Efficient Aggregation Algorithms for Probabilistic Data

T.S. Jayram
IBM Almaden
jayram@almaden.ibm.com

Satyen Kale*
Princeton University
satyen@cs.princeton.edu

Erik Vee†
Yahoo! Research
erikvee@yahoo-inc.com

Abstract

We study the problem of computing aggregation operators on probabilistic data in an I/O efficient manner. Algorithms for aggregation operators such as SUM, COUNT, AVG, and MIN/MAX are crucial to applications on probabilistic databases. We give a generalization of the classical data stream model to handle probabilistic data, called *probabilistic streams*, in order to analyze the I/O-requirements of our algorithms. Whereas the algorithms for SUM and COUNT turn out to be simple, the problem is harder for both AVG and MIN/MAX. Although data stream algorithms typically use randomness, all of the algorithms we present are deterministic.

For MIN and MAX, we obtain efficient one-pass data stream algorithms for estimating each of these quantities with relative accuracy $(1+\varepsilon)$, using constant update time per element and $O(\frac{1}{\varepsilon}\lg R)$ space, where each element has a value between 1 and R. For AVG, we present a new data stream algorithm for estimating its value to a relative accuracy $(1+\epsilon)$ in $O(\log n)$ passes over the data with $O(\frac{1}{\varepsilon}\log^2 n)$ space and update time $O(\frac{1}{\varepsilon}\log n)$ per element. On the other hand, we prove a space lower bound of $\Omega(n)$ for any *exact* one-pass deterministic data stream algorithm. Complementing this result, we also present an $O(n\log^2 n)$-time exact deterministic algorithm which uses $O(n)$ space (thus removing the data-streaming restriction), improving dramatically on the previous $O(n^3)$-time algorithm. Our algorithms for AVG involve a novel technique based on generating functions and numerical integration, which may be of independent interest.

Finally, we provide an experimental analysis and show that our algorithms, coupled with additional heuristics, have excellent performance over large data sets.

*Supported by Sanjeev Arora's NSF grants MSPA-MCS 0528414, CCF 0514993, ITR 0205594. Work done while the author was a summer intern at IBM Almaden.

†Work done while the author was at IBM Almaden.

1 Introduction

Recently, there have been several paradigm shifts in data management that motivate the need for building probabilistic database systems. There are multiple scenarios in which there is uncertainty associated with the input data; such scenarios include data cleansing, information extracted from unstructured data, and inconsistent databases. Second, an increasing number of such systems are returning ranked results. This implies that there is some (query) uncertainty being resolved, and ranking is a manifestation of this resolution. Third, there is a shift in data management applications from transactional support to decision support, and more broadly, business intelligence. Decision support applications can tolerate (potentially) inexact answers, as long as there are clearly defined semantics and guarantees associated with the results.

There are various data models proposed to address the representation of probabilistic databases. Perhaps the closest in spirit to our work is the data model proposed in [FR97], which uses an algebra of atomic events to define the semantics of conjunctive queries. The complexity of evaluating conjunctive queries in this framework has been addressed by [DS04].

As a motivating example, consider a car insurance company, which collects data about repair costs for various car models. Due to various external factors, the data they collect are not completely accurate; uncertainties remain about the exact car model for a given repair entry, which are translated into likelihood estimates by a data cleansing algorithm. An example repair entry would have a cost of $200 associated with a GM with probability 0.4, a Ford with probability 0.3, a BMW with probability 0.2 and a Honda with probability 0.1. The car insurance company is interested in various aggregate statistics grouped by car model. For instance, what is the average repair cost for a Honda? What is the maximum repair cost for a Ford?

To answer such questions in the face of uncertain data, we adopt the *possible worlds* semantics. In this setting, the answer to such a query is defined to be the expected value of the aggregation operator over *deter-*

ministic databases generated by the probability distributions associated with the entries. For the example above, the database is created probabilistically by independently generating a single automobile name for each repair entry using the corresponding probability distribution and then including the repair entry if the generated name equals Honda and excluding it otherwise. The above example repair entry would be included with probability 0.1 and excluded with probability 0.9.

