Ranking Distributed Probabilistic Data

Feifei Li[1]   Ke Yi[2]   Jeffrey Jestes[1]

[1]Department of Computer Science, FSU
Tallahassee, FL, USA
{lifeifei, jestes}@cs.fsu.edu

[2]Dept of Computer Science & Engineering
HKUST, Hong Kong, China
yike@cse.ust.hk

ABSTRACT

Ranking queries are essential tools to process large amounts of probabilistic data that encode exponentially many possible deterministic instances. In many applications where uncertainty and fuzzy information arise, data are collected from multiple sources in distributed, networked locations, e.g., distributed sensor fields with imprecise measurements, multiple scientific institutes with inconsistency in their scientific data. Due to the network delay and the economic cost associated with communicating large amounts of data over a network, a fundamental problem in these scenarios is to retrieve the global top-k tuples from all distributed sites with minimum communication cost. Using the well-founded notion of the expected rank of each tuple across all possible worlds as the basis of ranking, this work designs both communication- and computation-efficient algorithms for retrieving the top-k tuples with the smallest ranks from distributed sites. Extensive experiments using both synthetic and real data sets confirm the efficiency and superiority of our algorithms over the straightforward approach of forwarding all data to the server.

Categories and Subject Descriptors

H.2.4 [**Information Systems**]: Database Management—*Systems. Subject: Query processing*

General Terms

Algorithms

Keywords

Distributed query processing, probabilistic data, ranking queries, top-k, uncertain databases

1. INTRODUCTION

Data are increasingly stored and processed distributively as a result of the wide deployment of computing infrastructures and the readily available network services [4,10,17,19,24,25,31]. More and more applications collect data from distributed sites and derive results based on the collective view of the data from all sites. Examples include sensor networks, data integration from multiple data sources, and information retrieval from geographically separated data centers. In the aforementioned application domains, it is often very expensive to communicate the data set entirely from each site to the centralized server for processing, due to the large amounts of data available nowadays and the network delay incurred, as well as the economic cost associated with such communication [6]. Fortunately, query semantics in many such applications rarely require reporting every piece of data in the system. Instead, only a fraction of data that are the most relevant to the user's interest will appear in the query results. Typically, a ranking mechanism is implemented and only the top-k records are needed [6,14], e.g., displaying the sensor ids with the k highest temperature readings [32,35], or retrieving the images with the k largest similarity scores to a predefined feature [8].

This observation deems that expensive communication is unnecessary and could be avoided by designing communication efficient algorithms. Indeed, a lot of efforts have been devoted to this subject, from both the database and the networking communities [4,6,10,14,17,24,25,31]. However, none of these works deals with distributed queries on probabilistic data, which has emerged as a principled data type in many applications. Interestingly enough, many cases where uncertainty arises are distributed in nature, e.g., distributed sensor networks with imprecise measurements [13], multiple data sources for information integration based on fuzzy similarity scores [12,27]. Probabilistic data encodes an exponential number of possible instances, and ranking queries, by focusing attention on the most representative records, are arguably more important in such a context. Not surprisingly, top-k and ranking queries on probabilistic data have quickly attracted a lot of interests [11,16,22,23,27,33,34,37]. However, none of these works has addressed the problem in a distributed setting, either. We see that uncertainty and distributed query processing have been each studied quite extensively but separately, despite the fact that the two often arise concurrently in many applications nowadays.

To address this important issue, our focus in this work is to answer top-k queries in a communication-efficient manner on probabilistic data from multiple, distributed sites. Uncertainty opens the gate for many possible semantics with respect to ranking queries, and several definitions have been proposed in the literature [11,16,33,37]. They extend the semantics of ranking queries from certain data and study the

SIGMOD'09, June 29–July 2, 2009, Providence, Rhode Island, USA.
Copyright 2009 ACM 978-1-60558-551-1/09/06 ...$5.00.

problem of ranking tuples when there is an uncertain score attribute for each tuple. Under the *possible world* semantics, every uncertain database can be seen as a succinct encoding of a distribution over possible worlds. Each possible world is a certain relational table on which we can evaluate any traditional query. Different ways of combining the results from individual possible worlds lead to different query semantics. Specifically for ranking queries, the result on a single possible world is the ranking of tuples in that world based on their score values there. (We use the convention that tuples with higher scores have smaller ranks.) The existing top-k definitions [11,16,33,37] differ in the way how they combine all these rankings into a unified final ranking.

In this paper, we adopt the *expected ranks* approach from the most recent work [11] on this subject. Here, for each tuple t, we consider its rank in each of the possible worlds. These (exponentially many) ranks are viewed as an empirical probability distribution of t's ranking in a possible world randomly instantiated from the underlying uncertain database. And by the name, the *expected rank* of t is the expectation of this empirical distribution. To obtain a unified ranking, we simply rank all the tuples using their expected ranks, and accordingly, the top-k query returns the k tuples with the smallest expected ranks.

We adopt this expected ranks approach for two important reasons. First, a tuple's rank in a certain database is a fixed number, while in an uncertain database, this number becomes a random variable. Thus for ranking queries, the distribution of this random variable naturally should be of our central concern. Among all the statistics that characterize a probability distribution, the expectation is arguably the most important one. As an important first step, we should focus on getting the expectation, before considering other statistics such as the variance, median, and quantiles. Second, the expectation of a tuple's rank could serve as the basis for deriving the final ranking or the top-k of all the probabilistic tuples. As shown in [11], such an expected rank based definition possesses a number of essential properties (*exact-k, containment, unique-ranking, value-invariance*, and *stability*), which other ranking definitions do not satisfy.

The focus of uncertain query processing is (1) how to "combine" the query results from all the possible worlds into a meaningful result for the query; and (2) how to process such a combination efficiently without explicitly materializing the exponentially many possible worlds. Additionally, in a distributed environment, we also face the third challenge: (3) how to achieve (1) and (2) with the minimum amount of communication. This work concentrates on retrieving the top-k tuples with the smallest expected ranks from m distributed sites that collectively constitute the uncertain database $\mathcal{D}$. The challenge is to answer these queries both computation- *and* communication-efficiently.

Our contributions. We study ranking queries for distributed probabilistic data. We design communication efficient algorithms for retrieving the top-k tuples with the smallest ranks from distributed sites, with computation overhead also as a major consideration as well. In summary, our contributions are as follows:

- We formalize the problem of distributed ranking queries in probabilistic data (Section 2), and argue that the straightforward solution is communication-expensive.
- We first provide a basic approach using only a tuple's local rank (Section 3). Then, we introduce the sorted-access framework based on expected scores (Section 4). The Markov inequality is first applied, followed by an improvement that formulates a linear programming optimization problem at each site, which results in significantly less communication.
- We next propose the notion of approximate distributions in probabilistic data used for ranking (Section 5) to alleviate computation cost from distributed sites. We present a sampling algorithm that optimally minimizes the total error in the approximation at each site. By transmitting them to the server at the beginning of the algorithm, distributed sites are freed from any further expensive computation. These approximate distributions are used by the server to compute the terminating condition conservatively, so that we can still guarantee that the final top-k results returned by the server are exact and correct. Furthermore, these approximations can be incrementally updated by the server after seeing a tuple from the corresponding distributed site, improving their approximation quality as the algorithm progresses.
- We also extend our algorithms to deal with issues on latency, continuous distributions, scoring function, and multiple attributes (Section 6).
- We present a comprehensive experimental study that confirms the effectiveness of our approach (Section 7).

Finally, we survey the related works (Section 8) before concluding the paper.

2. PROBLEM FORMULATION

Many models for describing uncertain data have been proposed in the literature. Sarma *et al.* [28] describe the main features and compare their properties and descriptive abilities. Each model is basically a way to succinctly encode a probability distribution over a set of *possible worlds*, where each possible world corresponds to a single deterministic data instance. The most expressive approach is to explicitly list each possible world and its associated probability; such a method is referred to as being *complete*, as it can capture an arbitrary distribution of the possible worlds. However, complete models are very costly to describe and manipulate since there can be exponentially many combinations of tuples each generating a distinct possible world [28].

Typically, we are able to make certain independence assumptions, that unless correlations are explicitly described, tuples are assumed to be independent. This maintains the expressiveness of the models at a reasonable level, while keeping computation tractable. We consider the following model that has been used frequently within the database community, e.g., [2,3,9,11,20,23,28] and many others. Without loss of generality, a probabilistic database $\mathcal{D}$ contains simply one relation (table).

Uncertainty data model. The probabilistic database $\mathcal{D}$ is a table of N tuples. Each tuple has one attribute whose value is uncertain (together with other certain attributes). This uncertain attribute has a discrete pdf describing its value distribution. When instantiating this uncertain relation to a certain instance, each tuple draws a value for

tuples	score
t_1	$\{(v_{1,1},p_{1,1}),(v_{1,2},p_{1,2}),\ldots,(v_{1,b_1},p_{1,b_1})\}$
t_2	$\{(v_{2,1},p_{2,1}),\ldots,v_{2,b_2},p_{2,b_2})\}$
$\vdots$	$\vdots$
t_N	$\{(v_{N,1},p_{N,1}),\ldots,(v_{N,b_N},p_{N,b_N})\}$

Figure 1: The uncertainty data model.

tuples	score
t_1	$\{(120,0.8),(62,0.2)\}$
t_2	$\{(103,0.7),(70,0.3)\}$
t_3	$\{(98,1)\}$

world W	$\Pr[W]$
$\{t_1=120,t_2=103,t_3=98\}$	$0.8\times0.7\times1=0.56$
$\{t_1=120,t_3=98,t_2=70\}$	$0.8\times0.3\times1=0.24$
$\{t_2=103,t_3=98,t_1=62\}$	$0.2\times0.7\times1=0.14$
$\{t_3=98,t_2=70,t_1=62\}$	$0.2\times0.3\times1=0.06$

Figure 2: An example of possible worlds.

its uncertain attribute based on the associated pdf and the choice is independent among tuples. This model has many practical applications such as sensor readings [13,20], spatial objects with fuzzy locations [9,23], etc. More important, it is very easy to represent this model using the traditional, relational database, as observed by Antova *et al.* [2]. For ranking queries, the important case is when the uncertain attribute represents the score for the tuple, and we would like to rank the tuples based on this score attribute. Let X_i be the random variable denoting the score of tuple t_i. We assume that X_i has a discrete pdf with bounded size (bounded by b_i). This is a realistic assumption adopted in many practical applications, e.g., [5,7,12]. The general, continuous pdf case is discussed in Section 6 as well as in our experimental study. In this model we are essentially ranking the set of independent random variables $X_1,\ldots,X_N$. In the sequel, we will not distinguish between a tuple t_i and its corresponding random variable X_i. This model is illustrated in Figure 1. For tuple t_i, the score takes the value $v_{i,j}$ with probability $p_{i,j}$ for $1\le j\le b_i$, and for $\forall i$, $b_i\le b$, where b is an upper bound on the size of any pdf.

