Finding Frequent Items in Probabilistic Data

Qin Zhang[1∗] Feifei Li[2] Ke Yi[1†]

[1]Dept Computer Science and Engineering
Hong Kong University of Science & Technology
Clear Water Bay, Hong Kong
{qinzhang, yike}@cse.ust.hk

[2]Dept Computer Science
Florida State University
Tallahassee, FL, USA
lifeifei@cs.fsu.edu

ABSTRACT

Computing statistical information on probabilistic data has attracted a lot of attention recently, as the data generated from a wide range of data sources are inherently fuzzy or uncertain. In this paper, we study an important statistical query on probabilistic data: finding the frequent items. One straightforward approach to identify the frequent items in a probabilistic data set is to simply compute the expected frequency of an item and decide if it exceeds a certain fraction of the expected size of the whole data set. However, this simple definition misses important information about the internal structure of the probabilistic data and the interplay among all the uncertain entities. Thus, we propose a new definition based on the *possible world semantics* that has been widely adopted for many query types in uncertain data management, trying to find all the items that are likely to be frequent in a randomly generated possible world. Our approach naturally leads to the study of ranking frequent items based on confidence as well.

Finding likely frequent items in probabilistic data turns out to be much more difficult. We first propose exact algorithms for offline data that run in either quadratic or cubic time. Next, we design novel sampling-based algorithms for streaming data to find all approximately likely frequent items with theoretically guaranteed high probability and accuracy. Our sampling schemes consume sublinear memory and exhibit excellent scalability. Finally, we verify the effectiveness and efficiency of the developed algorithms using both real and synthetic data sets with extensive experimental evaluations.

Categories and Subject Descriptors

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

[∗]Supported by Hong Kong CERG Grant 613507.

[†]Supported in part by Hong Kong Direct Allocation Grant (DAG07/08).

General Terms

Algorithms

Keywords

Heavy Hitters, Probabilistic Data, Uncertain Databases, X-relation Model.

1. INTRODUCTION

An important problem in many data management systems is to identify frequent items. People are interested in tracking these items for a number of reasons. For example, in network traffic monitoring, frequent packets consuming most of the bandwidth should be efficiently reported for accounting purposes [17]; in other applications, frequent items are essential for answering the popular iceberg queries [18, 22] and association rule mining [3]. Frequent items in a large data set are commonly referred to as *heavy hitters*. More precisely, the heavy hitters in a data set are those items whose relative frequency exceeds a specified threshold, i.e., for a parameter ϕ, an item t is the *ϕ-heavy hitter* of a multiset W if

$$m_t^W > \phi \cdot |W|, \tag{1}$$

where m_t^W is the multiplicity of t in W.

Due to its importance in a wide range of applications, the topic of finding heavy hitters has been extensively explored [11, 12, 15, 25, 27, 28, 30, 31]. If we have $\Omega(n)$ memory, where n is the number of distinct items in the data set, the problem of finding ϕ-heavy hitters is trivially solved by keeping a counter for each distinct item that ever appears in the data set. So most of the existing works focus on algorithms using sublinear memory to find ϵ-approximate ϕ-heavy hitters in one pass. More precisely, if an item is a ϕ-heavy hitter, then the algorithm will return it; if an item is not a $(\phi - \epsilon)$-heavy hitter, then the algorithm must not return it; for the rest of the items, the algorithm may or may not return them. Among all the techniques proposed, the *Space-Saving* algorithm [32] delivers the best performance. It uses $O(1/\epsilon)$ space and spends $O(1)$ time to process each item.

However, little is known on finding frequent items in probabilistic data, which is an emerging area that has attracted a lot of attention recently, due to the observation that the data generated from many applications is inherently fuzzy or uncertain. For example, in data integration [8, 21, 23], data items in the output could have varied degrees of confidence, depending on how well the underlying tuples from different sources match with each other. In applications that handle measurement data, e.g., sensor readings, the data is inherently noisy, and ought to be represented by a probability distribution rather than a single deterministic value [9, 16]. Statistical information, such as the heavy hitters, plays a crucial role in the context of uncertain data management, as probabilistic data essentially represents a distribution of an exponential number of data set instances, and it is very important to be able to extract meaningful statistical information from all these instances. For example, it would be nice if we can find frequent items efficiently from sensor readings. Not surprisingly, people have already started to focus attention on tackling these problems [10, 24].

Probabilistic data models. A number of probabilistic data models have been proposed in the literature [2, 7, 13, 26, 36, 37], ranging from the *basic model* where each tuple appears with a certain probability independently, to powerful models that are *complete*, i.e., models that can represent any probability distribution of the data instances. However, complete models have exponential complexity and are hence infeasible to handle efficiently. Alternatively, extensions to the basic model are introduced to expand its expressiveness while keeping the computation tractable. The *x-relation* model has been proposed in the TRIO [2] system, in which an uncertain database $\mathcal{D}$ consists of a number of *x-tuples*. Each x-tuple T includes a number of *items* as its alternatives, associated with probabilities, representing a discrete probability distribution of these alternatives being selected. Independence is still assumed among the x-tuples. The probability of the an item t is denoted as $p(t)$. Thus, an x-tuple T is a set of a bounded number of items, subject to the constraint that $\sum_{t\in T} p(t) \le 1$. Without loss of generality, we assume that all items take values from the integer domain $[n] = \{1, \dots, n\}$. Following the popular *possible worlds semantics*, $\mathcal{D}$ is instantiated into a *possible world* with mutual independence of the x-tuples. More precisely, let $T_1, \dots, T_m$ be the x-tuples of $\mathcal{D}$, and let W be any subset of the tuples appearing in $\mathcal{D}$, the probability of W occurring is $\Pr[W] = \prod_{j=1}^{m} p_W(T_j)$ with:

$$p_W(T) = \begin{cases} p(t), & \text{if } T \cap W = \{t\}; \\ 1 - \sum_{t\in T} p(t), & \text{if } T \cap W = \emptyset; \\ 0, & \text{otherwise.} \end{cases}$$

If $\Pr[W] > 0$, we say W is a *possible world*, and we denote by $\mathcal{W}$ the set of all possible worlds. Please refer to Figure 1 for an example. In summary, items from the same x-tuple are mutually excluded and items from different x-tuples are independently selected. This model has been frequently used in the literature [2, 7, 24, 34] as it is a reasonable approximation of the uncertain nature of the data.

In addition to the offline data, high-speed, massive data streams are now gathered in a variety of applications such as network traffic, sensor readings, etc. These data streams often have an uncertain nature, so it is worth studying probabilistic data under the streaming setting as well. It is fairly easy to extend the x-relation model to data streams. We generally assume that each element in the data stream is an x-tuple, representing a discrete probability distribution of possible items this element might be. The basic probabilistic data model, being a special case of the x-relation model, can be also extended in the same way. In fact, these *probabilistic data stream* models have been adopted in several

	x-tuples
T_1	$\{(t_1, p(t_1)), (t_2, p(t_2)), (t_3, p(t_3))\}$
T_2	$\{(t_4, p(t_4)), (t_5, p(t_5))\}$

i	W	$\Pr[W]$
1	$\emptyset$	$(1 - p(t_1) - p(t_2) - p(t_3))(1 - p(t_4) - p(t_5))$
2	$\{t_1\}$	$p(t_1)(1 - p(t_4) - p(t_5))$
3	$\{t_2\}$	$p(t_2)(1 - p(t_4) - p(t_5))$
4	$\{t_3\}$	$p(t_3)(1 - p(t_4) - p(t_5))$
5	$\{t_4\}$	$(1 - p(t_1) - p(t_2) - p(t_3))p(t_4)$
6	$\{t_5\}$	$(1 - p(t_1) - p(t_2) - p(t_3))p(t_5)$
7	$\{t_1 t_4\}$	$p(t_1)p(t_4)$
8	$\{t_2 t_4\}$	$p(t_2)p(t_4)$
9	$\{t_3 t_4\}$	$p(t_3)p(t_4)$
10	$\{t_1 t_5\}$	$p(t_1)p(t_5)$
11	$\{t_2 t_5\}$	$p(t_2)p(t_5)$
12	$\{t_3 t_5\}$	$p(t_3)p(t_5)$

Figure 1: Uncertain database and possible worlds. We use a pair $(t, p(t))$ to represent an item t with probability $p(t)$.

recent works in this direction [10, 24].

Novel query processing techniques for various query types have flourished in uncertain databases. Among them, ranking queries [34, 39] and queries gathering statistical information [10, 24] are areas of particular interests since they help discover knowledge to reflect what is more likely to happen in real world and it is possible to concisely represent results of such queries. In this paper, we study one statistical query of particular importance: finding frequent items on uncertain data. We also study the ranking of frequent items on uncertain data based on confidence.

Heavy hitters in probabilistic data. When extending the interpretation of heavy hitters to probabilistic data, a natural way is to replace m_t^W and $|W|$ in (1) with their expected values, leading to the following definition.

Definition 1 (Expected heavy hitters) Given an uncertain database $\mathcal{D}$ with $\mathcal{W}$ as its space of all possible worlds, an item t is a *ϕ-expected heavy hitter* (EHH) of $\mathcal{D}$ if its expected multiplicity exceeds ϕ times the expected size of $\mathcal{D}$, that is,

$$\sum_{W\in\mathcal{W}} m_t^W \Pr[W] > \phi \cdot \sum_{W\in\mathcal{W}} |W| \Pr[W]. \qquad (2)$$

In fact, this definition has been adopted in the pioneer work of [10]. By the linearity of expectation, it does not take long to realize that the EHH problem is equivalent to the weighted (deterministic) heavy hitter problem, by treating $p(t)$ as the weight of t. As a result, the problem is again trivial if we have $\Omega(n)$ memory. If not, as noted in [10], many previous approximate heavy hitter algorithms can be modified to support the weighted version of the problem.

