d_{t+1}), f(d_{t+1})]$, if there exists another known k-selection S^a such that $L(S_{t^-}, x) < EBS(S^a)$. Then, the head selection S_{t^-} will not result in the optimal selection.*

Theorem 5 is correct because even with the optimal tail selection $S^*_{(t+1)^+}$, the linear relation of $L(S_{t^-}, x)$ will end up with a lower EBS than $\texttt{EBS}(S^a)$. This is called inter-selection pruning because the pruning selection S^a may come from any other k-selection without following the restriction of choosing t objects from D_{i^-}.

An example illustrating the intra- and inter-selection pruning is shown in Fig. 3, where we consider a conditional query of $Q(2, D \mid 1, D_{3^-})$ with $\binom{3}{1} = 3$ different possible head selections: $\{d_1\}, \{d_2\}$, and $\{d_3\}$. Their EBS linear functions as well as the bounding range of $\texttt{Opt}(1, D_{4^+})$ are shown in Fig. 3. Here, we can see that $L(\{d_3\}, x)$ is always lower than $L(\{d_1\}, x)$ and $L(\{d_2\}, x)$ within the value range of $[P(d_4)f(d_4), f(d_4)]$. Thus, according to intra-selection pruning, the head selection $\{d_3\}$ can be safely discarded. In addition, suppose the EBS of $S^a = \{d_1, d_2\}$ is already computed, as in Fig. 3. We can observe that the EBS function of the head selection $\{d_2\}$ is either lower than $\texttt{EBS}(\{d_1, d_2\})$ or head selection $\{d_1\}$. Therefore, following the inter-selection

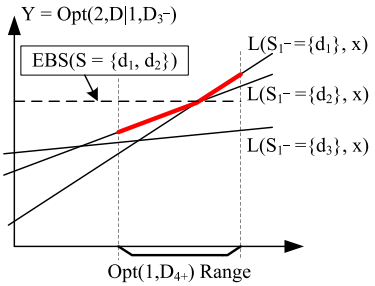


Fig. 3. Example Showing Three Combinations for Partial Dataset D_{4^-}

pruning, $\{d_2\}$ can be safely discarded. After pruning these redundant head selections, the remaining head selections are referred to as effective head selections.

Definition 4. ***Effective Head Selections:*** *Considering a sub-query $Q(k, D \mid t, D_{i^-})$, the effective head selections $\mathbb{ES}(t, i)$ is head selections remained after the intra- and inter-selection pruning.*

Theorem 6. ***Effective Head Selection Recursion:*** *The set of effective head selections $\mathbb{ES}(t, i)$, is a subset of $\mathbb{ES}(t, i-1) \cup (\mathbb{ES}(t-1, i-1) + \{d_i\})$.*[2]

Proof. Consider any effective head selection $S_{t^-} \in \mathbb{ES}(t, i)$, it either contains d_i or not. If $d_i \in S_{t^-}$, then $S_{t^-} - \{d_i\}$ must be an effective head selection for $D_{(i-1)^-}$ because otherwise S_{t^-} will not catch the optimal selection either. Similarly, if $d_i \notin S_{t^-}$, S_{t^-} must be an effective head selection for $D_{(i-1)^-}$.

Theorem 6 points out that we do not need to run a pruning algorithm for all possible head selections from D_{i^-} all the times. Instead, it is enough to consider the effective head selections from its predecessors and merging them and apply pruning rules. This recursion significantly reduces the computation of head selection pruning, and motivates a top-down bounding and pruning algorithm.

4.3 Bounding and Pruning (BP) Algorithm

Having introduced the pruning heuristics, now we present the top-down bounding and pruning algorithm to solve the k-selection query (see Algorithm 2)

The main frame of this algorithm is similar to the dynamic programming algorithm 1, except that the sub-queries here are accessed in a top-down fashion. For each sub-query, the algorithm first determines whether the sub-query $Q(k-t, D_{i^+})$ is a base-case query (line 7). If so, the optimal EBS for the tail selection $\texttt{Opt}(k-t, D_{i^+})$ is trivially solved (line 9,10). This result can be combined with every effective head selection to find the exact optimal conditional selection EBS. If this conditional selection is better, then the best found k-selection is updated in line 11. For those sub-queries that cannot be trivially solved, we only use constant time to obtain its EBS value bound of $\texttt{Opt}(k-t, D_{i^+})$ in line 13, then all the successors' *effective* head selections are collected and filtered using inter-/intra-selection pruning rules (Theorem 4, 5, 6). Consequently, if no effective head selection is left, the scenario is captured in line 4 and the entire algorithm terminates, asserting the found best k-selection as the global optimal.