The possible world semantics. In the above uncertainty model, an uncertain relation $\mathcal{D}$ is instantiated into a *possible world* by taking one value for each tuple's uncertain attribute independently according to its distribution. Denote a possible world as W and the value for t_i's uncertain attribute in W as w_{t_i}. The probability that W occurs is $\Pr[W]=\prod_{j=1}^{N}p_{j,x}$, where x satisfies $v_{j,x}=w_{t_j}$. It is worth mentioning that in this case we always have $\forall W\in\mathcal{W},|W|=N$, where $\mathcal{W}$ is the space of all the possible worlds. The example in Figure 2 illustrates the possible worlds for an uncertain relation in this model.

The ranking definition. As we have argued in Section 1, many definitions for ranking queries in probabilistic data exist. Among them, the expected rank approach is particularly important for the two reasons stated in Section 1. Each tuple t_i has a distribution of its ranks in all possible worlds and this could be viewed as a random variable (describing its rank). The expected rank for t_i is simply the expectation of this random variable. The expected rank is an important statistical value and it offers many nice and essential properties as a basis for deriving the final ranking among tuples [11]. Formally,

Definition 1 (Expected Rank) The rank of a tuple t_i in a possible world W is defined to be the number of tuples whose score is higher than t_i (so the top tuple has rank 0), i.e.,

$$\mathrm{rank}_W(t_i)=|\{t_j\in W|w_{t_j}>w_{t_i}\}|.$$

The expected rank $r(t_i)$ is then defined as:

$$r(t_i)=\sum_{W\in\mathcal{W}}\Pr[W]\cdot\mathrm{rank}_W(t_i) \tag{1}$$

For the example in Figure 2, the expected rank for t_1 is $r(t_1)=0.56\times0+0.24\times0+0.14\times2+0.06\times2=0.4$. Similarly $r(t_2)=1.1$, $r(t_3)=1.5$. So the final ranking is (t_1,t_2,t_3).

2.1 Distributed Top-k in Uncertain Data

Given m distributed sites $S=\{s_1,\ldots,s_m\}$, each holding an uncertain database $\mathcal{D}_i$ with size n_i, and a centralized server H, we denote the tuples in $\mathcal{D}_i$ as $\{t_{i,1},\ldots,t_{i,n_i}\}$ and their corresponding score values as random variables $\{X_{i,1},\ldots,X_{i,n_i}\}$. Extending the notation from Figure 1, $X_{i,j}$'s pdf is $\{(v_{i,j,1},p_{i,j,1}),\ldots,(v_{i,j,b_{ij}},p_{i,j,b_{ij}})\}$. We would like to report at H the top-k tuples with the lowest $r(t_{i,j})$'s as in Definition 1 among all tuples in the unified uncertain database $\mathcal{D}=\mathcal{D}_1\bigcup\mathcal{D}_2\cdots\bigcup\mathcal{D}_m$ of size $N=\sum_{i=1}^{m}n_i$. The main objective is to minimize the total communication cost in computing the top-k list, which is the same for many problems on distributed data [4,6].

The straightforward solution. Obviously, one can always ask all sites to forward their databases to H and solve the problem at H locally. If we have a centralized uncertain database $\mathcal{D}=\{t_1,\ldots,t_N\}$, efficient algorithms exist for computing the expected rank of each tuple $t_i\in\mathcal{D}$ [11]. By Definition 1 and the linearity of expectation, we have $r(t_i)=\sum_{i\neq j}\Pr[X_j>X_i]$. The brute-force algorithm requires $O(N)$ time to compute $r(t_i)$ for one tuple and $O(N^2)$ time to compute the ranks of all tuples. In [11], they have observed that $r(t_i)$ can be written as:

$$r(t_i)=\sum_{\ell=1}^{b_i}p_{i,\ell}\big(q(v_{i,\ell})-\Pr[X_i>v_{i,\ell}]\big), \tag{2}$$

where $q(v)=\sum_j\Pr[X_j>v]$. Let U be the universe of all possible values of X_i, $i=1,\ldots,N$. We have $|U|\le|bN|$. When b is a constant, we have $|U|=O(N)$. Let $\Lambda(v)=\sum_{v_{i,j}=v}p_{i,j}$ for $\forall i,j$, then $q(v)=\sum_{v'\in U\wedge v'>v}\Lambda(v')$. One can pre-compute $q(v)$ for all $v\in U$ with a linear pass over the input after sorting U (summing up $\Lambda(v')$'s for $v'>v$) which can be done in $O(N\log N)$. Following (2), exact computation of the expected rank for a single tuple can now be done in constant time given $q(v)$ for all $v\in U$. The overall cost of this approach to compute expected ranks for all tuples is $O(N\log N)$ (retrieving the top-k has an inferior cost $O(N\log k)$ by maintaining a priority queue of size k). This algorithm is denoted as *A-ERrank* [11].

This approach, however, is communication-expensive. In this case, the total communication cost is $|\mathcal{D}|=\sum_{i=1}^{m}|\mathcal{D}_i|$. This will be the baseline we compare against.

3. SORTED ACCESS ON LOCAL RANK

One common strategy in distributed query processing is to first answer the query within each site individually, and then combine the results together. For our problem, this corresponds to first compute the local ranks of the tuples at the sites they belong to. In this section we present an algorithm following this strategy.

Consider an uncertain database $\mathcal{D}_i$ in a local site s_i and any $t_{i,j} \in \mathcal{D}_i$. We define $r(t_{i,j}, \mathcal{D}_i)$ as the *local rank* of $t_{i,j}$ in $\mathcal{D}_i$:

$$r(t_{ij}, \mathcal{D}_i) = \sum_{W \in \mathcal{W}(\mathcal{D}_i)} \Pr[W] \cdot \text{rank}_W(t_{ij}), \qquad (3)$$

where $\mathcal{W}(\mathcal{D}_i)$ is the space of possible worlds of $\mathcal{D}_i$.

Following the algorithm *A-ERrank*, let the universe of values at the site s_i be U_i. We first compute $q_i(v) = \sum_j \Pr[X_{ij} > v]$ for all $v \in U_i$ in $O(n_i \log n_i)$ time. Then we can efficiently compute the local ranks of all tuples in $\mathcal{D}_i$ using (2), i.e.,

$$r(t_{ij}, \mathcal{D}_i) = \sum_{\ell=1}^{b_{ij}} p_{i,j,\ell}\big(q_i(v_{i,j,\ell}) - \Pr[X_{ij} > v_{i,j,\ell}]\big). \qquad (4)$$

We also extend the local rank definition of $t_{i,j}$ to a $\mathcal{D}_y$ where $y \neq i$. Since $t_{i,j} \notin \mathcal{D}_y$, we define its local rank in $\mathcal{D}_y$ as its rank in $\{t_{ij}\} \bigcup \mathcal{D}_y$. We can calculate $r(t_{ij}, \mathcal{D}_y)$ as

$$\begin{aligned} r(t_{ij}, \mathcal{D}_y) &= \sum_{Y \in \mathcal{D}_y} \Pr[Y > X_{ij}] \\ &= \sum_{Y \in \mathcal{D}_y} \sum_{\ell=1}^{b_{ij}} p_{i,j,\ell} \Pr[Y > v_{i,j,\ell}] \\ &= \sum_{\ell=1}^{b_{ij}} p_{i,j,\ell} \left(\sum_{Y \in \mathcal{D}_y} \Pr[Y > v_{i,j,\ell}] \right) \\ &= \sum_{\ell=1}^{b_{ij}} p_{i,j,\ell} q_y(v_{i,j,\ell}). \qquad (5) \end{aligned}$$

Note that in the last step of the derivation above, we use the fact that $t_{ij} \notin \mathcal{D}_y$, hence, X_{ij} does not contribute to U_y and $q_y(v)$.

An important observation on the (global) expected rank of any tuple t_{ij} is that its expected rank could be calculated accumulatively from all the sites. More precisely, we have the following.

Lemma 1 *The (global) expected rank of t_{ij} is*

$$r(t_{ij}) = \sum_{y=1}^{m} r(t_{ij}, \mathcal{D}_y),$$

where $r(t_{i,j}, \mathcal{D}_y)$ is computed using (4) if $y = i$, or (5) otherwise.

PROOF. First, since $\mathcal{D} = \mathcal{D}_1 \bigcup \mathcal{D}_2 \cdots \bigcup \mathcal{D}_m$, we have $U = U_1 \bigcup U_2 \cdots U_m$. Furthermore, $q(v) = \sum_{i=1}^{m} q_i(v)$ by definition. Then, we have

$$\begin{aligned} \sum_{y=1}^{m} r(t_{ij}, \mathcal{D}_y) &= \sum_{y=1, y \neq i}^{m} \sum_{\ell=1}^{b_{ij}} p_{i,j,\ell} q_y(v_{i,j,\ell}) \\ &\quad + \sum_{\ell=1}^{b_{ij}} p_{i,j,\ell}\big(q_i(v_{i,j,\ell}) - \Pr[X_{ij} > v_{i,j,\ell}]\big) \\ &= \sum_{\ell=1}^{b_{ij}} p_{i,j,\ell}\big(\sum_{y=1}^{m} q_y(v_{i,j,\ell}) - \Pr[X_{ij} > v_{i,j,\ell}]\big) \\ &= \sum_{\ell=1}^{b_{ij}} p_{i,j,\ell}\big(q(v_{i,j,\ell}) - \Pr[X_{ij} > v_{i,j,\ell}]\big) \\ &= r(t_{ij}). \qquad \text{(By equation (2)).} \end{aligned}$$

□

An immediate corollary of Lemma 1 is that any tuple's local rank is a lower bound on its global rank. Formally,

Corollary 1 *For any tuple t_{ij}, $r(t_{ij}) \geq r(t_{ij}, \mathcal{D}_i)$.*

Lemma 1 indicates that by forwarding only the tuple t_{ij} itself from the site s_i to all other sites, we can obtain its final global rank. This naturally leads to the following idea for computing the global top-k at the central server H. We sort the tuples at site s_i based on their local ranks $r(t_{ij}, \mathcal{D}_i)$. Without loss of generality, we assume $r(t_{i1}, \mathcal{D}_i) \leq r(t_{i2}, \mathcal{D}_i) \ldots \leq r(t_{in_i}, \mathcal{D}_i)$ for any site s_i. The central server H accesses tuples from the m sites in the increasing order of their local ranks. More precisely, H maintains a priority queue L of size m in which each site s_i has a representative local rank value and the tuple id that corresponds to that local rank value, i.e., a triple $\langle i, j, r(t_{i,j}, \mathcal{D}_i)\rangle$. The triples in the priority queue are sorted by the local rank value in ascending order. L is initialized by retrieving the first tuple's id and local rank from each site.