Indeed, if the multiplicity of an individual item is considered as a statistical aggregate, then the definition of EHH makes perfect sense. However, if we consider finding the heavy hitters as a *query* evaluated on the data set, then this definition becomes problematic. First, relying on the expectations over all possible worlds usually does not reflect the intricate characteristics in an uncertain database. For example, consider the following two uncertain databases $\mathcal{D}_1 = \{\{(1, 0.4), (2, 0.6)\}, \{(3, 0.5)\}\}$ and $\mathcal{D}_2 = \{\{(1, 0.4)\}, \{(2, 0.6)\}, \{(3, 0.5)\}\}$. In $\mathcal{D}_1$, item 1 and item 2 are mutually exclusive, they cannot both appear in a possible world. While in $\mathcal{D}_2$, they are independent and it is possible for them to appear together. However, in the eyes of EHH, $\mathcal{D}_1$ and $\mathcal{D}_2$ are identical. Second, since EHH is only concerned with the expected frequency of an item and ignores all other characteristics of the distribution of the possible worlds, in some situations it may miss important features in the underlying probability space the uncertain data model is representing, as illustrated by the following two simple examples.

Example 1 Consider the following uncertain database $\mathcal{D}_1 = \{\{(1, 0.9), (2, 0.1)\}, \{(3, 1)\}\}$ consisting of 2 x-tuples. Suppose we are interested in 0.5-heavy hitters. According to the definition, 1 is not a 0.5-expected heavy hitter. However, 1 in fact has a 90% chance of being a 0.5-heavy hitter; such a feature is not captured by the definition of expected heavy hitters.

Example 2 On the other hand, consider the following uncertain database $\mathcal{D}_2 = \{\{(1, 0.5)\}, \{(2, 0.5)\}\}$. Suppose we are still interested in 0.5-heavy hitters. We notice that 1 is a 0.5-expected heavy hitter, but only has 50% chance of being a 0.5-heavy hitter.

Thus a more rigorous approach of extending the heavy hitter definition to probabilistic data is to follow the *possible world semantics* [14]. Under this framework, the problem of finding all heavy hitters is treated as a query, and we are interested in evaluating this query on all possible worlds and computing for each possible set of items, its probability of being the query result. As there could be exponentially many possible results, a compact representation, called the *probabilistic ranking* [14], is typically returned to the user. Here, instead of computing the probability for each possible *set* of heavy hitters, we for each individual item compute its probability of belonging to the answer to the heavy hitter query evaluated on any possible world. Still, this probabilistic ranking contains one entry for each item in $\mathcal{D}$, so usually we only return all items with probability above a certain threshold τ. Intuitively speaking, we would like to identify all items that are *likely* to be frequent items in a possible world randomly instantiated from $\mathcal{D}$.

Definition 2 (Probabilistic heavy hitters) Given an uncertain database $\mathcal{D}$ with $\mathcal{W}$ as its space of all possible worlds, an item t is a (ϕ, τ)*-probabilistic heavy hitter* (PHH) of $\mathcal{D}$ if

$$\sum_{W \in \mathcal{W}, m_t^W > \phi|W|} \Pr[W] > \tau. \tag{3}$$

The definition of PHH treats the set of heavy hitters holistically. Whether an item is a heavy hitter in a particular possible world W does not only depend on its own frequency, but also other items' combined frequency. Unlike the EHH definition which simply relies on individual items' expected frequencies, the PHH definition captures the intricate interplay between the uncertain items in a randomly generated possible world represented by the probabilistic data model.

Henceforth we use R to denote a random possible world instantiated from $\mathcal{D}$. Now both m_t^R and $|R|$ are random variables, and we can conveniently rewrite (2) and (3) as

$$\mathbf{E}[m_t^R] > \phi \cdot \mathbf{E}[|R|] \text{ and,} \tag{4}$$

$$\Pr[m_t^R > \phi|R|] > \tau, \tag{5}$$

respectively. Furthermore, in addition to returning a collection of probabilistic heavy hitters, we would also like to return the value of $\Pr[m_t^R > \phi|R|]$ for each returned item, which represents the confidence of the item being a true result. By sorting these confidences, we can also consider top-k queries, that is, to find the k items with the largest confidence of being a heavy hitter.

Our contributions. The problem of finding the PHH's is much more difficult than finding the EHH's. It is not even clear how to do this efficiently even when there is sufficient memory. The problem is even more challenging under the streaming setting, when there is limited memory and the data items arrive at a fast rate. In this paper, we tackle these challenges by 1) formalizing the PHH model and pointing out its superiority against the EHH model; 2) giving low degree polynomial-time algorithms for computing the exact PHH's for offline data; 3) designing both space and time-efficient algorithms to compute the approximate PHH's for streaming data, with theoretically guaranteed accuracy and space/time bounds; 4) establishing a tradeoff between the accuracy and the per-tuple processing time of the proposed approximation algorithms for PHH; 5) formalizing the top-k query for ranking frequent items in uncertain data and giving a method to answer such queries efficiently with provable guarantees; and 6) showing that these algorithms are simple to implement and perform well in practice through extensive experimental evaluations on both real and synthetic data.

2. EXACT ALGORITHMS: OFFLINE DATA

In this section, we propose algorithms to find all probabilistic heavy hitters exactly according to (5) for offline data assuming there is sufficient memory.

2.1 Algorithms for a Single Item

We first give algorithms for computing the confidence of a given single item t being a heavy hitter, i.e., $\Pr[m_t^R > \phi|R|]$. The algorithms are based on dynamic programming and run in polynomial time.

The basic model. We start by considering the basic probabilistic model, in which each tuple appears independently with a certain probability. This corresponds to a degenerated case of the x-relation model, where every x-tuple contains only one alternative (thus one item). In this scenario, we can compute $\Pr[m_t^R > \phi|R|]$ in $O(m^2)$ time, where m is the number of x-tuples in the uncertain database $\mathcal{D}$.

To perform the dynamic programming, we create a two dimensional table $B^t[i, j]$ for item t, where the value of the cell indexed by $[i, j]$ denotes the probability that "item t appears i times in the first j x-tuples of $\mathcal{D}$". Let (w_j, p_j) be the (only) alternative of the j-th x-tuple, where w_j is the item and p_j is the probability that w_j appears. We have the following induction step, for $i \geq 1, j \geq 1$.

$$B^t[i,j] = \begin{cases} B^t[i, j-1], & \text{if } w_j \neq t; \\ B^t[i, j-1](1 - p_j) + B^t[i-1, j-1]p_j, & \text{if } w_j = t. \end{cases}$$

The base cases are: $B^t[0, 0] = 1$, $B^t[i, 0] = 0$ $(i \geq 1)$, and

$$B^t[0,j] = \begin{cases} B^t[0, j-1], & \text{if } w_j \neq t, j \geq 1; \\ B^t[0, j-1](1 - p_j), & \text{if } w_j = t, j \geq 1. \end{cases}$$

Next, we create a similar table $B^{\bar{t}}[i,j]$, where $\bar{t}$ stands for the set of all items other than t, and the cell indexed by $[i,j]$ denotes the probability of "items other than t appear i times in the first j x-tuples of $\mathcal{D}$". Filling up all the cells in B^t and $B^{\bar{t}}$ takes $O(m^2)$ time, as there are $O(m^2)$ cells in total, and computing each $B^t[i,j]$ or $B^{\bar{t}}[i,j]$ takes constant time.

After populating these two tables, $\Pr[m_t^R > \phi|R|]$ can be calculated as

$$\Pr[m_t^R > \phi|R|] = \sum_{i=1}^{m} B^t[i,m] \left(\sum_{j=1}^{\lfloor \frac{1-\phi}{\phi} \rfloor i} B^{\bar{t}}[j,m] \right). \quad (6)$$

Note that we do not need to compute the inner sum $\sum_{j=1}^{\lfloor \frac{1-\phi}{\phi} \rfloor i} B^{\bar{t}}[j,m]$ of (6) from scratch for each i. As i increases, this sum can be computed incrementally, taking $O(m)$ time in total. However, constructing the two tables B^t and $B^{\bar{t}}$ is the dominating term in the running time of the algorithm.

Lemma 1 *Under the basic probabilistic data model, our algorithm spends $O(m^2)$ time to compute $\Pr[m_t^R > \phi|R|]$ for a single item t.*

The general x-relation model. For the case where each x-tuple may contain multiple alternatives, our approach is an extension of the algorithm for the basic case. For a particular item t, we start by rewriting each x-tuple in the offline data in the form of $\{(t, p_k(t)), (\bar{t}, p_k(\bar{t})\}$, where $p_k(t)$ denotes the probability to choose t in the k-th x-tuple, $\bar{t}$ stands the set of all items other than t in that x-tuple and $p_k(\bar{t})$ is the sum of the probability of items in $\bar{t}$, i.e., $p_k(\bar{t})$ is the probability that t is not chosen in the k-th x-tuple. For instance, for an x-tuple $T = \{(1, 0.2), (2, 0.3), (3, 0.2)\}$, if $t = 1$, then we rewrite it as $T' = \{(1, 0.2), (\bar{1}, 0.5)\}$. It is possible that item t does not appear in some x-tuples, or t is the only item in an x-tuple. We set $p_k(t) = 0$ and $p_k(\bar{t}) = 0$ respectively in these two cases.

Next, we create a three dimensional table $A^t[i,j,k]$ ($0 \le i \le m, 0 \le j \le m, 0 \le k \le m$) for the item t to run the dynamic program. The cell $A^t[i,j,k]$ denotes the probability that "item t appears i times and items other than t appear j times in the first k x-tuples of $\mathcal{D}$". The induction steps as well as the base cases for DP are defined as follows.

$$A^t[i,j,k] = \begin{cases} A^t[i-1,j,k-1]p_k(t) + A^t[i,j-1,k-1]p_k(\bar{t}) \\ \quad + A^t[i,j,k-1](1-p_k(t)-p_k(\bar{t})), \\ \quad \text{if } i \ge 1, j \ge 1, k \ge 1; \\ A^t[0,j,k-1](1-p_k(t)-p_k(\bar{t})) \\ +A^t[0,j-1,k-1]p_k(\bar{t}), \quad \text{if } i=0, j \ge 1, k \ge 1; \\ A^t[i,0,k-1](1-p_k(t)-p_k(\bar{t})) \\ +A^t[i-1,0,k-1]p_k(t), \quad \text{if } i \ge 1, j=0, k \ge 1; \\ \Pi_{l=1}^{k}(1-p_l(t)-p_l(\bar{t})), \quad \text{if } i=0, j=0, k \ge 1; \\ 1, \quad \text{if } i=0, j=0, k=0; \\ 0, \quad \text{if } i \ge 0, j \ge 1, k=0 \text{ or } i \ge 1, j \ge 0, k=0. \end{cases}$$

After the whole table of A^t is calculated, we can obtain the probability $\Pr[m_t^R > \phi|R|]$ by simply summing up the values of all entries $A^t[i,j,m]$ with $i > \lfloor \frac{\phi}{1-\phi} \rfloor j$. Therefore we can determine whether item t is a probabilistic heavy hitter in time $O(m^3)$.