Algorithm 2: Bounding and Pruning (BP) Algorithm

Input: Dataset D, query size k
Output: Optimal Subset S^* for k-selection over D

Initialize effective head selection $\mathbb{ES}(0:k, 1:N) = \emptyset, \mathbb{ES}(0,1) = \{\emptyset\}$;
Initialize found best $k$selection as `bestEMS` $\leftarrow 0$, $S^* \leftarrow \emptyset$;
for *layer* $i \leftarrow 1, 2 \cdots N$ **do**
  if $\forall\, \mathbb{ES}(0:k, i) = \emptyset$ **then** `/* No effective head selections. */`
    break;
  for *query size* $t \leftarrow k, k-1 \cdots 0$ **and** $\mathbb{ES}(t, i) \neq \emptyset$ **do**
    if $t = k$ **or** $i = N$ **then** `/* Tail selection basecases. */`
      for $\forall S_{t^-} \in \mathbb{ES}(t, i)$ **do**
        if $t = k$ **then** `EBS`$(S) \leftarrow$ `EBS`(S_{t^-});
        if $i = N$ **then** `EBS`$(S) \leftarrow$ `EBS`$(S_{t^-}) + \prod_{d \in S_{t^-}} \overline{P(d)} \cdot P(d_i) f(d_i)$;
        if *EBS*$(S) >$ *bestEMS* **then** `bestEMS` $\leftarrow$ `EBS`(S) and $S^* \leftarrow S$
    else `/* Inter-and intra-selection pruning */`
      `TailBound` $\leftarrow [p(d_i) f(d_i), f(d_i)]$;
      $\mathbb{ES}(t, i) \leftarrow$ MERGEPRUNE$\big(\mathbb{ES}(t, i-1), \mathbb{ES}(t-1, i-1)\big)$;
return S^*, *bestEMS*

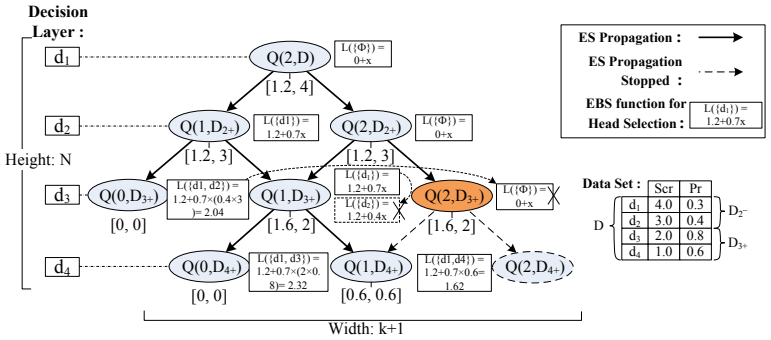


Fig. 4. Example for Bounding and Pruning Algorithm (k=2, N=4).

A running example for the bounding and pruning algorithm is shown in Fig. 4. The EBS bounding range for each tail selection is shown under each node in the figure (e.g. $[1.2, 3]$ for sub-query $Q(2, D_{2^+})$ because $f(d_2) * P(d_2) = 1.2$ and $f(d_2) = 3$). For the head selection S_{t^-}, the root query $Q(2, D)$ initializes its effective head selection as $\mathbb{ES}(0,1) = \{\emptyset\}$, and then propagates it to its child nodes $Q(1, D_{2^+})$ and $Q(2, D_{2^+})$ with $\emptyset \cup \{d_1\}$ and $\emptyset$, respectively. The pruning operation becomes effective on layer 3. For the sub-query $Q(1, D_{3^+})$, the before-pruning head selections are $\{d_1\}$ and $\{d_2\}$. Since these two head selections have the EBS functions as $1.2+0.7x$ and $1.5+0.5x$, respectively, and given tail selection EBS range as $[1.6, 2]$, the head selection $\{d_1\}$ always has a higher EBS (Theorem 4). Thus, $\{d_2\}$ is pruned according to intra-selection pruning. The inter-pruning happens at the node $Q(2, D_{3^+})$. Since the only head selection for this node is $\emptyset$ with EBS function $0 + 1x$ and tail selection EBS range $[1.6, 2]$. But at the time, node $Q(0, D_{3^+})$ already found a conditional 2-selection $S = \{d_1, d_2\}$ with `bestEMS` $= 2.04$. This 2-selection is always superior to $Q(2, D \mid 2, D_{3^+})$ because the maximum value of $0+1x$ over $[1.6, 2]$ is only 2. Thus, after this only head selection $\emptyset$ is

[2] The "+" means that d_i is added to each head selection in $\mathbb{ES}(t-1, i-1)$.

pruned, the node has no remaining head selection propagated. Therefore, its child node $Q(2, D_{4+})$ is ignored during the next layer's processing for empty effective front selection. The stopping node is highlighted in orange in Figure 4, and the optimal selection is actually found at node $Q(0, D_{4+})$ with its head selection $\{d_1, d_3\}$.