In each step, H obtains the first element from L, say $\langle i, j, r(t_{i,j}, \mathcal{D}_i)\rangle$. Then, H asks for tuple t_{ij} from site s_i as well as $r(t_{i,j+1}, \mathcal{D}_i)$, the local rank of the next tuple from s_i. The triple $\langle i, j+1, r(t_{i,j+1}, \mathcal{D}_i)\rangle$ will be inserted into the priority queue L. In order to compute the exact global rank of $t_{i,j}$ that H has just retrieved, H broadcasts $t_{i,j}$ to all sites except s_i and asks each site $\mathcal{D}_y$ to report back the value $r(t_{ij}, \mathcal{D}_y)$ (based on equation (5)). By Lemma 1, H obtains the exact global rank of tuple t_{ij}. This completes a round.

Let the set of tuples seen by H be $\mathcal{D}_H$. H dynamically maintains a priority queue for tuples in $\mathcal{D}_H$ based on their global ranks. In the λ-th round, let the k-th smallest rank from $\mathcal{D}_H$ be r^+_λ. Clearly, the local rank value of any unseen tuples by H from all sites is lower bounded by the head element from L. This in turn lower bounds the global rank value of any unseen tuples in $\mathcal{D} - \mathcal{D}_H$ by Corollary 1. Let r^-_λ be the local rank of the head element of L. It is safe for H to terminate the search as soon as $r^+_\lambda \leq r^-_\lambda$ at some round λ and output the top-k from the current $\mathcal{D}_H$ as the final result. We denote this algorithm as *A-LR*.

4. SORTED ACCESS ON EXPECTED SCORE

Sorted access on local rank has limited pruning power as it simply relies on the next tuple's local rank from each site to lower bound the global rank of any unseen tuple. This is too pessimistic an estimate. This section introduces the framework of sorted access on expected score, which incurs much less communication cost than the basic local rank approach.

4.1 The General Algorithm

The general algorithm in this framework is for H to access tuples from all the sites in the descending order of their expected scores. Specifically, each site sorts its tuples in the decreasing order of the expected score, i.e., for all $1 \le i \le m$ and $1 \le j_1, j_2 \le n_i$, if $j_1 < j_2$, then $\mathbf{E}[X_{ij_1}] \ge \mathbf{E}[X_{ij_2}]$. H maintains a priority queue L of triples $\langle i, j, E[X_{ij}]\rangle$, where the entries are sorted in the descending order of the expected scores. L is initialized by retrieving the first tuple's expected score from each of the m sites. In each round, H pops the head element from L, say $\langle i, j, E(X_{ij})\rangle$, and requests the tuple t_{ij} (and its local rank value $r(t_{ij}, \mathcal{D}_i)$) from site s_i, as well as the expected score of the next tuple at s_i, i.e., $E[X_{i,j+1}]$. Next, H inserts the triple $\langle i, j+1, E[X_{i,j+1}]\rangle$ into L. Let τ be the expected score of the top element from L after this operation. Clearly, τ is an upper bound on the expected score for any unseen tuple.

Similarly to the algorithm *A-LR*, H broadcasts t_{ij} to all sites (except s_i) to get its local ranks and derive the global rank for t_{ij}. H also maintains the priority queue for all tuples in $\mathcal{D}_H$ (the seen tuples by H) based on their global ranks and r^+_λ is similarly defined as in *A-LR* for any round λ. The key issue now is to derive a lower bound r^-_λ for the global rank of any unseen tuple t from $\mathcal{D} - \mathcal{D}_H$. H has the knowledge that $\forall t$ with a random variable X for its score attribute, $E(X) \le \tau$. We will show two methods in the sequel to derive r^-_λ based on τ. Once r^-_λ is calculated, the round λ completes. Then H either continues to the next round or terminates if $r^+_\lambda \le r^-_\lambda$.

4.2 Markov Inequality Based Approach

Given the expectation of a random variable X, the Markov inequality could bound the probability that the value of X is above a certain value. Since τ is an upper bound for the expected score of any unseen tuple t with the score attribute X, we have $E(X) \le \tau$ and for a site s_i:

$$\begin{aligned}
r(t, \mathcal{D}_i) &= \sum_{j=1}^{n_i} \Pr[X_j > X] = n_i - \sum_{j=1}^{n_i} \Pr[X \ge X_j] \\
&= n_i - \sum_{j=1}^{n_i}\sum_{\ell=1}^{b_{ij}} p_{i,j,\ell} \Pr[X > v_{i,j,\ell}] \\
&\ge n_i - \sum_{j=1}^{n_i}\sum_{\ell=1}^{b_{ij}} p_{i,j,\ell} \frac{\mathbf{E}[X]}{v_{i,j,\ell}}. \qquad \text{(Markov Ineq.)} \\
&\ge n_i - \sum_{j=1}^{n_i}\sum_{\ell=1}^{b_{ij}} p_{i,j,\ell} \frac{\tau}{v_{i,j,\ell}} = r^-(t, \mathcal{D}_i). \qquad (6)
\end{aligned}$$

This leads to the next lemma that lower bounds the global rank of any unseen tuple.

Lemma 2 *Let τ be the expected score for the head element from L at round λ. Then, for any unseen tuple t:*

$$r(t) \ge \sum_{i=1}^{m} r^-(t, \mathcal{D}_i) = r^-_\lambda,$$

for $r^-(t, \mathcal{D}_i)$ defined in equation (6).

Proof. By equation (6) and the Lemma 1. □

By Lemma 2 and the general algorithm in Section 4.1, our algorithm could terminate as soon as $r^+_\lambda \le r^-_\lambda$ at some round λ. This is denoted as the algorithm *A-Markov*. Since both n_i and $sm_i = \sum_{i=1}^{n_i}\sum_{\ell=1}^{b_{ij}} \frac{p_{i,j,\ell}}{v_{i,j,\ell}}$ are invariants for different rounds λ's in equation (6), a notable improvement to *A-Markov* is to have each site s_i transmit its n_i and sm_i to H at the beginning of the algorithm, once. Then, at each round λ, $r^-(t, \mathcal{D}_i)$ could be computed locally at H. Note that in order to compute the exact rank of seen tuples and derive r^+_λ as well as producing the final output, the server still needs to broadcast each new incoming tuple to all sites and collect its local ranks.

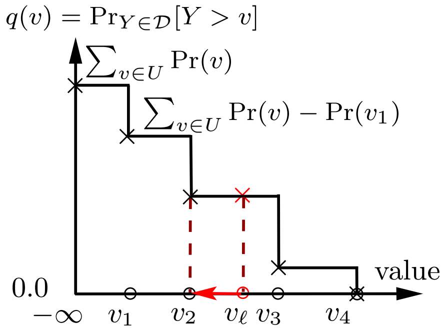

$q(v) = \Pr_{Y \in \mathcal{D}}[Y > v]$
$\sum_{v \in U} \Pr(v)$
$\sum_{v \in U} \Pr(v) - \Pr(v_1)$
value
0.0
$-\infty$
v_1
v_2
v_ℓ
v_3
v_4

Figure 3: Transform values in an unseen tuple X.

4.3 Optimization with Linear Programming

The Markov inequality in general gives a rather loose bound. In this section we give a much more accurate lower bound on $r(t, \mathcal{D}_i)$ for any tuple $t \notin \mathcal{D}_H$. Again let X be the uncertain score of t, and we have $E[X] \le \tau$. Our general idea is to let H send τ to all sites in each round and ask each site to compute a lower bound *locally* on the rank of any unseen tuples (from H's perspective), i.e., a lower bound on $r(t, \mathcal{D}_i)$ for all $\mathcal{D}_i'$s. All sites then send back these lower bounds and H will utilize them to compute the global lower bound on the rank of any unseen tuple, i.e., r^-_λ.

The computation for $r(X, \mathcal{D}_i)$ is different depending on whether $X \in \mathcal{D}_i$ or $X \notin \mathcal{D}_i$. We first describe how to lower bound $r(X, \mathcal{D}_i)$ if $X \notin \mathcal{D}_i$. The problem essentially is, subject to the constraint $E[X] \le \tau$, how to construct the pdf of X such that $r(X, \mathcal{D}_i)$ is minimized. The minimum possible $r(X, \mathcal{D}_i)$ is obviously a lower bound on $r(X, \mathcal{D}_i)$. Let U_i be the universe of possible values taken by tuples in $\mathcal{D}_i$. Suppose the pdf of X is $\Pr[X = v_\ell] = p_\ell, v_1 < v_2 < \cdots < v_\gamma$ for some γ. Let $q_i(v) = \sum_{Y \in \mathcal{D}_i} \Pr[Y > v]$; note that we always have $q_i(-\infty) = \sum_{v \in U_i} \Lambda(v)$ where $\Lambda(v) = \sum_{Y \in \mathcal{D}_i \wedge Y.v_j = v} Y.p_j$, and $q_i(v_L) = 0$ where v_L is the largest value in U_i. Since $X \notin \mathcal{D}_i$, by equation (5), the rank of X in $\mathcal{D}_i$ is

$$r(X, \mathcal{D}_i) = \sum_{Y \in \mathcal{D}_i} \Pr[Y > X] = \sum_{\ell=1}^{\gamma} p_\ell q_i(v_\ell). \qquad (7)$$

Note that $q_i(v)$ is a non-increasing, staircase function with changes at the values of U_i. (We also include $-\infty$ in U_i.) We claim that to minimize $r(X, \mathcal{D}_i)$, we only need to consider values in U_i to form the v_ℓ's, the values used in the pdf of X. Suppose the pdf uses some $v_\ell \notin U_i$. Then we decrease v_ℓ until it hits some value in U_i. During this process $E[X]$ decreases so the constraint $E[X] \le \tau$ is still satisfied. As we decrease v_ℓ while not passing a value in U_i, $q_i(v_\ell)$ does not change (see the example in Figure 3). So (7) stays unchanged during this transformation of the pdf. Note that this transformation will always reduce the number of v_ℓ's that are not in U_i by one. Applying this transformation repeatedly will thus arrive at some pdf of X with all $v_\ell \in U_i$ without changing $r(X, \mathcal{D}_i)$.