Lemma 2 *Under the general x-relation probabilistic data model, our algorithm spends $O(m^3)$ time to compute $\Pr[m_t^R > \phi|R|]$ for a single item t.*

A naive approach to find all the (ϕ,τ)-probabilistic heavy hitters is to repeat using Lemma 1 or Lemma 2 for each individual item, and then checking if $\Pr[m_t^R > \phi|R|] > \tau$, resulting in prohibitively high running times of $O(nm^2)$ or $O(nm^3)$ respectively (recall that n is the number of unique items in the data set). In the next subsection, we prove a pruning lemma that dramatically reduces the number of items for which we need to run the dynamic program.

2.2 The Pruning Lemma

The following lemma gives an upper bound on $\Pr[m_t^R > \phi|R|]$ depending on $\mathbf{E}[m_t^R]/\mathbf{E}[|R|]$, which effectively prunes many items that cannot be (ϕ,τ)-probabilistic heavy hitters, thereby dramatically speeding up the algorithm.

Lemma 3 *For any item t,*

$$\Pr[m_t^R > \phi|R|] < \frac{\mathbf{E}[m_t^R]}{\phi(1-c)\mathbf{E}[|R|]} + e^{-\frac{c^2}{2}\mathbf{E}[|R|]},$$

where $0 < c < 1$ is an arbitrary constant.

Proof. We decompose the event $(m_t^R > \phi|R|)$ into two disjoint events $(m_t^R > \phi|R|) \cap (|R| \ge (1-c)\mathbf{E}[|R|])$ and $(m_t^R > \phi|R|) \cap (|R| < (1-c)\mathbf{E}[|R|])$, and bound their probabilities respectively.

First,

$$\begin{aligned} & \Pr\left[(m_t^R > \phi|R|) \cap (|R| \ge (1-c)\mathbf{E}[|R|])\right] \\ \le\ & \Pr[m_t^R \ge \phi(1-c)\mathbf{E}[|R|]] \\ \le\ & \frac{\mathbf{E}[m_t^R]}{\phi(1-c)\mathbf{E}[|R|]}. \qquad \text{(Markov inequality)} \end{aligned}$$

Next,

$$\begin{aligned} & \Pr\left[(m_t^R > \phi|R|) \cap (|R| < (1-c)\mathbf{E}[|R|])\right] \\ <\ & \Pr[|R| < (1-c)\mathbf{E}[|R|]] \\ \le\ & e^{-\frac{c^2}{2}\mathbf{E}[|R|]}. \qquad \text{(Chernoff inequality)} \end{aligned}$$

Summing up the two terms above yields the lemma. □

To simplify the expression, we choose $c = 1/2$ in Lemma 3. For $E[|R|]$ large enough, we have $e^{-\mathbf{E}[|R|]/8} \le 1/\mathbf{E}[|R|]$. Therefore, Lemma 3 becomes

$$\Pr[m_t^R > \phi|R|] < \frac{3}{\phi} \cdot \frac{\mathbf{E}[m_t^R]}{\mathbf{E}[|R|]},$$

which implies that

$$\frac{\mathbf{E}[m_t^R]}{\mathbf{E}[|R|]} < \frac{\phi\tau}{3} \Rightarrow \Pr[m_t^R > \phi|R|] < \tau,$$

that is, a (ϕ,τ)-probabilistic heavy hitter is necessarily a $\frac{\phi\tau}{3}$-expected heavy hitter, while there are at most $\frac{3}{\phi\tau}$ such items. Thus, we can first compute $\mathbf{E}[m_t^R]$ for all items t (summing up items' probability for those equal to t) as well as $\mathbf{E}[|R|]$ (summing up all items' probability) by a single scan and locate all the $\frac{\phi\tau}{3}$-expected heavy hitters (we call them candidates). Next, we run the dynamic program to compute the exact $\Pr[m_t^R > \phi|R|]$ for each of these candidates t.

Finally, the straightforward implementation requires storing either a two or three dimensional table for the dynamic programming, leading to an $O(m^2)$ or $O(m^3)$ memory consumption. However, note that we only care about obtaining $B^t[i, m]$ for $i \in [0, m]$ or $A^t[i, j, m]$ for $i, j \in [0, m]$, i.e., the intermediate entries $B^t[i, j]$ for $j < m$ or $A^t[i, j, k]$ for $k < m$ need not be stored. Furthermore, at each step j or k in the dynamic programming, only $B^t[i, j-1]$ and $B^t[i, j]$ for $i \in [0, m]$ or $A^t[i, j, k-1]$ and $A^t[i, j, k]$ for $i, j \in [0, m]$ are required. Hence, we could effectively reduce the memory usage to $O(m)$ and $O(m^2)$ respectively.

Theorem 1 *For an offline uncertain data set $\mathcal{D}$, all the ϕ-probabilistic heavy hitters can be found by our algorithms in $O(\frac{1}{\phi\tau}m^2)$ time and $O(m)$ space for the basic probabilistic data model, or $O(\frac{1}{\phi\tau}m^3)$ time and $O(m^2)$ space for the x-relation model.*

3. APPROXIMATE ALGORITHMS FOR STREAMING DATA

As we have seen in the previous section, computing the PHH's is much more difficult than computing the EHH's for offline data. Under the streaming setting with limited memory, the problem becomes even more challenging. We know that even for EHH, if we do not have $\Omega(n)$ memory, then we can only settle for approximate heavy hitters, that is, for an error term ϵ, an item is an EHH if $\mathbf{E}[m_t^R] > \phi \cdot \mathbf{E}[|R|]$, not an EHH if $\mathbf{E}[m_t^R] < (\phi - \epsilon) \cdot \mathbf{E}[|R|]$, while the decisions for the other items are arbitrary. As observed in [10], the Space-Saving algorithm [32] can be modified to find all these approximate EHH's with $O(1/\epsilon)$ memory and $O(\log(1/\epsilon))$ processing time per element in the stream.

Lemma 4 ([10, 32]) *The modified space-saving algorithm finds all the approximate* EHH*'s in a data stream using $O(1/\epsilon)$ memory and $O(\log(1/\epsilon))$ processing time per element.*

In order to distinguish the two versions of Space-Saving algorithm in the rest of the paper, we call the modified one the weighted version and the original one the unweighted version. Recall that the unweighted space-saving algorithm finds $(\phi - \epsilon)$-heavy hitters for deterministic stream with $O(1/\epsilon)$ memory and $O(1)$ precessing time per tuple [32].

To find all the PHH's, we need to compute the probability that an item is a heavy hitter. However, since it is difficult to calculate $\Pr[m_t^R > \phi|R|]$ or $\Pr[m_t^R > (\phi - \epsilon)|R|]$ exactly, we have to introduce another level of approximation on τ. More precisely, we say that an item t is an approximate PHH if $\Pr[m_t^R > \phi|R|] > \tau$, and not an approximate PHH if

$$\Pr[m_t^R > (\phi - \epsilon)|R|] < (1 - \theta)\tau,$$

for some small θ. For the other items, the decisions are arbitrary. In other words, if we relax the requirement on ϕ by ϵ (which increases the probability of an item being a heavy hitter), the item still has some small gap $\theta\tau$ to the required probability τ, then it should not be treated as an approximate PHH. In the sequel we will drop the word "approximate" when there is no confusion.

Below we first give a basic sampling algorithm to compute the PHH's. It has the desired sublinear memory consumption, but suffers from a large processing time. Then we further improve its running time so that high-speed data streams can be processed in real time.

3.1 Basic Sampling Algorithm

We will use the unweighted space-saving algorithm [32] as a subroutine in our basic sampling algorithm to find heavy hitters in each possible world. An important property of the Space-Saving algorithm is that it will only overestimate the frequency of an item by at most ϵm, but never underestimate it. This property is important for our analysis.

The algorithm. Our general framework follows from the seminar work of Alon et. al. [4]. We keep $l = 2\ln(1/\delta')$ groups of possible worlds, say, $G_1, G_2, \ldots, G_l$, where δ' is a parameter concerning the possible error of the algorithm. For each group G_i, we create $k = \frac{8}{\theta^2\tau}$ possible worlds, say $W_{i1}, W_{i2}, \ldots, W_{ik}$. Each possible world will be a sample randomly generated from the probabilistic data stream. We will maintain all these possible worlds in parallel, that is, when an x-tuple T arrives, for each possible world $W_{ij}, 1 \leq i \leq l, 1 \leq j \leq k$, we sample one item from T and include it to W_{ij} independently. More precisely, if T contains only one alternative t, then we add it to W_{ij} with probability $p(t)$; if T contains multiple alternatives, then we randomly choose one item from these alternatives according to their probabilities and add it to W_{ij}.

For each possible world sample W_{ij}, we run the Space-Saving algorithm to find all of its approximate heavy hitters. For each item t, let $X_{ij}^t = 1$ if t is a heavy hitter in the possible world W_{ij} in group G_i, as decided by the Space-Saving algorithm, and $X_{ij}^t = 0$ otherwise. Let $Y_i^t = \frac{1}{k}\sum_{j=1}^{k} X_{ij}^t$, and

$$Y^t = \texttt{Median}\{Y_1^t, Y_2^t, \ldots Y_l^t\}.$$

If $Y^t > \tau$, we assert that item t is a (ϕ, τ)-probabilistic heavy hitter. If $Y^t < (1 - \theta)\tau$, we assert that item t is not a probabilistic heavy hitter. Conclusions in cases where $(1 - \theta)\tau \leq Y^t \leq \tau$ could be arbitrary.