In this paper, we study the problem of designing I/O efficient algorithms for aggregating probabilistic data. These algorithms have direct application for the class of OLAP aggregation queries on probabilistic databases as defined in [BDJ+05]. We generalize the classic data stream model to model algorithms on probabilistic data, and define the semantics for answering aggregation queries using possible worlds. The data stream model model has been traditionally useful for designing I/O efficient algorithms where the key resource requirements are (1) internal memory (2) processing time per element and (3) number of passes. The goal is to keep the resource requirements small while producing a good estimate of the query— with high confidence if the algorithm is randomized.

There are two main challenges that we have to overcome in order to answer queries efficiently. First, we cannot explicitly enumerate all the possible worlds because this could be exponentially large in the size of the input in the worst case. A more challenging problem is the possibility that an uncertain element may not even exist in the possible world with some probability, as in our motivating example.

Our first result is the first efficient one-pass data stream algorithms for estimating MIN and MAX to within relative accuracy ε. These algorithms use space $O(\frac{1}{\varepsilon}\lg R)$ and have an update time of $(\ell \lg \ell)$ per uncertain element, where $[1, R]$ is the domain, and ℓ is the size of the maximum support, taken over each pdf in the probabilistic stream.

Next, we present a data stream algorithm for estimating the average, AVG, to within relative accuracy ε that uses $O(\lg n)$ passes and has a per-element processing time of $O(\frac{1}{\varepsilon}\lg n)$. The computation of AVG is particularly tricky because the number of elements in the possible data streams is a variable quantity. Two obvious approaches have deficiencies: (1) the ratio SUM/COUNT is a heuristic estimate to AVG, which can be fairly accurate in practice, but has no worst case guarantees, and (2) sampling the data stream a number of times, and taking the mean of all the averages, requires only a single pass but a lot of samples (and space) of the order of $\Theta(\frac{1}{\epsilon^2})$, for getting $(1+\varepsilon)$ relative accuracy. In addition, our implementations of the sampling-based algorithm worked poorly in practice.

We introduce the technique of generating functions to estimate AVG, reducing the problem to estimating a certain integral efficiently. This novel technique, which may be of independent theoretical interest, can be applied to other averaging statistics as well; for the sake of clarity we only include the application to estimating AVG. This reduction also enables us to get an *exact* algorithm which runs in $O(n \lg^2 n)$ time and uses $O(n)$ space. This improves the previous dynamic programming exact algorithm for AVG that had a running time of $O(n^3)$ [BDJ+05]. In addition, we also give a space lower bound of $\Omega(n)$ for exact computation of AVG.

Finally, we present an experimental evaluation that addresses scalability as well as result quality for data sets generated under varying sets of parameters. Our experiments show that the algorithms for AVG and MIN/MAX have overall excellent performance, and that they scale almost linearly with the amount of data.

1.1 Related Work Aggregation queries over imprecise and uncertain data have been studied extensively, including in the context of hierarchical data such as OLAP [RB92, CCT96, MSS01, CKP03, RSG05, BDJ+05]. One of the main contributions of [BDJ+05] is to provide appropriate desiderata that can be used evaluate the query semantics of any system that handles imprecise and uncertain data for hierarchical data. Motivated by the desiderata, they propose an allocation-based framework to handle both imprecise and uncertain data. This can be shown to be equivalent to what [FR97] calls an extended probabilistic database system. Probabilistic databases have been studied extensively in the past few years in the context of other applications such as inconsistent databases, data cleansing, and integration of heterogenous databases [AKG87, CP87, Mot90, GMP92, BGL96, LLRS97, DS98, ABC, ABC+03, Mot96]. We refer the reader to the tutorial by [SD05] and the accompanying materials for an overview of these applications, including how probabilistic databases can help in resolving some of the problems. These papers either study non-aggregation queries or attempt to "repair" the databases to a deterministic state. In [DS04], it has been shown that the problem of evaluating conjunctive queries in general is #P-complete. #P is the complexity class of counting problems such as counting the number of satisfying assignment to a given boolean formula. The paper shows that the data (expression) complexity of conjunctive queries is #P-complete. The paper also proves a dichotomy theorem for conjunctive queries without self-joins, showing that queries are either #P-complete, or can be eval-

uated in polynomial time. The class of aggregation queries that we consider are motivated by the probabilistic database for OLAP as defined by [BDJ+05]. Therefore we are able to obtain highly efficient algorithms for such queries.