Therefore we can assume without loss of generality that the pdf of X has the form $\Pr[X = v_\ell] = p_\ell$ for each $v_\ell \in U_i$, where

$$0 \le p_\ell \le 1, \qquad \ell = 1, \ldots, \gamma = |U_i|; \tag{8}$$

$$p_1 + \cdots + p_\gamma = 1. \tag{9}$$

The constraint $E[X] \le \tau$ becomes

$$p_1 v_1 + \cdots + p_\gamma v_\gamma \le \tau. \tag{10}$$

Therefore, the problem is to minimize (7) subject to the linear constraints (8)(9)(10), which can be solved using linear programming.

Next consider the case $X \in \mathcal{D}_j$ for some j. Then $r(X, \mathcal{D}_j)$ can be computed as in (4), i.e.,

$$\begin{aligned} r(X, \mathcal{D}_j) &= \sum_{\ell=1}^{\gamma} p_\ell (q_j(v_\ell) - \Pr[X > v_\ell]) \\ &= \sum_{\ell=1}^{\gamma} p_\ell q_j(v_\ell) - \sum_{\ell=1}^{\gamma} p_\ell \Pr[X > v_\ell] \\ &\ge \sum_{\ell=1}^{\gamma} p_\ell q_j(v_\ell) - \sum_{\ell=1}^{\gamma} p_\ell = \sum_{\ell=1}^{\gamma} p_\ell q_j(v_\ell) - 1, \end{aligned}$$

where $q_j(v_\ell) = \sum_{Y \in \mathcal{D}_j} \Pr[Y > v_\ell]$. Therefore, we can still minimize as we do for any other $\mathcal{D}_i$, but simply subtract one from the final lower bound on $r(X)$ that we obtain after aggregating the minimum of (7) from all $\mathcal{D}_i$'s. These observations are summarized in the next lemma.

Lemma 3 *Let X be a random unseen tuple by H. For $\forall i \in \{1, \ldots, m\}$, suppose $r^-(X, \mathcal{D}_i)$ is the optimal minimum value from the linear program using (7) as the objective function and (8), (9), (10) as the constraints for each site s_i respectively. Then,*

$$r(X) \ge \sum_{i=1}^{m} r^-(X, \mathcal{D}_i) - 1 = r^-(X).$$

PROOF. By Lemma 1 and the optimal minimum local rank returned by each linear programming formulation. □

This naturally leads to an optimization to the sorted by expected score framework. H maintains the current k'th tuple's rank among all the seen tuples at round λ as r^+_λ and $r^-(X)$ at round λ as r^-_λ. As soon as $r^+_\lambda \le r^-_\lambda$, H stops the search and outputs the current top-k from $\mathcal{D}_H$. This is the *A-LP* algorithm.

5. APPROXIMATE $q(v)$: REDUCING COMPUTATION AT DISTRIBUTED SITES

In many distributed applications (e.g., sensors), the distributed sites often have limited computation power or cannot afford expensive computation due to energy concerns. Algorithm *A-LP* finds the optimal lower bound for the local rank at each site, but at the expense of solving a linear program each round at all sites. This is prohibitive for some applications. This section presents a method to approximate the $q(v)$, which enables the site to shift almost all the computation costs to the server H while still keeping the communication cost low.

5.1 $q^*(v)$: An Approximate $q(v)$

Given a database $\mathcal{D}$ and its value universe U, $q(v)$ represents the aggregated cumulative distribution $\Pr_{X \in \mathcal{D}}[X > v]$, which is a staircase curve (see Figure 3). Let $U = \{v_0, v_1, \ldots, v_\gamma\}$ where $v_0 = -\infty$ and $\gamma = |U|$. Then, $q(v_i) = \sum_{j>i} \Lambda(v_j)$ where $\Lambda(v) = \sum_{X \in \mathcal{D} \wedge X.v_i = v} X.p_i$. $q(v)$ is decided by a set of points $\{(v_0, q(v_0)), (v_1, q(v_1)), \ldots, (v_\gamma, 0)\}$, but it is well defined for any value v even if it is not in U, i.e., for $v \notin U$, $q(v) = \sum_{v_j > v \wedge v_j \in U} \Lambda(v_j)$.

In the *A-LP* approach, the computation of $r^-(X, \mathcal{D}_i)$ only depends on $q_i(v)$. If each site s_i sends its $q_i(v)$ to H at the beginning of the algorithm, then the server could compute $r^-(X, \mathcal{D}_i)$ locally at each round without invoking the linear programming computation at each site in every round. However, $q_i(v)$ is expensive to communicate if $|U_i|$ is large. In the worst case when tuples in $\mathcal{D}_i$ all take distinct values, $|q_i(v)| = |\mathcal{D}_i|$ and this approach degrades to the straightforward solution of forwarding the entire $\mathcal{D}_i$ to H.

This motivates us to consider finding an approximate $q^*(v)$ for a given $q(v)$, such that $|q^*(v)|$ is small and adjustable, and provides a good approximation to $q(v)$. The approximation error ε is naturally defined to be the area enclosed by the approximate and the original curves, i.e.,

$$\varepsilon = \int_{v \in [-\infty, +\infty]} |q(v) - q^*(v)| dv. \tag{11}$$

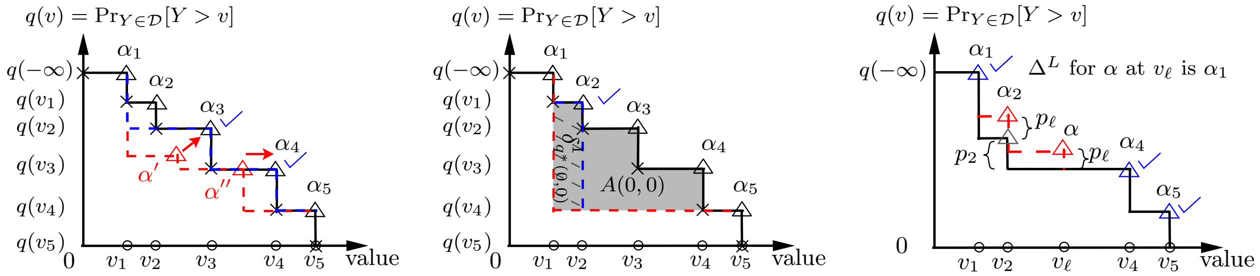
We use such a definition of error because the site does not know beforehand how $q^*(v)$ is going to be used by the server. If we assume that each point on $q(v)$ is equally likely to be probed, then the error ε defined in (11) is exactly the expected error we will encounter.

The approximation $q^*(v)$ is naturally a staircase curve as well. Thus, the problem is, given a tunable parameter $\eta \le |U|$, how to obtain a $q^*(v)$ represented by η points while minimizing ε.

However, not all approximations meet the problem constraint. We need to carefully construct $q^*(v)$ so that given an upper bound value τ on the expected score, the solution from the linear program w.r.t $q^*(v)$ is still a lower bound for $r(X, \mathcal{D})$ for any unknown tuple X with $E(X) \le \tau$. In other words, let $r^*(X, \mathcal{D})$ be the optimal value identified by the linear program formulated with $q^*(v)$, and $r^-(X, \mathcal{D})$ be the optimal value from the linear program using $q(v)$ directly. We must make sure that $r^*(X, \mathcal{D}) \le r^-(X, \mathcal{D})$ so that the lower bounding framework still works, and the returned top-k results are still guaranteed to be exact and correct. Intuitively, we need $q^*(v)$ below $q(v)$ in order to have this guarantee. In what follows, we first present an algorithm that finds the optimal $q^*(v)$ below $q(v)$ that minimizes the error ε, then we show that such a $q^*(v)$ indeed gives the desired guarantee.

There are still many possible choices to construct a $q^*(v)$ as there are infinite number of decreasing staircase curves that are always below $q(v)$ and are decided by η turning points. The next question is, among the many possible choices, which option is the best in minimizing the error ε? The insight is summarized by the next Lemma.

Lemma 4 *Given any $\eta \le |U|$ and $q(v)$, $q^*(v)$, s.t., $q^*(v) \le q(v)$ for $\forall v \in [-\infty, +\infty]$ and $|q^*(v)| = \eta$, the approximation*



(a) Approximate $q(v)$.   (b) Base case and $\delta^{j+1}_{q^*(i,j)}$.   (c) Update $q^*_{\lambda-1}(v)$.

Figure 4: $q^*(v)$: definition, optimal computation and update.

error ε is minimized iff $q^*(v)$'s right-upper corner points only sample points from the set of right-upper corner points in the staircase curve of $q(v)$, i.e., $q^*(v)$ is determined by a subset of $\Delta_{q(v)} = \{\alpha_1 : (v_1, q(v_0)), \ldots, \alpha_\gamma : (v_\gamma, q(v_{\gamma-1})\}$.

PROOF. We concentrate on the necessary condition; the sufficient condition can be argued similarly. We prove this by contradiction. Suppose this is not true, then we have a $q^*(v)$ with the smallest approximation error that contains a right-upper corner point $\alpha' \notin \Delta_{q(v)}$. Since $q^*(v)$ is always below $q(v)$ for $\forall v \in [-\infty, +\infty]$, moving α' towards the $\alpha_i \in \Delta_{q(v)}$ that is the first to its right will only reduce the area enclosed by $q^*(v)$ and $q(v)$. Please see Figure 4(a) for an example where we move α' to α_3 and α'' to α_4. This conflicts with the fact that $q^*(v)$ minimizes the approximation error ε and completes the proof. $\square$

A Corollary for constructing $q^*(v)$ is that the two boundary points from $\Delta_{q(v)}$ should always be sampled, otherwise the error ε could be unbounded.