Analysis. We will prove that our algorithm finds all approximate (ϕ, τ)-probabilistic heavy hitters with with high probability. First we show that for any item t, our algorithm mistakenly classifies it with probability at most δ'.

Lemma 5 *For any item t, if $\Pr[m_t^R > \phi|R|] > \tau$, then $Y^t \geq (1 - \theta)\tau$ with probability at least $1 - \delta'$. If $\Pr[m_t^R > (\phi - \epsilon)|R|] < (1 - \theta)\tau$, then $Y^t < \tau$ with probability at least $1 - \delta'$.*

PROOF. Let $\mu = \mu_{ij}^t = \mathbf{E}[X_{ij}^t]$ for $1 \leq i \leq l, 1 \leq j \leq k$. We have the property that $\mu \geq \Pr[m_t^R > \phi|R|]$ if we use the Space-Saving algorithm to find the heavy hitters for each possible world W_{ij}, since Space-Saving only overestimates the frequency of a particular item.

Note that each X_{ij}^t is a Bernoulli variable with mean μ and variance $\mu - \mu^2$. So, for each random variable $Y_i^t, 1 \leq i \leq l$, we have

$$\mathbf{E}[Y_i^t] = \frac{1}{k}\sum_{j=1}^{k}\mathbf{E}[X_{ij}^t] = \mu, \text{ and,}$$

$$\mathbf{Var}[Y_i^t] = \frac{1}{k^2}\mathbf{Var}\left[\sum_{j=1}^{k} X_{ij}^t\right] = \frac{1}{k}\mathbf{Var}[X_{ij}^t] = \frac{1}{k}(\mu - \mu^2). \tag{7}$$

Thus, by Chebyshev's inequality, we have

$$\Pr[|Y_i^t - \mu| > \theta\mu] \le \frac{\mathbf{Var}[Y_i^t]}{\theta^2\mu^2} < \frac{1}{k\theta^2\mu}. \tag{8}$$

We prove the two assertions in the lemma separately. First, if $\Pr[m_t^R > \phi|R|] > \tau$, or $\mu > \tau$, plugging this fact and $k = \frac{8}{\theta^2\tau}$ to inequality (8), we get

$$\Pr[|Y_i^t - \mu| > \theta\mu] \le \frac{1}{\frac{8}{\theta^2\tau}\theta^2\mu} \le \frac{1}{8}.$$

Let I_i^t be the indicator variable such that

$$I_i^t = \begin{cases} 1, & \text{if } |Y_i^t - \mu| > \theta\mu; \\ 0, & \text{if } |Y_i^t - \mu| \le \theta\mu. \end{cases}$$

Let $I^t = \sum_{i=1}^{l} I_i^t$, thus we have $\mathbf{E}[I^t] \le \frac{l}{8}$. If the final value Y^t deviates by more than $\theta\mu$ from μ, there must be more than $\frac{l}{2}$ of the Y_i^t's that deviate more than $\theta\mu$ from μ since Y^t is the median of all Y_i^ts. According to the Chernoff inequality, we have

$$\Pr\left[I^t \ge \frac{l}{2}\right] = \Pr\left[I^t - \frac{l}{8} \ge 3\cdot\frac{l}{8}\right] \le e^{-\frac{3^2\cdot l/8}{2}} < \delta'.$$

Therefore with probability at least $1-\delta'$, $|Y^t - \mu| \le \theta\mu$. Together with $\mu > \tau$, we have $Y^t \ge (1-\theta)\tau$ with probability at least $1-\delta'$.

We prove the second assertion in the lemma in a similar fashion. Suppose that $\Pr[m_t^R > (\phi - \epsilon)|R|] < (1-\theta)\tau$, that is, $\mu < (1-\theta)\tau$, by Chebyshev's inequality,

$$\Pr[|Y_i^t - \mu| > \theta\tau] \le \frac{\mathbf{Var}[Y_i^t]}{\theta^2\tau^2} \le \frac{\mu}{k\theta^2\tau^2} < \frac{1}{8}. \tag{9}$$

Then we use the Chernoff inequality to prove that $\Pr[Y^t < \tau] \ge 1 - \delta'$, in the same way as above. □

According to Lemma 5, we conclude that our algorithm is correct with probability at least $1-\delta'$ for any particular item t. In order to guarantee that our algorithm is correct for *all* items simultaneously with probability $1-\delta$, we need to set $\delta' = \delta/n$ using the union bound. Here the pruning lemma (Lemma 3) comes in again to help lower the space and time bounds.

Lemma 3 and the discussion that follows tell us that there are at most $\frac{3}{\phi\tau}$ items who have a chance of being probabilistic heavy hitters, and we can find all such candidates by running a weighted version of Space-Saving to find all the $\left(\frac{\phi\tau}{3}\right)$-expected heavy hitters. Note that, however, the actual number of candidates we find might be a little more than $\frac{\phi\tau}{3}$ since the Space-Saving algorithm for EHH will introduce some errors. Nevertheless, if we use $\epsilon' = \frac{\phi\tau}{12}$ in Space-Saving, the number of candidates found will be bounded by $\frac{4}{\phi\tau}$.

By this observation, we could run the weighted Space-Saving to find all the candidate probabilistic heavy hitters in parallel. At the end we only check those candidate items t whether Y^t is greater than τ (or smaller than $(1-\theta)\tau$). Those non-candidates are discarded directly. Consequently, we only need to guarantee that our sampling algorithm is correct on these at most $\frac{4}{\phi\tau}$ candidates. So if we choose $\delta' = \frac{\phi\tau}{4}\delta$, the overall error would be bounded by δ. We have the following theorem. Notice that both time and space are independent on n.

Theorem 2 *The basic sampling algorithm finds all approximate (ϕ,τ)-probabilistic heavy hitters with probability at least $1-\delta$, using $O(\frac{1}{\epsilon\theta^2\tau}\log(\frac{1}{\delta\phi\tau}))$ memory and $O(\frac{1}{\theta^2\tau}\log(\frac{1}{\delta\phi\tau}) + \log(1/\epsilon))$ processing time for each x-tuple.*

Note that the extra term $O(\log(1/\epsilon))$ in the processing time for each x-tuple comes from the weighted Space-Saving algorithm.

3.2 Improved Sampling Algorithm

The basic sampling algorithm above requires considerable time to process each tuple. For each incoming x-tuple T, we have to repeat sampling for $O(\frac{1}{\theta^2\tau}\log(\frac{1}{\delta\phi\tau}))$ times, one for each possible world W_{ij} maintained. This limits our capability to process high-speed streaming data in real time when θ is small. In this subsection, we give an improved sampling algorithm that dramatically reduces the expected processing time. The worst case memory consumption remains the same, however in practice it also reduces significantly as evident by our experimental study, see Figure 3(b).

The improvement is based on the following two observations. First, if we reduce the sampling rate for all items in the probabilistic data stream by the same factor, then the sample possible world obtained is a "scaled-down" version of the original possible world, and the original heavy hitters are still very likely to be the heavy hitters in the scaled-down world. Second, in order for the analysis of the basic sampling algorithm to go through, all we need is to ensure that in each group G_i, the k sample possible worlds W_{ij} are pairwise independent, such that the inequalities (8), (9) continue to hold; we don't require those samplings to be totally independent. Below we first give our improved algorithm, then prove its correctness by formalizing these observations.

The improved sampling algorithm. Our general idea is trying to reduce the sampling times for each x-tuple in each group such that, for a particular group $G_i, 1 \le i \le l$, when a new x-tuple T comes, only a small number of possible worlds in G_i will get an item from T. We call a possible world getting an item from T a *lucky* possible world. There are several conditions we have to meet.

1. There should be only a small number (constant if possible) of lucky possible worlds in each group G_i.
2. We have to find all the lucky possible worlds quickly (in constant time if possible), otherwise there would be no improvement in running time compared with the basic sampling algorithm.
3. We have to make sure that the samplings for the possible worlds in group G_i are pairwise independent so that the inequalities (8), (9) continue to hold.
4. The modification will not affect the precision of the basic sampling algorithm too much.

For each x-tuple T, our improved sampling algorithm consists of two steps. In the first step, we choose zero, one, or all k possible worlds from each group G_i as the lucky possible worlds. In the second step, for each lucky possible world W_{ij}, we sample an item t from T according to its probability $p(t)$ and add it to W_{ij}. The second step is the same as before, so we only describe below how the first step is performed.

For each group G_i, let $I_{i1}, \dots, I_{ik}$ be the indicator random variables of the possible worlds being lucky, i.e., $I_{ij} = 1$ if W_{ij} is a lucky possible world and $I_{ij} = 0$ otherwise. Of course we cannot afford to explicitly generate these I_{ij}'s, so as to meet condition 2 above. Instead, we give a fast scheme that finds all the j's such that $I_{ij} = 1$.

Scheme 1 Improved sampling scheme

- *With probability $1/k^2$, set $I_{ij} = 1$ for all $1 \le j \le k$.*
- *With prob. $(k-1)/k^2$, set $I_{ij} = 0$ for all $1 \le j \le k$.*
- *Otherwise, select j^* uniformly at random from $\{1, \dots, k\}$ and set $I_{ij^*} = 1$ and $I_{ij} = 0$ for all $j \ne j^*$.*

Analysis. We will prove that our scheme meets all the four criteria above.

Lemma 6 *The expected number of lucky worlds per group is $O(1)$, and it takes $O(1)$ time to identify them.*

PROOF. Since

$$\mathbf{E}[I_{ij}] = 1 \cdot \frac{1}{k^2} + \frac{1}{k} \cdot \left(1 - \frac{1}{k^2} - \frac{k-1}{k^2}\right) = \frac{1}{k},$$

the expected number of lucky worlds per group is $\mathbf{E}[\sum_{j=1}^{k} I_{ij}] = 1$.