The bound and pruning algorithm is guaranteed to find the optimal k-selection because all the head selections are kept until it is guaranteed not to catch the optimal selection. And solving the tail selection has always been postponed by using the bounding properties until it is trivial base case.

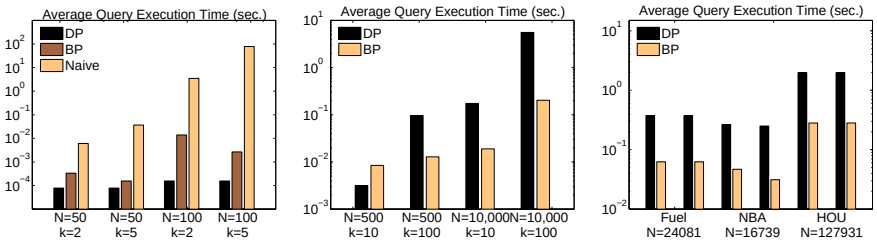
5 Performance Evaluation

To test k-selection queries, we use both synthetic data and real-world dataset. Real datasets include three data sets named FUEL, NBA and HOU, respectively. FUEL is a 24k 6-dimensional dataset, in which each point stands for the performance of a vehicle. NBA contains around 17k 13-dimensional data points corresponding to the statistics of NBA players' performance in 13 aspects. And HOU consists of 127k 6-dimensional data points, each representing the percentage of an American familys annual expense on 6 types of expenditures.[3] In our experiments, we arbitrarily choose two dimensions from each data set for testing, and assign uniform distributed probability between $[0, 1]$ for each data point. These real-world datasets are used to validate the k-selection performance effectiveness for practical applications. On the other hand, we use the synthetic data to learn insights of k-selection queries and proposed algorithms. For synthetic data, the membership probability of an object is modeled by $P(d_i) = \mu + \delta u(i)$, where $\mu > 0, \delta > 0$, and $u(i)$ uniformly distributed between $[-0.5, 0.5]$. Thus, with default value $\mu = 0.5$ and $\delta = 1$, $P(d_i)$ is a uniform distribution between $[0, 1]$. To generate the ranking score for each object, we set $f(d_i) = 0.5 + \beta u(i) + (1 - |\beta|)u'(i)$, with $u(i)$ the same random variable as in $P(d_i)$ but $u'(i)$ another identical independent random variable. Therefore, β models the *covariance* between $f(d_i)$ and $P(d_i)$ as $\eta = \mathtt{Cov}(f(d_i), P(d_i)) = \delta\beta$. In addition to data modeling, we model the data access I/Os by concerning the object size. We set the page size at 4 KB. Therefore assuming one object occupies θ bytes, it will need one I/O operation every $4000/\theta$ object access. The experiment parameters are summarized in Figure 5.All the experiments are implemented and conducted on a Window Server with Intel Xeon 3.2 GHz CPU and 4 GB RAM. The results presented in this section are averaged over 100 independent runs.

Parameter	Setting	Default
Dataset Size (N)	$100 \sim 100,000$	$10,000$
Query Selection Size (k)	$1 \sim 1,000$	100
Probability Mean (μ)	$0.2 \sim 0.8$	0.5
Probability Range (δ)	$0.2 \sim 1$	1
Probability Score Covariance (η)	$-0.8 \sim 0.8$	0
Object Size (θ)	$10 \sim 1000$	100

Fig. 5. Experiment Parameters

[3] Those datasets are collected from www.nba.com, www.ipums.org and www.fueleconomy.gov