Corollary 2 *Both $\alpha_1 : (v_1, q(v_0))$ and $\alpha_\gamma : (v_\gamma, q(v_{\gamma-1}))$ from $\Delta_{q(v)}$ should be included in $q^*(v)$'s right corner points in order to have a finite error ε.*

Lemma 4 is illustrated in Figure 4(a) where the $\times$'s are the points (the lower-left corner points) in a $q(v)$ and the $\triangle$'s are the right-upper corner points that should be used to determine $q^*(v)$ in order to minimize the approximation error ε. The dashed line denotes a possible curve for $q^*(v)$ with $\eta = 2$. The two extreme points $(v_1, q(-\infty))$ and $(v_5, q(v_4))$ are automatically included, plus α_3 and α_4. Once we have found the optimal set of α points from $\Delta_{q(v)}$, the $\times$ points in $q^*(v)$ could be easily constructed. As a convention, we do not include the two boundary $\triangle$ points in the budget η.

With Lemma 4 and Corollary 2, we are ready to present the algorithm that obtains an optimal $q^*(v)$ given a budget η. Note that $q^*(v)$ always have the two boundary $\triangle$ points from $q(v)$. The basic idea is to use dynamic programming.

Let $\Delta^\sharp_{q(v)} = \{\alpha_2, \ldots, \alpha_{\gamma-1}\}$. Let $A(i,j)$ be the approximation error corresponding to the optimal $q^*(v)$ with i points selected from the first j points in $\Delta^\sharp_{q(v)}$ for all $1 \leq i \leq j \leq \gamma - 2$ (since $\Delta^\sharp_{q(v)}$ has $\gamma - 2$ number of points), together with the two boundary Δ points (α_1 and α_γ). The optimal curve achieving $A(i,j)$ is denoted as $q^*(i,j)$. Next, let $\delta^{j+1}_{q^*(i,j)}$ be the area reduced by adding the $(j+1)$-th point from $\Delta^\sharp_{q(v)}$, i.e., α_{j+2}, to $q^*(i,j)$. Now, we have:

$$A(i,j) = \min \begin{cases} \min_{x \in [i-1, j-1]} \{A(i-1, x) - \delta^j_{q^*(i-1,x)}\}; \\ \min_{x \in [i, j-1]} \{A(i,x)\}. \end{cases} \tag{12}$$

Our goal is to find $A(\eta, \gamma - 2)$ and the corresponding $q^*(\eta, \gamma - 2)$ will be $q^*(v)$ with the minimum approximation error to $q(v)$ using only η right-upper corner points (plus α_1 and α_γ).

Given any $q^*(i,j)$ and α_{j+2}, $\delta^{j+1}_{q^*(i,j)}$ could be easily and efficiently computed using only subtraction and multiplication since $q^*(i,j)$ is a staircase curve. Suppose the last $\triangle$ point, except α_γ, in $q^*(i,j)$ is α_x, recall that α_{j+2} is $(v_{j+2}, q(v_{j+1}))$ and α_x is $(v_x, q(v_{x-1}))$, then:

$$\delta^{j+1}_{q^*(i,j)} = (v_{j+2} - v_x) \times (q(v_{j+1}) - q(v_{\gamma-1})). \tag{13}$$

The base case is when $i = j = 0$. This simply corresponds to having only the two boundary Δ points in $q^*(0,0)$ and

$$A(0,0) = \sum_{i \in [2, \gamma-1]} (v_i - v_{i-1})(q(v_{i-1}) - q(v_{\gamma-1})).$$

For example, in Figure 4(b), $q^*(0,0)$'s right-upper corner points are $(v_1, q(-\infty))$ and $(v_5, q(v_4))$, $A(0,0)$ corresponds to the the gray area in Figure 4(b), and $\delta^1_{q^*(0,0)}$ is the area reduced by adding α_2 to $q^*(0,0)$ (marked by the gray dotted lines in Figure 4(b)). This gives us a dynamic programming formulation for finding the right-upper corner points in optimal $q^*(v)$ for any η.

This dynamic programming algorithm requires only subtraction, addition and multiplication, and only needs to be carried out once per distributed site. Hence, even a site with limited computation power is able to carry out this procedure. Compared with the linear programming approach in Section 4.3, each site has shifted the expensive linear programming computation to the server.

It still remains to argue that such a $q^*(v)$ computed as above guarantees that $r^*(X, \mathcal{D}) \leq r^-(X, \mathcal{D})$, such that our lower bounding framework is still correct. This is formalized in the following theorem.

Theorem 1 *If $q^*(v)$ is constructed only using upper-right corners from the set of points $\Delta_{q(v)}$, then for any unknown tuple X with $E(X) \leq \tau$, $r^*(X, \mathcal{D}) \leq r^-(X, \mathcal{D})$.*

PROOF. The unknowns in the linear program of Section 4.3 are the p_ℓ's for $\ell = 1, \ldots, \gamma$, where $\gamma = |U|$. The v_ℓ's in the constraint from equation (10) only take values from U (or equivalently, the x-coordinates of the turning points of $q(v)$).

Suppose $q^*(v)$ has η points. Then the LP constructed from $q^*(v)$ has η unknowns. In the following, we will transform this LP into one also with γ unknowns, with the same constraints as the LP constructed from $q(v)$, while having a smaller objective function.

Denote the set of x-coordinates of the turning points in $q^*(v)$ as U^*. For any value $\bar{v} \in U - U^*$, we have $q^*(\bar{v}) \leq q(\bar{v})$. Now, we add the value $\bar{v}$ to U^* and a corresponding unknown to the LP. Note this transformation does not change the staircase curve defined by $q^*(v)$. We apply such transformations for all values from $U - U^*$. Now we obtain a LP that has the same set of unknowns and the same constraints (8)(9)(10) as the LP generated from $q(v)$. The objective function (7) of this transformed LP has smaller or equal coefficients. Thus the optimal solution to this transformed LP is no larger than that of the original LP constructed from $q(v)$.

Finally, we need to argue that this transformed LP is actually the same as the LP constructed from $q^*(v)$, namely, this transformation is merely conceptual and we do not need to actually do so. Indeed, since all the new unknowns that are added during the transformation are not at the turning points of $q^*(v)$, by the same reasoning of Section 4.3, we know that in the optimal solution of this transformed LP, these new unknowns will be zero anyway. Thus we do not need to actually include these unknowns in the LP, and it suffices to solve the simpler LP that is constructed just from $q^*(v)$. □

Theorem 1 indicates that by using $q_i^*(v)$'s, the server is able to find a lower bound for the local rank of any unseen tuple and check the terminating condition by solving the linear programming formulation *locally*. Note that the server still forwards every seen tuple to all sites to get its exact global rank based on the $q_i(v)$'s stored at individual sites. This, together with Theorem 1, guarantees that the final top-k are exact answers. There is the overhead of communicating $q_i^*(v)$'s to H at the beginning of the algorithm. However, it is a one-time cost. Furthermore, in each subsequent round the communication of passing τ from H to all sites and sending lower bound values from all sites back to the server in the *A-LP* algorithm is saved. This will compensate the cost of sending $q_i^*(v)$'s as evident from our experiments.

5.2 Updating the $q_i^*(v)$'s at the Server

Initially, each site computes the approximate $q_i^*(v)$ with a budget η for $q_i(v)$ and sends it to the server H. A nice thing about these approximate $q_i^*(v)$'s is that they can be incrementally updated locally by H after seeing tuples from the $\mathcal{D}_i$'s so that the approximation quality keeps improving after each update. In our framework, the budget η is only important for the initial transmission of $q_i^*(v)$. After that, the server has no constraint to keep only η number of points in $q_i^*(v)$. As the algorithm progresses and the sites send in their tuples, the server can also use these tuples to improve the quality of $q_i^*(v)$ "for free". Intuitively, when H receives a tuple from some site s_i, H's knowledge about $q_i(v)$ for the database $\mathcal{D}_i$ should be expanded, hence a better approximation for $q_i(v)$ should be possible.

The general problem is the following. Assume the server H has an initial $q^*(v)$ with a budget η for a database $\mathcal{D}$ that H does not possess. When a tuple $X \in \mathcal{D}$ is forwarded to H, we would like to update $q^*(v)$ with X such that the approximation error ε between $q^*(v)$ and $q(v)$ could be reduced.

Recall that $q^*(v)$ is represented by the set of right-upper corners obtained at the end of the dynamic programming. Suppose H gets a new tuple $X = \{(v_{x_1}, p_{x_1}), \ldots, (v_{x_z}, p_{x_z})\}$ for some z. Below we show how to update $q^*(v)$ for one pair $(v_\ell, p_\ell) \in X$; the same procedure is applied to all the pairs one by one.

We call the upper-right corner points in the initial $q^*(v)$ the *initial points*. Note that the two boundary points from $q(v)$ are always initial points. An obvious observation is that all the initial points should not be affected by any update since they are accurate points from the original $q(v)$. Consider an update (v_ℓ, p_ℓ), and the first initial point to its left, denoted Δ^L. Another observation is that this update will not affect the curve $q^*(v)$ outside the interval $[\Delta^L.v, v_\ell]$, where $\Delta^L.v$ is the value of Δ^L. This is because the update (v_ℓ, p_ℓ) will only raise the curve on the left side of v_ℓ, while the initial point Δ^L already incorporates all the information on the right side of Δ^L on the original curve $q(v)$.

We are now ready to describe how to update $q^*(v)$ with (v_ℓ, p_ℓ). By the definition of $q(v)$, the part of the curve of $q^*(v)$ to the left of v_ℓ should be raised by an amount of p_ℓ, if such a contribution has not been accounted for. As observed from above, this will raise $q^*(v)$ from v_ℓ all the way to the left until we hit Δ^L.