Since we only care about those I_{ij}'s where $I_{ij} = 1$, the expected running time to find them is also $O(1)$. □

Lemma 7 *For any fixed i, the random variables I_{ij} $(1 \le j \le k)$ generated this way are pairwise independent.*

PROOF. Following the definition of pairwise independence, we prove that for all $x, y \in \{0, 1\}$, and $j \ne j', 1 \le j, j' \le k$,

$$\Pr[I_{ij} = x \wedge I_{ij'} = y] = \Pr[I_{ij} = x] \Pr[I_{ij'} = y].$$

First, $\Pr[I_{ij} = 1] = \mathbf{E}[I_{ij}] = 1/k$ and $\Pr[I_{ij} = 0] = 1 - 1/k$, for all j.

For the case $x = y = 1$, we know that $\Pr[I_{ij} = 1 \wedge I_{ij'} = 1] = 1/k^2$ since both possible W_{ij} and $W_{ij'}$ are lucky only when all the possible worlds in group G_i are lucky. Therefore $\Pr[I_{ij} = 1 \wedge I_{ij'} = 1] = \Pr[I_{ij} = 1] \Pr[I_{ij'} = 1]$.

For the case $x = y = 0$, we have

$$\begin{aligned}\Pr[I_{ij} = 0 \wedge I_{ij'} = 0] &= \frac{k-1}{k^2} + \left(1 - \frac{1}{k}\right)\frac{k-2}{k} \\ &= \frac{k^2 - 2k + 1}{k^2} \\ &= \Pr[I_{ij} = 0] \Pr[I_{ij'} = 0].\end{aligned}$$

For the case $x = 1, y = 0$, we have

$$\begin{aligned}\Pr[I_{ij} = 1 \wedge I_{ij'} = 0] &= \left(1 - \frac{1}{k}\right)\frac{1}{k} \\ &= \Pr[I_{ij} = 1] \Pr[I_{ij'} = 0].\end{aligned}$$

The case $x = 0, y = 1$ is symmetric to the case above. □

It remains to show that the improved sampling algorithm still finds all the PHH's with high probability. We make the following observation concerning the difference between

Algorithm 1: Finding Probabilistic Heavy Hitter in Data Streams

```
1  $l = O(\ln(1/\phi\tau\delta))$;  /* number of groups of possible worlds */
2  $k = O(\frac{1}{\theta^2\tau})$;  /* number of possible worlds in each group */
3  let PHH the set of probabilistic heavy hitters found by the algorithm as the output;
4  let $SS_{\texttt{unwei}}$ denotes the unweighted version of the Space-Saving algorithm and $SS_{\texttt{wei}}$ the weighted version.
5  for $i = 1, \dots, l$ do
6      for $j = 1, \dots, k$ do
7          allocate an array of $1/\epsilon$ cells for each possible world $W_{ij}$ for algorithm $SS^{ij}_{\texttt{unwei}}(\phi, \epsilon)$;
8  allocate an array of $1/\epsilon$ cells for the (global) algorithm $SS_{\texttt{wei}}(\frac{\phi\tau}{3}, \epsilon)$;
9  for each x-tuple $T$ do
10     for all tuples $(t, p(t))$ in $T$ do
11         feed item $t$ together with its confidence $p(t)$ to $SS_{\texttt{wei}}(\frac{\phi\tau}{3}, \epsilon)$;
12     for $i = 1, \dots, l$ do
13         toss an unbiased coin whose value is uniformly distributed over the range $\{1, 2, \dots, k^2\}$ and let $r$ be the result.
14         if $r = 1$ then
15             include all $j$ $(1 \le j \le k)$ to $S_i$;
16         else if $2 \le r \le k$ then
17             let $S_i$ to be empty set;
18         else if $(k-1) * j + 2 \le r \le (k-1) * (j+1) + 1$ for some $j \in \{1, 2, \dots, k\}$ then
19             include $j$ to $S_i$;
20         for all $j \in S_i$ do
21             randomly choose one tuple containing item $t$ from $T$ with probability $p(t)$;
22             feed item $t$ into $SS^{ij}_{\texttt{unwei}}(\phi, \epsilon)$;
                                                        /* final checking step */
23 let $C$ be the set of candidate probabilistic heavy hitter computed by algorithm $SS_{\texttt{wei}}(\frac{\phi\tau}{3}; \epsilon)$;
24 for $i = 1, \dots, l$ do
25     for $j = 1, \dots, k$ do
26         if $t$ is a heavy hitter in the possible world $W_{ij}$ by $SS^{ij}_{\texttt{unwei}}(\phi, \epsilon)$ then
27             let $x^t_{ij} = 1$;
28         else
29             let $x^t_{ij} = 0$;
30     let $y^t_i = \frac{1}{k}\sum_{j=1}^{k} x^t_{ij}$;
31 for all $t \in C$ do
32     let $y^t = \texttt{Median}\{y^t_1, y^t_2, \dots y^t_l\}$;
33     if $y^t > \tau$ then include $t$ to PHH;
```

the improved sampling algorithm and the basic sampling algorithm. In the improved algorithm, each possible world W_{ij} can be seen as being sampled from a modified data stream $\mathcal{D}'$ in which each probability $p(t)$ is decreased by a factor of k, since $Pr[I_{ij} = 1] = 1/k$. Now, we need to show that this modification does not affect too much the probability of an item t becoming a heavy hitter in W_{ij}.

More precisely, we need to show that if t is a PHH in $\mathcal{D}$, then it still has a high probability to be a PHH in $\mathcal{D}'$; and if t is not a PHH in $\mathcal{D}$, then with high probability, it is not a PHH in $\mathcal{D}'$, either.

Formally, let R' be a random possible world generated from the modified uncertain data stream $\mathcal{D}'$, and let $m_t^{R'}$ be the multiplicity of item t in R', then we have the following result.

Lemma 8 *For any $0 < \zeta < 1$, with probability at least $1 - 8e^{-\frac{\zeta^2}{3}\cdot\frac{\phi\tau}{4}\cdot\frac{\mathbf{E}[|R|]}{k}}$, we have*

1. *If $\Pr[m_t^R > \phi|R|] > \tau$, then $\Pr[m_t^{R'} > (1-4\zeta)\phi|R'|] > \tau$.*
2. *If $\Pr[m_t^R > (\phi-\epsilon)|R|] < (1-\theta)\tau$, then $\Pr[m_t^{R'} > (1+5\zeta)(\phi-\epsilon)|R'|] < (1-\theta)\tau$.*

PROOF. By the pruning lemma (Lemma 3) and the discussion that follows, we only have to consider items with $\frac{\mathbf{E}[m_t^R]}{\mathbf{E}[|R|]} \geq \frac{\phi\tau}{4}$. Let $\mu_1 = \mathbf{E}[|R|]$ and $\mu_2 = \mathbf{E}[m_t^R]$. Applying the Chernoff inequality on both R and m_t^R, we have

$$\Pr\left[||R| - \mu_1| \geq \zeta\mu_1\right] < 2e^{-\zeta^2\mu_1/3},$$

and

$$\Pr[|m_t^R - \mu_2| \geq \zeta\mu_2] < 2e^{-\zeta^2\mu_2/3}.$$

So, by the union bound, with probability at least $1-4e^{-\zeta^2\mu_1/3}$,

$$\frac{(1-\zeta)\mu_2}{(1+\zeta)\mu_1} \leq \frac{m_t^R}{|R|} \leq \frac{(1+\zeta)\mu_2}{(1-\zeta)\mu_1}. \tag{10}$$

Similarly, let $\mu_3 = \mathbf{E}[|R'|]$ and $\mu_4 = \mathbf{E}[m_t^{R'}]$ and apply the Chernoff inequality on R' and $m_t^{R'}$, we obtain that with probability at least $1 - 4e^{-\zeta^2\mu_3/3}$,

$$\frac{(1-\zeta)\mu_4}{(1+\zeta)\mu_3} \leq \frac{m_t^{R'}}{|R'|} \leq \frac{(1+\zeta)\mu_4}{(1-\zeta)\mu_3}. \tag{11}$$

Since $\mu_2/\mu_1 = \mu_4/\mu_3$ and $\min\{\mu_i : i = 1,2,3,4\} \geq \frac{\phi\tau}{4}\cdot(\mu_1/k)$, by (10) and (11), we conclude that with probability at least $1 - 8e^{-\frac{\zeta^2}{3}\cdot\frac{\phi\tau}{4}\cdot\frac{\mu_1}{k}}$,

$$(1-4\zeta)\frac{m_t^R}{|R|} \leq \frac{m_t^{R'}}{|R'|} \leq (1+5\zeta)\frac{m_t^R}{|R|},$$

which implies the two statements in the lemma. $\square$

Note that the success probability in Lemma 8 approaches 1 exponentially fast as $E[|R|]$, the expected size of the data stream, increases. Moreover, our analysis in the proof is very loose, and the actual error probability, as we verified empirically in Section 5, is in fact much smaller than the bound given in the lemma.

By Lemma 8, if we can decide whether $\Pr[m_t^{R'} > (1-4\zeta)\phi|R'|] > \tau$ or $\Pr[m_t^{R'} > (1+5\zeta)(\phi-\epsilon)|R'|] < (1-\theta)\tau$ for all items t, we can guarantee that the algorithm does not have false positive. We still need to argue that all items returned are indeed heavy hitters. The key point here is that by setting ζ small enough, say $\zeta = \epsilon/10$, there is a gap of $\epsilon/10$ between $(1+5\zeta)(\phi-\epsilon) = (\phi - \frac{2}{5}\epsilon)$ and $(\phi - \frac{1}{2}\epsilon)$, which means that the two conditions in lemma 8 are disjoint.