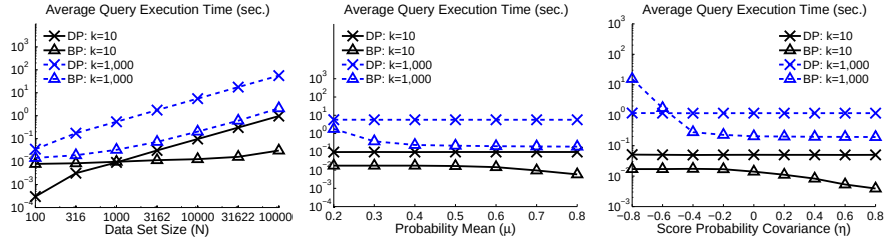


(a) Synthetic Data (b) Subset of Real Data (c) Real Data

Fig. 6. Algorithm Performance

We evaluate the performance of different k-selection algorithms, including dynamic programming algorithm (DP), the bounding and pruning algorithm (BP), and a naive algorithm that examines all $\binom{N}{k}$ candidate selections. Since the naive algorithm cannot scale up to a large dataset, we first compare these algorithms under small datasets. As shown in Fig. 6(a), with 100 objects and 5-selection, the execution time of the naive algorithm is already raised to 100 seconds, which is 10^6 times to the DP algorithm, and 10^5 times to the BP algorithm. For these small datasets, we also find that the BP has a worse performance than the DP algorithm. This is because building a small k by N recursion graph for DP is fast, but the pruning overhead for BP is relatively costly while not much objects can be pruned for a small dataset.

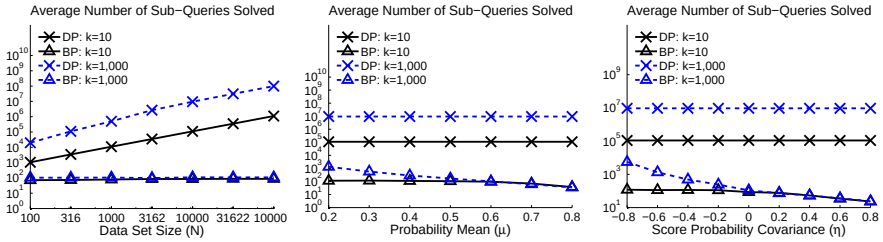
We now proceed to compare the performance of DP and BP under larger datasets. Fig. 6(b) and (c) shows the results using real data. In Fig. 6(b), we find DP is better than BP for a small dataset, which is consistent with what we observed in Fig. 6(a). However, for all other large dataset settings, BP performs much better than DP because of its effective pruning: most of the sub-queries are pruned and left out of computation.



(a) Dataset Size (N) (b) Probability Mean (μ) (c) Score Prob. Covariance (η)

Fig. 7. Query Response Time under Synthetic Data

To gain more insight into these two algorithms, we further evaluate them using synthetic data under different workload settings. Fig. 7(a) plots the performance results for two series of tests ($k = 10$ and $k = 1,000$) as the dataset size varies from 100 to 10,000. For both of the algorithms, it is found that the query execution time increases with increasing the dataset size or the selection size. This is because the larger is N or k, more sub-queries need to be solved before finding the optimal selections (see Fig. 8(a)). Comparing these two algorithms, again, only when the selection size is small and the dataset size is small, DP outperforms BP. On the other hand, BP outperforms DP by more than an order of magnitude for most of the larger-scale cases. Next, we examine



(a) Dataset Size (N) (b) Probability Mean (μ) (c) Score Prob. Covariance (η)

Fig. 8. Number of Sub-Queries Solved under Synthetic Data

the performance of the two algorithms under different data distribution settings. As for the setting of the membership probabilities, Fig. 7(b) shows that the DP algorithm, again, exhibits a similar performance with various settings. However, the BP algorithm shows a significant decrease in cost when the probability mean is high. The reason is that with a higher probability mean, the optimal k-selection favors to include high-score objects such that the inter-selection pruning could terminate the BP algorithm earlier (as observed in Fig. 8(b)). In Fig. 7(c), a similar trend is found when the score and probability covariance are increased. This is because when the score and probability are correlated, many high-score objects will be selected, thereby making the BP algorithm to explore less objects before termination.

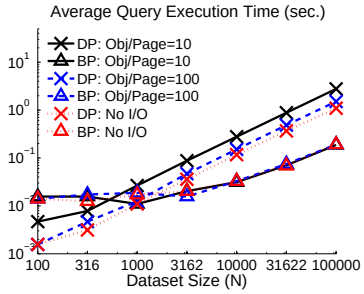


Fig. 9. Impact of Object Size under Various Dataset Sizes

Finally, the impact of the object size is shown in Fig. 9. Although both algorithms need to access more data when a larger dataset is concerned, the BP algorithm performs similarly when the object size is varied. This is because BP only accesses a small portion of objects with few I/Os and, hence, even if the object size is large, BP does not suffer from it. On the other hand, however, it is shown that the query execution time of DP increases significantly when the object size increases. This, from another angle, implies that more objects are accessed in DP than BP.