An example of this procedure is shown in Figure 4(c) with two updates. Suppose the initial points in $q^*(v)$ are α_1, α_4 and α_5. We first update with (v_2, p_2). This will raise the portion $(\Delta^L.v = v_1, v_2)$ by p_2. This corresponds to adding a new upper-right corner point α_2 to $q^*(v)$. Next, we update $q^*(v)$ with (v_ℓ, p_ℓ). As reasoned above, this will raise the portion $(\Delta^L.v = v_1, v_\ell)$ by p_ℓ. To record such a raise, we need to add a new upper-right corner α to $q^*(v)$, and then raise all the $\triangle$ points between $\Delta^L.v$ and v_ℓ by p_ℓ. In this example we will raise α_2 by p_ℓ.

6. REDUCING LATENCY AND OTHER ISSUES

Reducing latency. All of our algorithms presented so far process one tuple from some site s_j in a single round. The latency of obtaining the final result could be high if there are many rounds. However, there is an easy way to reduce latency. Instead of looking up one tuple at a time, our algorithms could process β tuples before running the lower bounding calculation, for some parameter β. Such a change could be easily adopted by all algorithms. The overall latency will be reduced by a factor of β. However, we may miss the optimal termination point, but by at most β tuples. In Section 7 we will further investigate the effects of β empirically.

Continuous distributions. When the input data in the uncertainty model is specified by a continuous distribution (e.g., Gaussian or Poisson), it is often hard to compute the probability that such a random variable exceeds another (e.g., there is no closed formula for Gaussian distributions). However, by discretizing the distributions to an appropriate level of granularity (i.e., represented by a histogram), we can reduce to an instance of the discrete pdf problem. The error in this approach is directly related to the granularity of the discretization.

Scoring function and other attributes. Our analysis has assumed that the score is an attribute. In general, the score can be specified at query time by a user defined function that could even involve multiple uncertain attributes. Our algorithms also work under this setting, as long as the scores can be computed, by treating the output of the scoring function as an uncertain attribute. Finally, users might be interested in retrieving attributes other than the ranking attribute(s) by the order of the scoring function. We could modify our algorithms to work with trimmed tuples that only contain the necessary attribute(s) for the ranking purpose. When the algorithm has terminated, we retrieve the top-k tuples from distributed sites with user-interested attributes based on the ids of the top-k truncated tuples at server H.

7. EXPERIMENTS

We implemented all the algorithms proposed in this paper: *A-LR*, *A-Markov*, *A-LP*, *A-BF* (the straightforward solution that sends all $\mathcal{D}_i$'s to H and process $\mathcal{D}$ locally using the *A-ERank* from [11]), *A-ALP* (the algorithm using approximate $q_i(v)$'s). For *A-LP* and *A-ALP*, we used the GNU linear programming kit library (GLPK) [15] to solve LPs.

Data sets. We used three real data sets and one synthetic data set. The first real data set is the *movie* data set from the MystiQ project [5], which contains probabilistic records as a result of information integration of the movie data from IMDB and Amazon. The *movie* data set contains roughly $56,000$ tuples. Each tuple is uniquely identified by the movie id. We rank tuples by the *ASIN* attribute, which varies significantly in different movie records, and may have up to 10 different choices, each associated with a probability.

The second real data set is the lab readings of 54 sensors from the Intel Research, Berkeley lab [13]. This data set contains four sets of sensor readings corresponding to the temperature, light intensity, humidity, and voltage of lab spaces over a period of eight days. These data sets exhibit similar results in all of our experiments, so we only report the results on the *temperature* data set. To reflect the fuzzy measurement in sensor readings, we put near-by g sensors (e.g., in the same room) into a group where g is a number between 1 and 10. We treat these g readings as a uniformly distributed discrete pdf of the temperature. The *temperature* data set has around $67,000$ such records. We rank tuples by their temperature attribute.

The third real data set is the *chlorine* data from the EPANET project that monitors and models the hydraulic and water quality behavior in water distribution piping systems [26]. This data set records the amounts of chlorine detected at different locations in the piping system collected over several days. The measurements were inherently fuzzy and usually several monitoring devices were installed at the same location. Hence, we process this data set in a similar way as the *temperature* data set. The *chlorine* data set has approximately $140,000$ records and each record has up to 10 choices on the value of the chlorine amount. We rank tuples by their chlorine amount attribute.

Finally, we also generated the synthetic *Gaussian* data set where each record's score attribute draws its values from a Gaussian distribution. For each record, the standard deviation σ is randomly selected from $[1, 1000]$ and the mean μ is randomly selected from $[5\sigma, 100000]$. Each record has g choices for its score values where g is randomly selected from 1 to 10. This data set can be also seen as a way to discretize continuous pdf's.

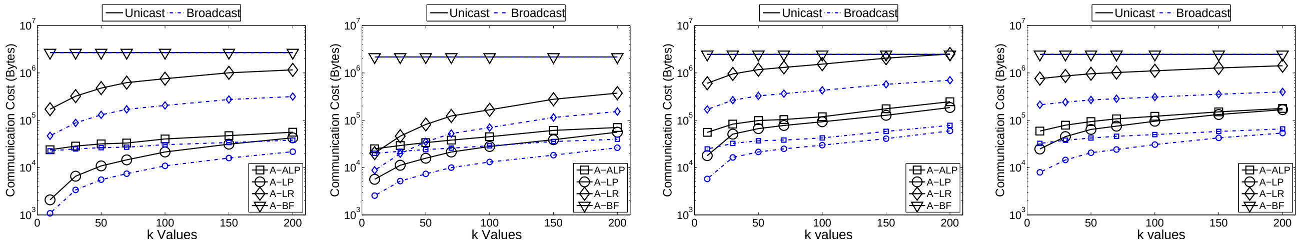
Setup. In each experiment, given the number of sites m, each record from the uncertain database $\mathcal{D}$ is assigned to a site s_i chosen uniformly at random. Once the $\mathcal{D}_i$'s are formed, we apply all algorithms on the same set of $\mathcal{D}_i$'s.

We measure the total communication cost in terms of bytes, as follows. For each choice of the score attribute, both the value and the probability are four bytes. The tuple id is also four bytes. We do not send attributes other than the score attribute and the tuple id. The expected score value is considered to be four bytes as well. We distinguish communication costs under either the broadcast or the unicast scenario. In the broadcast case, whenever the server sends a tuple or an expected score value to all sites, it is counted as one tuple or one value regardless of the number of sites. In the unicast case, such communication is counted as m tuples or m values being transmitted. In either case, all site-to-server communication is unicast.

We truncated all data sets to $N = 56,000$ tuples, the size of the *movie* data set. The default number of sites is $m = 10$ and the default budget η in the *A-ALP* algorithm is set to be 1% of $|q_i(v)|$. The default value of k is 100.

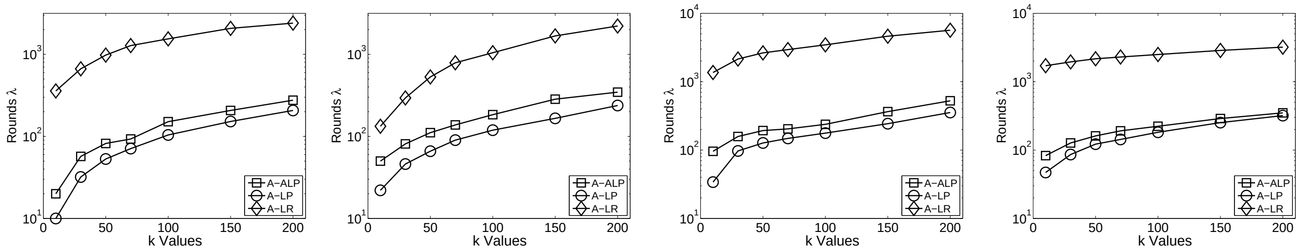
Results with different k. We first study the performance of all the algorithms for different k values from 10 to 200. Figure 5 shows the communication costs of the algorithms for the four data sets under both the broadcast and unicast settings. Clearly, *A-LP* and *A-ALP* save the communication cost by at least one to two orders of magnitude compared with *A-BF* in all cases. *A-LR* does provide communication savings over the brute-force approach, but as k increases, it quickly approaches *A-BF*. This indicates that the simple solution of using the local rank alone to characterize the global rank of a tuple is not good enough. *A-Markov* is consistently much worse than the *A-LP* and *A-ALP* algorithms in the access by expected score framework. In some cases, it actually retrieves almost all tuples from all sites. So we omit *A-Markov* from this and all remaining experiments. All algorithms have increasing communication costs as k gets larger (except *A-BF* of course). The costs of *A-LP* and *A-ALP* gradually increase as k gets larger. A useful consequence of this is that *A-LP* and *A-ALP* are able to return the top tuples very quickly to the user, and then return the remaining tuples progressively. We would like to emphasize that these results were from relatively small databases ($N = 56,000$). In practice, N could be much larger and the savings from *A-LP*, *A-ALP* comparing to *A-BF* could be from several to tens of orders of magnitude .

Another interesting observation is that *A-ALP* achieves similar communication cost as *A-LP*, while with very little computation overhead on the distributed sites. Recall that other than the initial computation of the $q_i^*(v)$'s, the distributed sites have little computation cost during the subsequent rounds in *A-ALP*. A reason is that *A-ALP* does not require the server to send the expected score value τ to all sites and collect the lower bounds on the local ranks based on τ. This results in some communication savings that compensate the needs of communicating $q_i^*(v)$'s at the beginning. We also show the number of rounds required in Figure 6. Note that for all values of k, *A-LP* and *A-ALP* need only slightly more than k rounds.



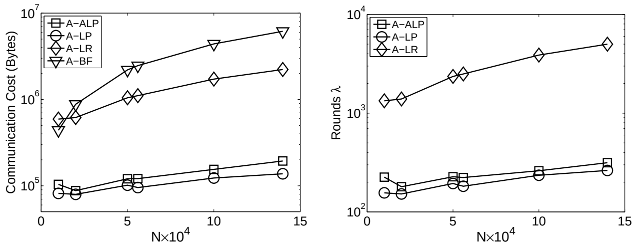
(a) Synthetic Gaussian. (b) Movie. (c) Temperature. (d) Chlorine.

Figure 5: Communication cost: $N = 56,000$, $m = 10$, $\eta = 1\% \times |q_i(v)|$, vary k.