More precisely, by setting $\zeta = \epsilon/10$, it suffices to decide whether

$$\Pr[m_t^{R'} > (\phi - \frac{2}{5}\epsilon)|R'|] > \tau \text{ or,} \tag{12}$$

$$\Pr[m_t^{R'} > (\phi - \frac{1}{2}\epsilon)|R'|] < (1-\theta)\tau. \tag{13}$$

If an item satisfies (13), then it must not comply with (12), hence cannot be a PHH; if an item satisfies (12), then it must not comply with (13), hence cannot be a non-PHH. Also, since our algorithm is allowed to either return or not return those items which are neither PHH nor non-PHH, the correctness of the algorithm is guaranteed. Furthermore, by properly setting the parameters in the unweighted Space-Saving algorithm, i.e., setting $\phi' = \phi - \frac{2}{5}\epsilon$ and $\epsilon' = \frac{1}{10}\epsilon$, we can correctly determine whether (12) or (13) holds for each item.

The overall improved sampling algorithm is detailed in Algorithm 1. The following theorem summarizes its space and processing time bounds.

Theorem 3 *The improved sampling algorithm correctly finds all the approximate (ϕ,τ)-probabilistic heavy hitters in a data stream with high probability, using $O(\frac{1}{\epsilon}\frac{1}{\theta^2\tau}\log(\frac{1}{\delta\phi\tau}))$ memory and $O(\log(\frac{1}{\delta\phi\tau}) + \log(1/\epsilon))$ expected processing time for each x-tuple.*

3.3 Generalized Sampling Algorithm

Our empirical study (Section 5.2) shows that, as expected, the improved sampling algorithm is order-of-magnitude faster than the basic sampling scheme. However, on the other hand, its accuracy is also lower than that of the basic scheme, especially on small data sets, due to the reduced sampling rate in each possible world. In the following, we present a generalized scheme to trade processing time for better accuracy, so that the accuracy of the improved scheme approaches that of the basic scheme, while still being highly efficient.

Our tradeoff scheme is a generalization of the improved sampling scheme presented in Section 3.2. Instead of choosing 1 lucky possible worlds (in expectation) in each group for each incoming x-tuple, we pick roughly s (more precisely, $s + o(s)$) lucky possible worlds at a time. Note that when $s = 1$, it is equivalent to the improved sampling scheme shown in section 3.2, and when $s = k$, it is the same as the basic sampling scheme shown in Section 3.1. The flexibility of choosing s between 1 and k provides us with the ability to control the tradeoff between accuracy and processing time.

More precisely, for each group, we repeatedly perform Scheme 1 for s times, and declare a possible world to be lucky if it is lucky in at least one of the s trials of Scheme 1.

We briefly show that this generalized sampling scheme also meets the four criteria specified before, that is, (1) the expected number of lucky possible worlds is $O(s)$; (2) the $O(s)$ lucky possible worlds could be quickly found, in time $O(s)$; and (3) the whole sampling scheme is pairwise independent; (4) the modification will not affect the precision of the algorithm too much. The proofs are similar as those shown in section 3.2.

As before, let $I_{ij} = 1$ if W_{ij} is lucky, and 0 otherwise. Let $I_{ij}^{\ell} = 1$ if W_{ij} is lucky in the ℓ-th trial of Scheme 1. We have $I_{ij} = \max_{1\leq\ell\leq s} I_{ij}^{\ell}$.

First, since $\Pr[I_{ij}^{\ell} = 1] = 1/k$,

$$\Pr[I_{ij} = 1] = 1 - \left(1 - \frac{1}{k}\right)^s = \frac{s}{k} + o\left(\frac{1}{k}\right),$$

thus the expected number of lucky possible worlds per group is $\mathbf{E}[\sum_{j=1}^{k} I_{ij}] = O(s)$. The second criterion is met since we only need to generate a random number s times for each x-tuple. Third, to show that I_{ij} $(1 \le j \le k)$ are pairwise independent, we observe that for any $j \neq j'$, all the $2s$ random variables, $I_{ij}^{\ell}, I_{ij'}^{\ell}$, $\ell = 1, \ldots, s$, are mutually independent. Finally, we could use a similar argument as lemma 8 to show that the modification has little impact on the precision of the whole algorithm.

4. TOP-K QUERIES

We can easily adapt our algorithms to answer top-k queries, returning the k items with the largest confidence of being heavy hitters. The confidence of an item t is simply $\Pr[m_t^R > \phi|R|]$ and we denote this confidence as $\rho(t)$. Let t_k be the item with the k-th largest $\rho(t)$. Ideally, we would want to return and only return those items t with $\rho(t) \ge \rho(t_k)$. One can achieve this by our exact algorithms in Section 2. As exact computation is quite expensive, if approximation is allowed, we can adapt our sampling algorithms to answer top-k queries efficiently. Since our algorithm provides an approximate confidence Y^t for $\rho(t)$, we can simply output the k items with the top k largest Y^t's. We show that this solution guarantees:

- All items t with $\rho(t) > (1 + \theta)\rho(t_k)$ will be returned.
- For an item t, consider the uncertain data set $\mathcal{D}'$ after adding ϵm x-tuples $\{(t, 1)\}$ into $\mathcal{D}$. If in $\mathcal{D}'$, we still have $\Pr[m_t^{R'} > \phi|R'|] < (1 - \theta)\rho(t_k)$, where R' is a random possible world instantiated from $\mathcal{D}'$, then t will not be returned.

The above two approximation guarantees follow directly from our approximation algorithms and the definition of approximate probabilistic heavy hitters.

5. EXPERIMENTS

We have implemented the proposed algorithms, including the Space-Saving algorithm, and studied their efficiency and effectiveness using both real and synthetic data sets. The experimental evaluation is designed to investigate a number of key issues: the efficiency of the exact algorithms, the approximation quality and scalability of both the basic and improved, one-pass sampling algorithms, the tradeoff between efficiency and effectiveness when comparing the basic sampling algorithm and the improved algorithm (and its generalization), and finally the impact of various characteristics in the uncertain data such as the skewness of popularity distribution, the number of unique items, the number of x-tuples and the correlation between item popularity and probability. All algorithms are implemented using C++ under Linux. The experiments are performed in a Linux box with a P4 2.8GHz CPU and 2 GB of memory.

Data sets. We used two real data sets and a number of synthetic data sets. The first real data set is obtained from the MystiQ project[1]. It contains probabilistic movie records

data	n	freq distr	prob distr & correlation
movie	112	not skew	not skew, correlated
wcday46	2931	skew	skew, not correlated
zipfu1.60	1840	skew	uniform, not correlated

Table 1: The data sets (first $100,000$ xtuples).

reflecting the matching probability as a result of data integration from multiple sources. This data set, referred to as *movie*, does not have a skewed popularity distribution. It has a total of approximately $100,000$ x-tuples, most of which have only one alternative, but some have a few. The second real data set is obtained from the 1998 World Cup web server request traces, available at the Internet Traffic Archive. In particular we used the trace from day 46 (referred to as *wcday46*). Each record in the trace represents an entry in the access log of the world cup web server. Specifically, it contains a time stamp, an object id (the particular web page being requested), a status code, etc. The status code indicates the response from the server to the client chosen from 38 unique values. We first computed the distribution of the status code of all records, and converts the status field to a probability according to this distribution. We then group records whose time stamps are close into an x-tuple. This data set has a considerable skewed popularity distribution. Since most of requests are served successfully and the status code is 200, it also has a very skewed distribution on the items' probability. The *wcday46* data set contains roughly half a million x-tuples.

We also generated synthetic data sets with various data characteristics. Specifically, we generated data sets following Zipfian distributions with the adjustable skewness parameter a ranging from 1.1 (least skewed) to 1.9 (most skewed). The probability of items follows various distributions and it was generated to be strongly, or totally not, or negatively correlated with the item's popularity. Strongly correlated probability means that if the item is highly popular, then its probability will be higher as well. The item popularity distribution, probability distribution and the parameter a are combined to name a synthetic data set, e.g., *zipfu1.60*. As a final remark, the number of unique items, n, plays an important role for the cost of various algorithms and they are summarized in Table 1 for the first $100,000$ x-tuples.

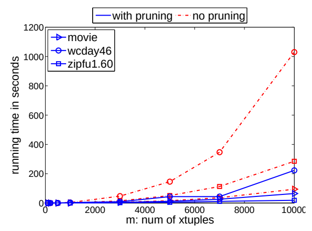
5.1 Exact Algorithms

The exact algorithms perform as indicated in our analysis, either with quadratic, or cubic time complexity depending on x-tuples have single item or multiple items. To produce data sets with single-item x-tuples, for each of the data sets described above, we simply retain only the first item in each x-tuple. The running time is shown in Figure 2(a) and 2(b), respectively. Clearly, the exact algorithms are costly and do not scale when data sets increase, although they are able to return exact results. The *wcday*46 data set is the most expensive due to high number of unique items. We also observe that the pruning lemma indeed dramatically reduces the cost. Figure 2(d) further illustrates the effectiveness of the pruning lemma, where for skewed data sets, more than 90% of the items are pruned. Obviously, the higher ϕ and τ are, the more items will be pruned. We also reported in Figure 2(c) the memory usage of the two exact algorithms, clearly demonstrating either a linear or quadratic trend.

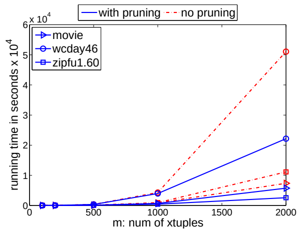
5.2 Approximation algorithms: streaming data

Motivated by the high costs of the exact algorithms, as

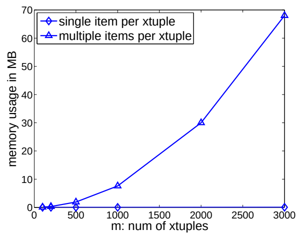
[1]www.cs.washington.edu/homes/suciu/project-mystiq.html



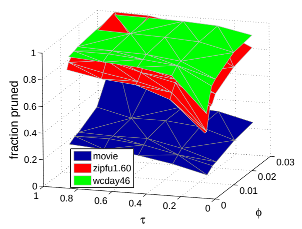
(a) single item.



(b) multiple items.



(c) memory usage.