The data stream model is a powerful mechanism for designing I/O efficient algorithms. We refer the reader to two overview papers: one by [Mut06] which contains many applications and algorithmic techniques, and another by [BBD+02] for a database perspective on the applications; there are also database implementations that have good support for stream computation e.g. Stanford Stream Data Manager [SSD].

2 Probabilistic Streams

In this section, we consider a simple extension of the stream model with a view towards designing IO-efficient algorithms for probabilistic data.

Definition 2.1. (Uncertain Domain) *Let $\mathcal{B}$ denote a discrete* base domain*, and let $\perp$ denote a special symbol that does not belong to $\mathcal{B}$. An* uncertain domain $\mathcal{U}$ *over $\mathcal{B}$ is the set of all probability distribution functions, or pdfs, over $\mathcal{B} \cup \{\perp\}$.*

Each value $\vartheta \in \mathcal{U}$ is a pdf that, intuitively, encodes our degree of belief that the "true" value of some entity equals b, for each b in the base domain $\mathcal{B}$. The special character $\perp$ allows for the possibility that the entity should not be included. As stated in the introduction, this latter situation arises naturally in the context of evaluating queries on probabilistic databases [FR97, BDJ+05].

For this paper, we let $\mathcal{B} = [1, R]$ so that each pdf ϑ can be described as $\{\langle i_1, p_1\rangle, \ldots, \langle i_\ell, p_\ell\rangle\}$. Here, $\Pr_\vartheta[i_s] = p_s$ for $s = 1 \ldots \ell$, and $\Pr_\vartheta[\perp] = 1 - \sum_s p_s$. We assume that the probabilities are specified using a fixed precision such as the standard floating point precision.

In the standard data stream algorithm, the input is defined as a sequence of elements from a base domain $\mathcal{B}$. We call such a sequence a *deterministic stream.* Extending this definition, we define a *probabilistic stream* to be a sequence of pdfs from an uncertain domain $\mathcal{U}$.

We now define the semantics for defining aggregation queries on probabilistic streams. It can be shown that they correspond to the usual semantics of aggregation on probabilistic databases.

Definition 2.1. (Possible Streams) *Given a probabilistic stream $\mathcal{P} = \vartheta_1, \vartheta_2, \ldots, \vartheta_n$, consider the probability distribution on deterministic streams defined by the following experiment.* Independently *for each $i = 1, 2, \ldots, n$ in order, we first choose an element $b \in \mathcal{B} \cup \{\perp\}$ according to the pdf ϑ_i. If $b = \perp$, we output nothing, otherwise we output b; the resulting deterministic stream is the sequence of base elements produced by the above experiment. The set of deterministic streams that can be produced this way is called the* possible streams generated by $\mathcal{P}$*. We associate a positive probability with each possible stream $\mathcal{S}$ according to the probability that $\mathcal{S}$ is generated by the above experiment. Note that this defines a legal probability distribution on the set of possible streams.*

Intuitively, the possible streams represent potential ways in which we can eliminate the uncertainty in a probabilistic stream $\mathcal{P}$. It is crucial to note that possible streams can vary in length (but can be no more than the length of $\mathcal{P}$), depending on how many base elements were chosen by the experiment, and in the extreme case, the possible stream can even be empty. Moreover, the number of possible streams will be exponential in the number of elements of $\mathcal{P}$ in the worst case.

Definition 2.2. (Aggregation Query Semantics) *Let Q be an aggregation query such as* SUM*,* COUNT*,* AVG*,* MIN *or* MAX*. The result of evaluating Q over a probabilistic stream $\mathcal{P}$ equals the expected value of computing Q over the possible streams of $\mathcal{P}$ (according the distribution on the possible streams).*

Computing the expected value is intuitively justified, since it is a natural way to summarize the answers of computing Q over an exponential number of possible streams. In [BDJ+05], this is further justified from the standpoint of satisfying desiderata for handling uncertainty on OLAP queries (which use the same aggregation queries as those considered here). As a warm-up, we have:

Proposition 2.1. ([BDJ+05]) *Both* SUM *and* COUNT *can be computed exactly by a one-pass algorithm on a probabilistic stream, using negligible memory and processing time per element.*

Proof. For a probabilistic stream $\vartheta_1, \vartheta_2, \ldots, \vartheta_n$, it is easy to see that $\mathsf{SUM} = \sum_{i=1}^{n} \mathbb{E}_{X \sim \vartheta_i}[X | X \neq \perp] \cdot \Pr[X \neq \perp]$ and $\mathsf{COUNT} = \sum_{i=1}^{n} (1 - \Pr_{\vartheta_i}[\perp])$, and so these statistics can be computed in one pass over the stream. ■

3 AVERAGE

To motivate our algorithm, we first consider a simple approximation to AVG, namely SUM/COUNT, that can be computed efficiently. It was shown in [BDJ+05] that this approximation works very well when the probabilistic stream is dominated by elements with virtually no uncertainty (technically they are represented as pdfs with support on a single base element). The following

example shows that this can be a poor estimate for even simple cases.

EXAMPLE 1. *Consider a probabilistic stream* $\{\langle 1, 1.0\rangle\}, \{\langle 10, 0.1\rangle\}$ *consisting of 2 pdfs where the first has virtually no uncertainty. The possible streams are* 1 *with probability* 0.9 *and* $1, 10$ *with probability* 0.1. *Therefore* AVG *equals* $0.9*1+0.1*(1+10)/2 = 1.45$. *However* SUM/COUNT $= (1+0.1*10)/(1+0.1) = 2/1.1 \approx 1.81$. *Thus the relative error of this estimate is approximately* 25%, *showing that this is a poor estimate.*

An algorithm for computing AVG exactly was also proposed in [BDJ+05] that has a running time of $O(n^3)$, and uses a dynamic-programming based approach. This algorithm can be adapted to our setting but it takes $O(n)$ passes and both the space and the processing time per element takes time $O(n)$ as well, making it highly impractical. Our goal in this section is to obtain algorithms that have significantly better characteristics by employing a technique based on generating functions.

For technical reasons, we will consider the smoothed version of AVG in which we add a dummy pdf with value 0 and probability 1. This ensures that the possible streams are always non-empty and avoids the complications of dealing with division by 0. Moreover, the effect on the real estimate of AVG is negligible for large amounts of data.

We anticipate that our generating function technique will have other applications, therefore, we will state this in a more general form. Suppose $W_1, W_2, \ldots, W_n$ are a set of mutually independent random variables. For every $i = 1 \ldots n$, let $W_i = (A_i, B_i)$, where A_i and B_i are non-negative discrete random variables, possibly correlated. Suppose we want to compute

$$\mathsf{RATIO} = \mathbb{E}\left[\frac{\sum_i A_i}{1+\sum_i B_i}\right].$$

Note that AVG is obtained as follows. Let $X_i \sim \vartheta_i$. Then $(A_i, B_i) = (X_i, 1)$ if $X_i \neq \perp$ and $(0, 0)$ otherwise. We state the following important theorem.

THEOREM 3.1. *Let* $W_1, W_2, \ldots, W_n$ *be mutually independent, where* $W_i = (A_i, B_i)$ *is a pair of non-negative random variables, for* $i = 1 \ldots n$. *Define the function* $h_{\mathsf{R}}(x) = \sum_i \mathbb{E}[A_i x^{B_i}] \prod_{j \neq i} \mathbb{E}[x^{B_j}]$. *Then* RATIO $= \int_0^1 h_{\mathsf{R}}(x)\, dx$.

Proof. Define the generating function $g_{\mathsf{R}}(x)$ as follows:

$$g_{\mathsf{R}}(x) = \mathbb{E}\left[\frac{\sum_i A_i}{1+\sum_i B_i} \cdot x^{1+\sum_i B_i}\right] \tag{3.1}$$