6 Conclusion

In this paper, we introduce a new k-selection query operation over uncertain data, which finds a subset of objects, that yields the highest expected best score (EBS). While the query is very useful for various applications, it may incur very high processing cost due to the extremely large search space. To address this problem, we first analyze the characteristics of the k-selection query and identify a number of properties. Then, we propose two efficient k-selection query processing algorithms. The Dynamic Programming

(DP) algorithm, which employs the EBS decomposition and query recursion properties, evaluates sub-queries in a bottom-up fashion recursively. Bound-and-Pruning (BP) algorithm, however, utilize a linear relation of EBS decomposition and bounding, to efficiently reduce the problem search space. Through a set of comprehensive evaluations, we demonstrate that our proposed algorithms are superior to the naive brute-force approach, and are efficient for on-line applications.

Acknowledgement

This research was supported by the National Science Foundation under Grant No. IIS-0328881, IIS-0534343 and CNS-0626709, and also supported in part by HK RGC grants HKBU211307 and HKBU210808.

References

1. S. Abiteboul, P. Kanellakis, and G. Grahne. On the Representation and Querying of Sets of Possible Worlds. In *Proceedings of SIGMOD'87*.
2. G. Beskales, M. A. Soliman, and I. F. Ilyas. Efficient search for the top-k probable nearest neighbors in uncertain databases. In *Proceedings of VLDB'08*.
3. M. A. Cheema, X. Lin, W. Wang, W. Zhang, and J. Pei. Probabilistic Reverse Nearest Neighbor Queries on Uncertain Data. *TKDE*, 99(1).
4. R. Cheng, L. C. 0002, J. Chen, and X. Xie. Evaluating probability threshold k-nearest-neighbor queries over uncertain data. In *Proceedings of EDBT'09*.
5. R. Cheng, J. Chen, M. F. Mokbel, and C.-Y. Chow. Probabilistic Verifiers: Evaluating Constrained Nearest-Neighbor Queries over Uncertain Data. In *Proceedings of ICDE'08*.
6. R. Cheng, D. V. Kalashnikov, and S. Prabhakar. Querying Imprecise Data in Moving Object Environments. *TKDE*, 16(9).
7. G. Cormode, F. Li, and K. Yi. Semantics of ranking queries for probabilistic data and expected ranks. In *Proceedings of ICDE'09*.
8. N. Dalvi and D. Suciu. Efficient Query Evaluation on Probabilistic Databases. In *Proceedings of VLDB'04*.
9. N. Fuhr and T. Rölleke. A Probabilistic Relational Algebra for the Integration of Information Retrieval and Database Systems. *ACM Transaction on Information System*, 15(1).
10. M. Hua, J. Pei, W. Zhang, and X. Lin. Ranking queries on uncertain data: a probabilistic threshold approach. In *Proceedings of SIGMOD'08*.
11. C. Jin, K. Yi, L. Chen, J. X. Yu, and X. Lin. Sliding-window top-k queries on uncertain streams. *Proceedings of the VLDB Endowment*, 1(1).
12. H.-P. Kriegel, P. Kunath, and M. Renz. Probabilistic Nearest-Neighbor Query on Uncertain Objects. In *Proceedings of DSFAA'07*.
13. L. V. S. Lakshmanan, N. Leone, R. Ross, and V. S. Subrahmanian. ProbView: a Flexible Probabilistic Database System. *ACM Transaction on Database System*, 22(3).
14. I. F. I. Mohamed A. Soliman and K. C.-C. Chang. Top-k Query Processing in Uncertain Databases. In *Proceedings of ICDE'07*.
15. J. Pei, B. Jiang, X. Lin, and Y. Yuan. Probabilistic skylines on uncertain data. In *Proceedings of VLDB'07*.
16. S. Prithviraj and A. Deshpande. Representing and Querying Correlated Tuples in Probabilistic Databases. In *Proceedings of ICDE'07*.
17. A. D. Sarma, O. Benjelloun, A. Halevy, and J. Widom. Working Models for Uncertain Data. In *Proceedings of ICDE'06*.
18. W. Zhang, X. Lin, Y. Zhang, W. Wang, and J. X. Yu. Probabilistic Skyline Operator over Sliding Windows. In *Proceedings of ICDE'09*.