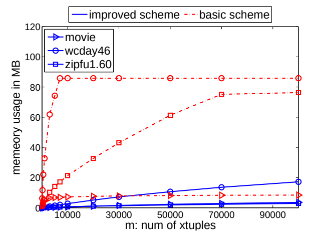
(d) pruning power.

Figure 2: Exact algorithms: running time and memory usage analysis, $\phi = 0.01$, $\tau = 0.5$.

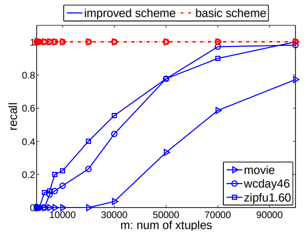
	improved	basic
movie	$3.03*10^{-5}$	$5.5*10^{-3}$
wcday46	$9.83*10^{-5}$	$2.7*10^{-2}$
zipfu1.60	$4.05*10^{-5}$	$9.8*10^{-3}$

time in seconds

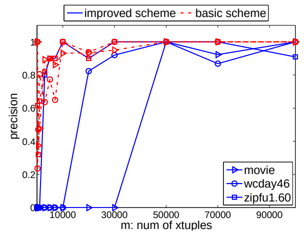
(a) update cost per x-tuple.



(b) memory usage.



(c) recall.



(d) precision.

Figure 3: Approximate, streaming algorithms: $\phi = 0.01$, $\tau = 0.8$, $\delta = 0.05$, $\theta = 0.05$, $\epsilon = 0.001$.

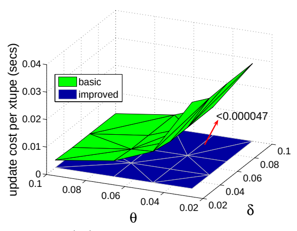
well as the need to handle streaming data, we have designed one-pass approximation algorithms. Experimental results show that the basic sampling scheme has excellent accuracy. Its processing time is much lower than the exact algorithm, but is still too costly to operate under a high-speed streaming environment. The improved sampling scheme has excellent low cost, and very good approximation quality for relatively large m. Finally, our generalized algorithm gives a simple, adjustable tradeoff between the basic and improved scheme. In the following, we present the detailed experimental results. Unless specified otherwise, the default values for various parameters are set as follows: $m = 100,000$, $\phi = 0.01$, $\tau = 0.8$, $\delta = 0.05$, $\theta = 0.05$ and $\epsilon = 0.001$. To measure the approximation quality, we adopt the common *recall* (returned number of true PHH/ total number of true PHH) and *precision* (returned number of true PHH/total returned number of PHH) metrics.

Basic scheme vs. improved scheme. We first compare the processing time of the basic scheme and the improved scheme. Note that the update cost per x-tuple does not depend on the size of the data set, so we report the average processing time for each x-tuple on three data sets in Figure 3(a). It shows that the improved scheme is 2 orders of magnitude faster, which makes it able to handle close to 10,000 x-tuples per second. Also, the number of unique items in a data set does impact the update cost a bit due to the difference in updating an existing item or inserting a new item (possibly with a replacement) in the Space-Saving algorithm. The memory usage (Figure 3(b)) for the basic scheme initially increases with m and eventually levels off when all the $1/\epsilon$ cells in each sample possible world are filled up. Similar argument goes for the improved scheme. However, the worlds are filled up much more slower in this case. The savings in both the update cost and memory usage of the improved scheme could be attributed to the fact that on expectation only one world in each group is updated for every x-tuple, whereas the basic scheme has to update all worlds. As a result, the improved scheme only consumes less than few megabytes of memory. Finally, basic scheme exhibits excellent approximation quality in terms of both recall (Figure 3(c)) and precision (Figure 3(d)) even with small m, whereas the improved scheme does require m to be sufficiently large to be accurate. Figure 3(c) and 3(d) indicate that after $m = 50,000$ it becomes very close to the basic scheme.

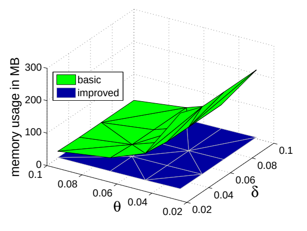
Next we investigate the effects of various parameters on the algorithms' efficiency and approximation quality, starting with δ and θ. Recall that δ bounds the overall failure probability of the approximation and θ bounds the error of approximation. For conciseness, only results from the zipfu1.60 data set is reported, reason being that it represents an "average" of the three data sets on various key characteristics (see Table 1). It is clear from the theoretical analysis that the smaller δ and θ get, the more expensive the algorithms become. This has been reflected from Figure 4(a) and 4(b). In all cases, the improved scheme is very insensitive to such changes and much cheaper than the basic scheme. In terms of recall (Figure 4(c)), for a relative large m (100,000 in this case) both schemes find all true PHH insensitively for reasonable values of δ and θ. Finally, the improved scheme in general performs better with smaller θ and δ in terms of precision and it is more sensitive to the change of θ, as evident in Figure 4(d).

Continuing with ϕ and τ, the results are reported in Figure 5. As expected, smaller ϕ and τ values lead to increase in update cost and memory usage, as shown in Figure 5(a) and 5(b), and the improved scheme is very insensitive to such changes. Finally, Figure 5(c) and 5(d) for extremely large value of τ the recall of the improved scheme drops; while for combination of extremely large ϕ and small τ values the precision of the improved scheme drops. For both cases, the basic scheme retains high approximation quality.

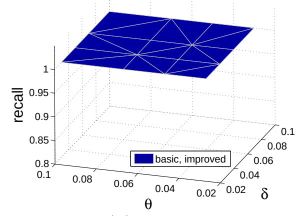
Finally, we investigate how the correlation between the item's popularity and its probability impact our approximation algorithms. To manifest such effects, we generated data sets that are strongly, or not-correlated, or negatively correlated between the two, using the zipfu1.60 data set as the basis (note that for strongly and negatively correlated data sets, the probability distribution will no longer be uni-



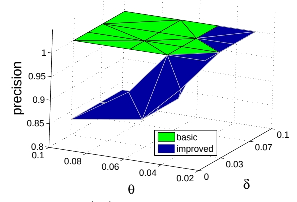
(a) update cost.



(b) memory usage.

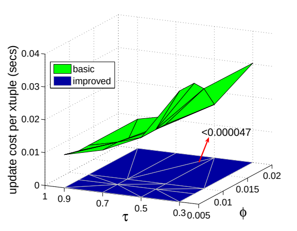


(c) recall.

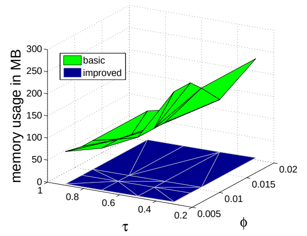


(d) precision.

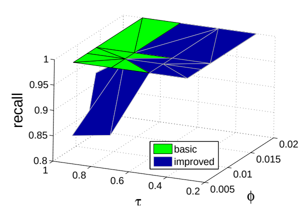
Figure 4: Varying δ and θ: zipfu1.60, $m = 100000$, $\phi = 0.01$, $\tau = 0.8$, $\epsilon = 0.001$.



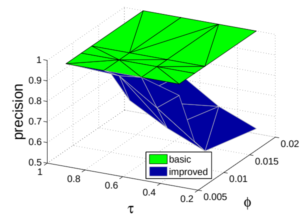
(a) update cost.



(b) memory usage.



(c) recall.



(d) precision.

Figure 5: Varying ϕ and τ: zipfu1.60, $m = 100000$, $\delta = 0.05$, $\theta = 0.05$, $\epsilon = 0.001$.

form). For brevity we only report the results on varying θ in Figure 6. Interestingly, we learn that the degree of correlation has opposing effects on update cost and memory usage. Figure 6(a) shows that positive correlation increases the update cost while negative correlation reduces it. This is due to the fact that the higher probabilities for more frequent items lead to higher effective sample rates in the positive correlation case, whereas in the negatively correlation case, we have the opposite. In terms of memory usage, Figure 6(b) shows the reverse effect. This is explained by the fact that lower probabilities for infrequent items in the positive correlation case effectively reduce the unique number of items sampled. Finally, for the improved scheme Figure 6(c) shows that positive correlation increases the approximation quality on recall and negative correlation tampers it. The impact on precision is not obvious as reflected in Figure 6(d). This is due to the fact that negative correlation increases the chances that the approximate algorithm may miss some boundary PHH (hence reduce the recall). However, it will increase the chance that it misses some falsely identified boundary non-PHH as well (hence the precision is not impacted). It should be highlighted that a high degree of negative correlation is generated to demonstrate its effect and the basic scheme retains high approximation quality.

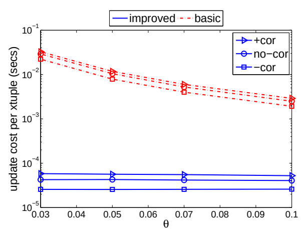
Generalized scheme: tradeoff between cost and accuracy. The above experimental results show that the basic scheme and the improved scheme are actually the two extremes in the tradeoff between processing time and accuracy. Especially for small m, the basic scheme has better accuracy but also high cost, while the improved scheme has low processing cost but the accuracy is not satisfactory. Our generalized scheme thus provides a smooth transition between the two extremes, and we would like to see how much processing time we exactly need to sacrifice in order to boost the accuracy.

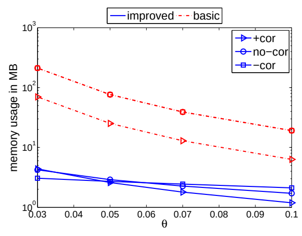
We performed experiments with different s, which controls the number of the possible worlds updated for each x-tuple (recall that in the generalized scheme, we update $s + o(s)$ out of k worlds in each group G_i). Figure 7 shows the performance of the generalized algorithm as a function of s/k, namely the fraction of possible worlds updated on each x-tuple. This set of experiments gives very encouraging results: both the recall and precision of the generalized algorithm approach 1 very quickly as s increases, while the update cost and memory usage only increase linearly in s. For s/k as small as 0.05, its accuracy is already very close to perfect, keeping in mind that $s/k = 0.05$ corresponds to a 20-fold speedup from the basic scheme.