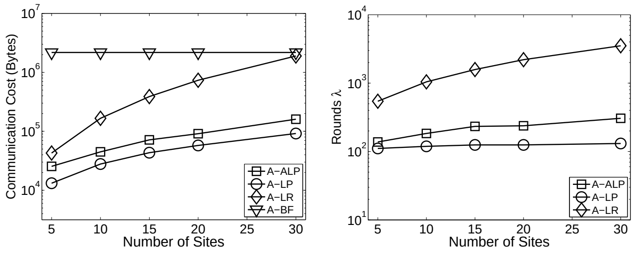
(a) Synthetic Gaussian. (b) Movie. (c) Temperature. (d) Chlorine.

Figure 6: Number of rounds: $N = 56,000$, $m = 10$, $\eta = 1\% \times |q_i(v)|$, vary k.



(a) Communication cost. (b) Number of rounds.

Figure 7: Effect of N, Chlorine data set: $m = 10$, $\eta = 1\%$, $k = 100$.

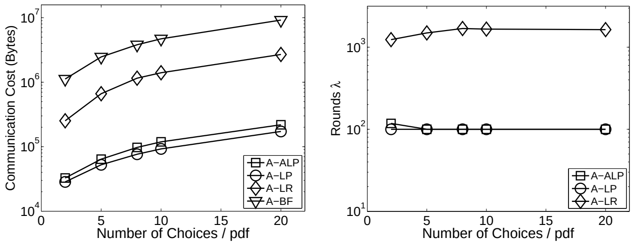


(a) Communication cost. (b) Number of rounds.

Figure 8: Effect of m, Movie data set: $N = 56,000$, $\eta = 1\%$, $k = 100$.

Finally, it is not surprising that our algorithms perform better in the broadcast case. In the rest, we only show the unicast scenario as the other case can only be better.

Results with different N. We next study the effects of N, the total number of records in the database $\mathcal{D}$, using the *chlorine* data set as it is the largest real data set. Not surprisingly, Figure 7 shows that the communication cost of the *A-BF* approach linearly increases with N (note that it is shown in log scale). On the other hand, both the communication cost and the number of rounds for *A-LP* and *A-ALP* increase at a much slower rate. For example, when $k = 100$, both algorithms only access less than 300 tuples (or rounds) even for the largest $N = 140,000$. This means that *A-LP* and *A-ALP* have excellent scalability w.r.t the size of the distributed database while others do not. The gap between *A-LP*, *A-ALP* comparing to *A-BF* quickly increases as N becomes larger.

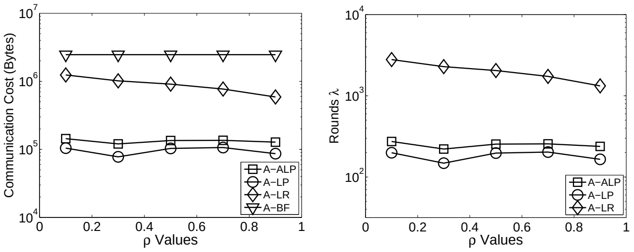


(a) Communication cost. (b) Number of rounds.

Figure 9: Effect of b, Synthetic Gaussian data set: $N = 56,000$, $m = 10$, $\eta = 1\%$, $k = 100$.

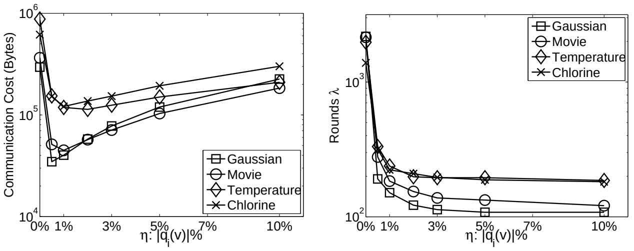
Results with different m. Our next goal is to investigate the effects of m, the number of sites. Figure 8 shows the experimental results on the *movie* data set where we varied m from 5 to 30 but kept N, the total database size (the union of all sites) fixed. Since we use unicast, as expected, the communication cost for our algorithms increase as m gets larger. Nevertheless, even with 30 sites, *A-LP* and *A-ALP* are still an order of magnitude better than the basic *A-BF* solution (Figure 8(a)). We would like to emphasize that this is the result from the *smallest database* with only 56,000 tuples and N is kept as a constant. In practice, when m increases, N will increase as well and *A-LP*, *A-ALP* will perform much better than *A-BF*. Finally, the number of sites does not have an obvious impact on the number of rounds required for *A-LP* and *A-ALP* (Figure 8(b)), which primarily depends on N and k.

Results with different b. We next study the effects of b, the upper bound on the size of each individual pdf. Recall



(a) Communication cost. (b) Number of rounds.

Figure 10: Effect of ρ (skewness of the pdf), Chlorine data set: $N = 56,000$, $m = 10$, $\eta = 1\%$, $k = 100$.



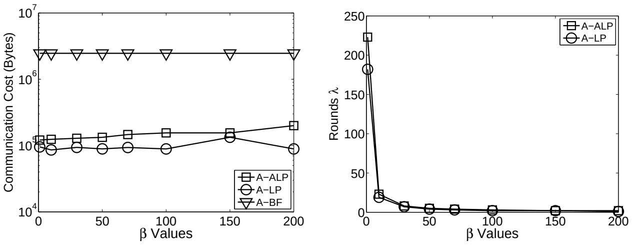
(a) Communication cost. (b) Number of rounds.

Figure 11: Effect of η on approximate $q_i(v)$'s: $N = 56,000$, $m = 10$, $k = 100$.

that for continuous pdf's, we discretize them into a discrete pdf's with up to b choices. The larger b is, the better we can approximate the original continuous pdf's. For this purpose, we use the synthetic *Gaussian* data set in which we can control b. The results are shown in Figure 9. As we can see from Figure 9(a), the communication costs of all algorithms increase roughly linearly with b. The relative gap among the algorithms basically stay the same. Figure 9(b) indicates that the number of rounds λ is almost not affected by b. This is because the dominant factor that determines λ is the expected score value. Changing the number of choices in a pdf does not shift its expected score value too much. Note that with $b = 20$, a continuous pdf and its discrete version is already very close in most cases.

Results with different skewness of the pdf's. We also study the effects of the skewness of the pdf's. Recall that by default for the *temperature* and *chlorine* data sets the probabilities for a pdf are set uniformly at random to reflect the scenario that the reading is randomly selected in a group of sensors. In practice, we may have a higher priority to collect the reading from one specified senor in a group, resulting in a skewed distribution. For this purpose, for a group of sensors, we always give a probability of ρ to one of them, while dividing the remaining probability equally among the rest of the sensors. Obviously, the larger ρ is, the higher the skewness. Figure 10 studies how this affects the algorithms using the *chlorine* data set. Obviously, this has no effect on the *A-BF* algorithm. Interestingly, Figure 10(a) and 10(b) indicate that both the communication cost and the number of rounds required for *A-LR*, *A-LP* and *A-ALP* algorithms actually reduce on more skewed distributions.

Results with different η. We then study the impact of the budget η of the approximate $q_i^*(v)$ for the *A-ALP* algorithm. Intuitively, smaller η's reduce the communication cost of transmitting these approximate $q_i^*(v)$'s, but also reduce their quality of approximation leading to larger number of rounds. So there is expected to be some sweet spot for the choice of η. It turns out that a fairly small η already reaches the sweet spot. Figure 11 shows the results on all four data sets. As seen in Figure 11(a), the communication cost drops sharply when η increases from 0% to 1% on all data sets. Note that $\eta = 0\%$ means that $q_i^*(v)$ contains only the two boundary points from $q_i(v)$. This indicates that by just adding a small number of points into $q_i^*(v)$, it does a very good job at representing $q_i(v)$ and hence gives a pretty tight lower bound $r^*(X, \mathcal{D}_i)$ on the estimated local rank of any unseen tuple X using the upper bound τ for the expected score of X. This is also evident from Figure 11(b) where the number of rounds drops significantly when η changes from 0% to 1%. As η gets larger, the overhead of communicating $q_i^*(v)$ starts to offset the communication savings. These results confirm that our algorithm does an excellent job in finding the optimal representation of $q_i(v)$'s given a limited budget. In practice, a tiny budget as small as 1%, seems already good enough. A small η also reduces the computation cost for both the server and the distributed sites. Thus the algorithm *A-ALP* is both computation- and communication-efficient. It requires minimal computational resources from the distributed sites, since only addition, subtraction and multiplication operations are needed to compute the $q_i^*(v)$'s. Subsequently, solving LPs is only done at the server.



(a) Communication cost. (b) Number of rounds.

Figure 12: Effect of β, Chlorine data set: $N = 56,000$, $m = 10$, $k = 100$, $\eta = 1\%|q_i(v)|$.

Results with different β. In all the experiments above, in each round our algorithms process only one tuple and then immediately check if it is safe to terminate. This causes a long latency if there are network delays. However, as argued in Section 6, we can easily reduce the latency by processing β tuples per round before checking the termination condition. This will reduce the number of rounds by a factor of β while only incurring an *additive* communication overhead of at most β tuples. In the last set of experiments, we empirically study the effects of β. The results are shown in Figure 12. First, as expected, the number of rounds is greatly reduced as β gets larger. For $\beta = 100$, both *A-LP* and *A-ALP* just need 2 rounds to complete, i.e., 2 round trips of communication. On the other hand, in this particular case, since the β value is still quite small when the number of rounds have been reduced to just 2, the increase in the total communication cost is almost negligible. This can be explained by the fact that although having a larger β makes the algorithms send $\leq \beta$ more tuples, we also save the communication cost of checking the termination condition repeatedly. This results in a net effect of quite flat curves that we see from Figure 12, meaning that the algorithms are both communication-efficient and fast.

In general, there is obviously a trade-off between the number of rounds and the total communication cost. In the extreme case, when β equals the total size of the distributed data sets (N) from all sites, this approach degrades to the *A-BF* approach. For any case with $\beta > 1$, the number of tuples retrieved by the server will be larger than the case with $\beta = 1$, resulting in a communication overhead. However, larger β values also imply that the server only has to communicate a single tuple to all sites to get the lower bounds back from every site after seeing β tuples, except in the case of *A-ALP* where the lower bounds are computed locally by the server. In the latter case larger β values reduce the computation overhead at the server, i.e., the server only needs to do the LPs once for every β tuples. For other algorithms, for small values of β, the savings of only retrieving the lower bounds once after seeing every β tuples cancels off the overhead of retrieving more tuples over the "optimal" terminating point with $\beta = 1$, as they cannot miss the "optimal" point by more than β tuples. We run this experiment with small β values (up to 200) as this already reduces the total rounds to just 1 or 2 for all data sets. If we keep increasing β, there will be a cut-off value where the delayed termination (have to look at many more tuples beyond the "optimal" point) will eventually result in more communication overhead than the savings.