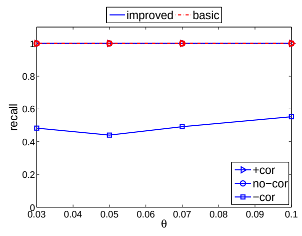
Further issues. We can also easily adapt our algorithms for top-k query processing. As the analysis in Section 4 points out, the cost stays the same and in practice the approximation quality is good as well. Another factor in the cost of approximation algorithms is the number of xtuples having multiple items and how many items it may have. Note that they do not affect the approximation quality and only has a small impact to the cost (assuming that the number of items an xtuple may take is a constant). This study is omitted for brevity. Our experimental study suggests that for typical values of ϕ and τ, setting δ and θ with values ranging between $[0.05, 0.1]$ are good enough. The value of ϵ bounds the approximation error of item frequency at each world and a value in $[0.001, 0.05]$ is sufficient for typical ϕ values. Lastly, as Figure 7 has revealed, the generalized scheme is preferred over the basic and improved schemes with a small s ($\leq 0.1k$, k is fixed given θ and τ).

6. RELATED WORK

Many efforts have been devoted to modeling and processing uncertain data and a complete survey of this area is beyond the scope of this paper. Nevertheless, TRIO [2, 7, 36], MayBMS [5, 6] and Probabilistic Databases [13] are promising systems that are currently under developing. General query processing techniques have been extensively studied under the possible worlds semantics [9, 13, 19, 37], and important query types with specific query semantics are explored in more depths, such as top-k queries [34, 39] and skyline queries [33]. It is interesting to observe that many problems in traditional deterministic data management be-


(a) update cost.


(b) memory usage.


(c) recall.

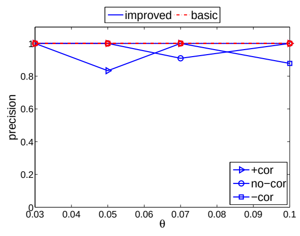
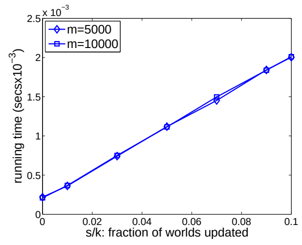
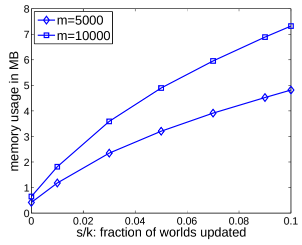
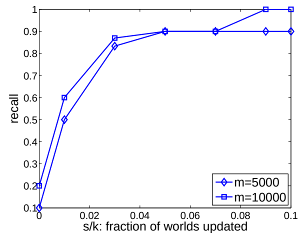

(d) precision.

Figure 6: Popularity-probability correlation, varying θ: $m = 100000$, $\delta = 0.05$, $\phi = 0.01$, $\tau = 0.8$, $\epsilon = 0.001$.


(a) update cost.


(b) memory usage.


(c) recall.

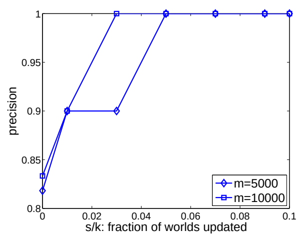

(d) precision.

Figure 7: Generalized scheme: tradeoff in cost/accuracy, varying s, $\delta = 0.05$, $\theta = 0.05$, $\phi = 0.01$, $\tau = 0.8$, $\epsilon = 0.001$.

come much more challenging and require special treatment under the uncertain data model. Examples include our work on heavy hitters, indexing techniques [38, 40, 29], view management [35], and many more.

The most relevant works to this paper are [10, 24], where the authors study the problem of estimating various statistical aggregates on probabilistic streams, and in particular, EHH has been discussed in [10] on probabilistic streams. Our work formalizes the concept of probabilistic heavy hitters with confidence, rather than relying on the expectation over all possible worlds, and proposes efficient algorithms in the more general x-relation model for both the offline and streaming case, hence is fundamentally different from [10].

Another set of related work is the study of heavy hitters in deterministic data management. Lossy counting [31] and majority counting [27], together with [12, 25], for data streams are among the first in the literature, followed by the improvement in [32]. The solutions to our problem in the streaming case could be viewed as non-trivial extensions to these techniques under the new model. Many other works exist for deterministic heavy hitters with different twists, e.g., heavy hitters in multidimensional data [11], and considering the distributed environment [30], etc. The idea of using sampling to obtain approximate query results could be traced back to [20].

Finally, items from an x-tuple in the x-relation model could be viewed as a simplified representation of correlations in uncertain data, where the correlations only include mutual exclusion. More advanced models must be used for complex correlations. Recent works based on graphical probabilistic models and Bayesian networks have shown promising results in both offline [37] and streaming data [26].

7. REPEATABILITY ASSESSMENT RESULT

Most of experimental results (Figures 2, 3a, 3b, 4a, 4b, 5, 6 7) have been verified by the experiment repeatability committee.

8. CONCLUSION

This is the first work with a comprehensive study of finding frequent items in probabilistic data. We formalize the notion of probabilistic heavy hitters following the commonly adopted possible world query semantics in uncertain databases. Efficient algorithms with theoretical guarantees have been presented for both offline and streaming data, under the widely adopted x-relation model. Future work includes handling distributed data, and more interestingly, supporting other uncertain data models [36], for example graphical probabilistic models [37]. Our study also opens the door for exploring frequent item sets and association rule mining in the context of uncertain data mining [1].

9. ACKNOWLEDGMENT

We thank the anonymous reviewers for the insightful comments.

10. REFERENCES

[1] C. Aggarwal. On density based transforms for uncertain data mining. In *ICDE*, 2007.

[2] 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.

[3] R. Agrawal and R. Srikant. Fast algorithms for mining association rules in large databases. In *VLDB*, 1994.

[4] N. Alon, Y. Matias, and M. Szegedy. The Space Complexity of Approximating the Frequency Moments. *Journal of Computer and System Sciences*, 58(1):137–147, 1999.

[5] L. Antova, C. Koch, and D. Olteanu. 10^{10^6} worlds and beyond: Efficient representation and processing of incomplete information. In *ICDE*, 2007.

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

[7] O. Benjelloun, A. D. Sarma, A. Halevy, and J. Widom. ULDBs: databases with uncertainty and lineage. In *VLDB*, 2006.

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

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

[10] G. Cormode and M. Garofalakis. Sketching probabilistic data streams. In *SIGMOD*, 2007.

[11] G. Cormode, F. Korn, S. Muthukrishnan, and D. Srivastava. Diamond in the rough: finding hierarchical heavy hitters in multi-dimensional data. In *SIGMOD*, 2004.

[12] G. Cormode and S. Muthukrishnan. What's hot and what's not: tracking most frequent items dynamically. In *PODS*, 2003.

[13] N. Dalvi and D. Suciu. Efficient query evaluation on probabilistic databases. In *VLDB*, 2004.

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

[15] E. D. Demaine, A. López-Ortiz, and J. I. Munro. Frequency estimation of internet packet streams with limited space. In *ESA*, 2002.

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

[17] C. Estan and G. Varghese. New directions in traffic measurement and accounting. In *SIGCOMM*, 2002.

[18] M. Fang, N. Shivakumar, H. Garcia-Molina, R. Motwani, and J. D. Ullman. Computing iceberg queries efficiently. In *VLDB*, 1998.

[19] A. Fuxman, E. Fazli, and R. J. Miller. ConQuer: efficient management of inconsistent databases. In *SIGMOD*, 2005.

[20] P. B. Gibbons and Y. Matias. New sampling-based summary statistics for improving approximate query answers. In *SIGMOD*, 1998.

[21] A. Halevy, A. Rajaraman, and J. Ordille. Data integration: the teenage year. In *VLDB*, 2006.

[22] J. Han, J. Pei, G. Dong, and K. Wang. Efficient computation of iceberg cubes with complex measures. In *SIGMOD*, 2001.

[23] M. A. Hernandez and S. J. Stolfo. Real-world data is dirty: Data cleansing and the merge/purge problem. *Data Mining and Knowledge Discovery*, 2(1):9–37, 1998.

[24] T. S. Jayram, A. McGregor, S. Muthukrishnan, and E. Vee. Estimating statistical aggregates on probabilistic data streams. In *PODS*, 2007.

[25] C. Jin, W. Qian, C. Sha, J. X. Yu, and A. Zhou. Dynamically maintaining frequent items over a data stream. In *CIKM*, 2003.

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

[27] R. M. Karp, S. Shenker, and C. H. Papadimitriou. A simple algorithm for finding frequent elements in streams and bags. *ACM Trans. Database Syst.*, 28(1), 2003.

[28] L. K. Lee and H. F. Ting. A simpler and more efficient deterministic scheme for finding frequent items over sliding windows. In *PODS*, 2006.

[29] V. Ljosa and A. Singh. APLA: Indexing arbitrary probability distributions. In *ICDE*, 2007.

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

[31] G. S. Manku and R. Motwani. Approximate frequency counts over data streams. In *VLDB*, 2002.

[32] A. Metwally, D. Agrawal, and A. E. Abbadi. An integrated efficient solution for computing frequent and top-k elements in data streams. *ACM Trans. Database Syst.*, 31(3):1095–1133, 2006.

[33] J. Pei, B. Jiang, X. Lin, and Y. Yuan. Probabilistic skylines on uncertain data. In *VLDB*, 2007.

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

[35] C. Re and D. Suciu. Materialized views in probabilistic databases for information exchange and query optimization. In *VLDB*, 2007.

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

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

[38] S. Singh, C. Mayfield, S. Prabhakar, R. Shah, and S. Hambrusch. Indexing uncertain categorical data. In *ICDE*, 2007.

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

[40] Y. Tao, R. Cheng, X. Xiao, W. K. Ngai, B. Kao, and S. Prabhakar. Indexing multi-dimensional uncertain data with arbitrary probability density functions. In *VLDB*, 2005.