Computation cost of solving the linear programs. Our main focus in this paper is to save the communication cost. However, in practice, the computation overhead should not be ignored. Our main algorithms, namely the *A-LP* and *A-ALP*, require solving the linear programs (LPs). It is interesting to examine the associated computation overhead. In our experiments, we found that such overheads are quite small. All of our experiments were executed on a linux machine with an Intel Xeon CPU 5130@2GHz and 4GB memory. On this machine, solving the LP in each round takes only a few seconds at most, and our best algorithm takes only two or tree rounds (the optimization with a β value that is larger than 1). The GLPK library is extremely efficient. Note that we cannot assume the distributed sites in real world applications are powerful, and this is precisely the reason why we wanted to migrate the computation cost to the server with the *A-ALP* algorithm.

8. RELATED WORK

There has been a large amount of efforts devoted to modeling and processing uncertain data, so we survey only the works most relevant to ours. TRIO [1, 28], MayBMS [2, 3] and MystiQ [12] are promising systems that are currently being developed. Many query processing and indexing techniques have been studied for uncertain databases and the most relevant works to this paper are top-k queries [11, 16, 27, 33, 37]—their definitions and semantics have been discussed in detail by the latest work [11] and the expected rank approach was shown to have important properties that others do not guarantee. Techniques used include the Monte Carlo approach of sampling possible worlds [27], AI-style branch-and-bound search of the probability state space [33], dynamic programming approaches [37], and applying tail (Chernoff) bounds to determine when to prune [16]. There is ongoing work to understand top-k queries in a variety of contexts. For example, the work of Lian and Chen [22] deals with ranking objects based on spatial uncertainty, and ranking based on linear functions. Recently, Soliman *et al.* [34] have extended their study on top-k queries [33] to Group-By aggregate queries. The Markov inequality was also applied in [11], however, in a different setting. They used the Markov inequality in a centralized approach and any unseen tuples cannot be accessed. In our case, the server does not access any unseen tuple. However, each distributed site could scan any tuple from that site and apply the Markov inequality over entire tuples in one site.

To the best of our knowledge, this is the first work on query processing on distributed probabilistic data. Distributed top-k queries have been extensively studied in certain data, including both the initial computation of the top-k [6, 14, 25, 32, 36] and the incremental monitoring/update version [4, 24]. Our work falls into the first category. Some works consider minimizing the scan depth at each site to be the top priority, i.e., the number of tuples a site has to access, such as the seminal work by Fagin *et al.* [14]. Arguably, the more important metric for distributed systems is to minimize the communication cost [4, 6, 25] which is our objective.

To capture more complex correlations among tuples, more advanced rules and processing techniques are needed in the uncertainty data model. Recent works based on graphical probabilistic models and Bayesian networks have shown promising results in both offline [29] and streaming data [20]. Converting prior probability into posterior probability also offers positive results [21]. In these situations, the general approaches are using Monte-Carlo simulations [18, 27] to obtain acceptable approximations or inference rules from graphical model and Bayesian networks, e.g., [21, 30].

9. CONCLUSION

This is the first work that studies ranking queries for distributed probabilistic data. We show that significant communication cost could be saved by exploring the interplay between the probabilities and the scores. We also demonstrate how to alleviate the computation burden at distributed sites so that communication and computation efficiency are achieved simultaneously. Many ranking semantics are still possible in the context of probabilistic data, extending our framework to those cases is an important future work. Finally, when updates are present at distributed sites, how to incrementally track (or monitor) the top-k results is also an intriguing open problem.

10. ACKNOWLEDGMENT

We thank the anonymous reviewers for the insightful comments. Feifei Li was supported by the Startup Grant from the Computer Science Department, Florida State University and NSF Grant CT-ISG-0831278. Ke Yi was supported in part by Hong Kong Direct Allocation Grant DAG07/08. Jeffrey Jestes was supported by the GAAN Fellowship from the US Department of Education.

11. REFERENCES

[1] P. Agrawal, O. Benjelloun, A. Das Sarma, C. Hayworth, S. Nabar, T. Sugihara, and J. Widom. Trio: A system for data, uncertainty, and lineage. In *VLDB*, 2006.

[2] L. Antova, T. Jansen, C. Koch, and D. Olteanu. Fast and simple relational processing of uncertain data. In

ICDE, 2008.

[3] L. Antova, C. Koch, and D. Olteanu. From complete to incomplete information and back. In *SIGMOD*, 2007.

[4] B. Babcock and C. Olston. Distributed top-k monitoring. In *SIGMOD*, 2003.

[5] J. Boulos, N. Dalvi, B. Mandhani, S. Mathur, C. Re, and D. Suciu. MYSTIQ: a system for finding more answers by using probabilities. In *SIGMOD*, 2005.

[6] P. Cao and Z. Wang. Efficient top-k query calculation in distributed networks. In *PODC*, 2004.

[7] S. Chaudhuri, K. Ganjam, V. Ganti, and R. Motwani. Robust and efficient fuzzy match for online data cleaning. In *SIGMOD*, 2003.

[8] S. Chaudhuri, L. Gravano, and A. Marian. Optimizing top-k selection queries over multimedia repositories. *IEEE TKDE*, 16(8):992–1009, 2004.

[9] R. Cheng, D. Kalashnikov, and S. Prabhakar. Evaluating probabilistic queries over imprecise data. In *SIGMOD*, 2003.

[10] G. Cormode, M. Garofalakis, S. Muthukrishnan, and R. Rastogi. Holistic aggregates in a networked world: distributed tracking of approximate quantiles. In *SIGMOD*, 2005.

[11] G. Cormode, F. Li, and K. Yi. Semantics of ranking queries for probabilistic data and expected ranks. In *ICDE*, 2009.

[12] N. Dalvi and D. Suciu. Efficient query evaluation on probabilistic databases. *VLDB Journal*, 16(4):523–544, 2007.

[13] A. Deshpande, C. Guestrin, S. Madden, J. Hellerstein, and W. Hong. Model-driven data acquisition in sensor networks. In *VLDB*, 2004.

[14] R. Fagin, A. Lotem, and M. Naor. Optimal aggregation algorithms for middleware. In *PODS*, 2001.

[15] GLPK. GNU Linear Programming Kit. http://www.gnu.org/software/glpk/.

[16] M. Hua, J. Pei, W. Zhang, and X. Lin. Ranking queries on uncertain data: A probabilistic threshold approach. In *SIGMOD*, 2008.

[17] R. Huebsch, M. Garofalakis, J. M. Hellerstein, and I. Stoica. Sharing aggregate computation for distributed queries. In *SIGMOD*, 2007.

[18] R. Jampani, F. Xu, M. Wu, L. L. Perez, C. M. Jermaine, and P. J. Haas. MCDB: a monte carlo approach to managing uncertain data. In *SIGMOD*, 2008.

[19] S. Jeyashanker, S. Kashyap, R. Rastogi, and P. Shukla. Efficient constraint monitoring using adaptive thresholds. In *ICDE*, 2008.

[20] B. Kanagal and A. Deshpande. Online filtering, smoothing and probabilistic modeling of streaming data. In *ICDE*, 2008.

[21] C. Koch and D. Olteanu. Conditioning probabilistic databases. In *VLDB*, 2008.

[22] X. Lian and L. Chen. Probabilistic ranked queries in uncertain databases. In *EDBT*, 2008.

[23] V. Ljosa and A. K. Singh. Top-k spatial joins of probabilistic objects. In *ICDE*, 2008.

[24] A. Manjhi, V. Shkapenyuk, K. Dhamdhere, and C. Olston. Finding (recently) frequent items in distributed data streams. In *ICDE*, 2005.

[25] S. Michel, P. Triantafillou, and G. Weikum. KLEE: a framework for distributed top-k query algorithms. In *VLDB*, 2005.

[26] S. Papadimitriou, J. Sun, and C. Faloutsos. Streaming pattern discovery in multiple time-series. In *VLDB*, 2005.

[27] C. Re, N. Dalvi, and D. Suciu. Efficient top-k query evaluation on probalistic databases. In *ICDE*, 2007.

[28] A. D. Sarma, O. Benjelloun, A. Halevy, and J. Widom. Working models for uncertain data. In *ICDE*, 2006.

[29] P. Sen and A. Deshpande. Representing and querying correlated tuples in probabilistic databases. In *ICDE*, 2007.

[30] P. Sen, A. Deshpande, and L. Getoor. Exploiting shared correlations in probabilistic databases. In *VLDB*, 2008.

[31] I. Sharfman, A. Schuster, and D. Keren. A geometric approach to monitoring threshold functions over distributed data streams. In *SIGMOD*, 2006.

[32] A. S. Silberstein, R. Braynard, C. Ellis, K. Munagala, and J. Yang. A sampling-based approach to optimizing top-k queries in sensor networks. In *ICDE*, 2006.

[33] M. A. Soliman, I. F. Ilyas, and K. C.-C. Chang. Top-k query processing in uncertain databases. In *ICDE*, 2007.

[34] M. A. Soliman, I. F. Ilyas, and K. C.-C. Chang. Probabilistic top-k and ranking-aggregate queries. *TODS*, To Appear, 2008.

[35] M. Wu, J. Xu, X. Tang, and W.-C. Lee. Top-k monitoring in wireless sensor networks. *IEEE TKDE*, 19(7):962–976, 2007.

[36] D. Zeinalipour-Yazti, Z. Vagena, D. Gunopulos, V. Kalogeraki, V. Tsotras, M. Vlachos, N. Koudas, and D. Srivastava. The threshold join algorithm for top-k queries in distributed sensor networks. In *DMSN*, 2005.

[37] X. Zhang and J. Chomicki. On the semantics and evaluation of top-k queries in probabilistic databases. In *DBRank*, 2008.