Observe that $g_{\mathsf{R}}(0) = 0$ and $g_{\mathsf{R}}(1) =$ RATIO. Therefore, it suffices to show that the derivative of $g_{\mathsf{R}}(x)$ with respect to x equals $h_{\mathsf{R}}(x)$ in order to prove the theorem. Since differentiation is a linear operator, we obtain:

$$\begin{aligned} g'_{\mathsf{R}}(x) &= \mathbb{E}\left[\frac{\sum_i A_i}{1+\sum_i B_i} \cdot (1 + \sum_i B_i) \cdot x^{\sum_i B_i}\right] \\ &= \mathbb{E}[(\sum_i A_i) \cdot x^{\sum_i B_i}] \;=\; \sum_i \mathbb{E}[A_i \cdot x^{\sum_j B_j}] \\ &= \sum_i \mathbb{E}[A_i x^{B_i} \prod_{j \neq i} x^{B_j}] \\ &\overset{(*)}{=} \sum_i \mathbb{E}[A_i x^{B_i}] \prod_{j \neq i} \mathbb{E}[x^{B_j}] \;=\; h_{\mathsf{R}}(x), \end{aligned}$$

where $(*)$ follows by the independence of W_i's. This proves the theorem. ■

The reduction to computing an integral of a function $h_{\mathsf{R}}(x)$ is extremely important. Note that evaluating $h_{\mathsf{R}}(x)$ becomes feasible because of the decoupling of the expressions involving the different pairs. The following theorem shows how this can be accomplished in an iterative manner, whose correctness can be proved using a simple induction

LEMMA 3.1. *Define functions* $f_0(x) = 1$ *and* $h_0(x) = 0$ *for all* x. *For* $s = 1, \ldots, n$, *define the functions* $h_s(x)$ *and* $f_s(x)$ *as follows:* $f_s(x) = \mathbb{E}[x^{B_s}] \cdot f_{s-1}(x)$, *and* $h_s(x) = \mathbb{E}[x^{B_s}] \cdot h_{s-1}(x) + \mathbb{E}[A_s x^{B_s}] \cdot f_{s-1}(x)$ *Then,* $h_{\mathsf{R}}(x) = h_n(x)$.

As remarked earlier, AVG is a special case of RATIO, so we obtain:

THEOREM 3.2. *For the probabilistic stream,* $\vartheta_1, \ldots, \vartheta_n$ *set* $p_i = 1 - \Pr_{\vartheta_i}[\perp]$ *and* $\alpha_i = \mathbb{E}_{X \sim \vartheta_i}[X | X \neq \perp] \cdot \Pr[X \neq \perp]$ *for all* i. *Define* $h_{\mathsf{AVG}}(x) = \sum_i \alpha_i x \cdot \prod_{j \neq i}(1 - p_j + p_j x)$. *Then,* AVG $= \int_0^1 h_{\mathsf{AVG}}(x)\, dx$. *Moreover,* $h_{\mathsf{AVG}}(x)$ *can be computed efficiently in one pass for any* $x \in [0, 1]$.

Proof. For $i = 1, \ldots, n$, let $X_i \sim \vartheta_i$. In the premise of Theorem 3.1 set $(A_i, B_i) = (X_i, 1)$ if $X_i \neq \perp$ and $(0, 0)$ otherwise. A simple algebra shows that $\mathbb{E}[x^{B_i}] = 1 - p_i + p_i x$ and $\mathbb{E}[A_i x^{B_i}] = \alpha_i x$.

Applying Theorem 3.1 yields the desired expression for AVG. Since p_i and α_i can be computed easily for the pdf ϑ_i, by Lemma 3.1, $h_{\mathsf{AVG}}(x)$ can be computed efficiently for any $x \in [0, 1]$ in a single pass using negligible space. ■

An interesting implication of Theorem 3.2 is that we do not need the full description of the pdfs ϑ_i. It can be seen that the *sufficient statistics* required to compute AVG are just the p_i's and α_i's as defined by Theorem 3.2. For the algorithms computing AVG, we assume that such a preprocessing step involving one scan has been formed, and that in fact what is presented in the input is the sequence of pairs $\langle \alpha_i, p_i \rangle$ for all i.

CalcPolys($\mathcal{P}, s, t$) // *Calculate* $h_{s,t}(x)$, $P_{s,t}(x)$.
1 **IF** $s = t$ **THEN RETURN**
$h_{s,t}(x) = \alpha_s x$ and $P_{s,t}(x) = 1 - p_s + p_s x$.
2 Let $r = \lfloor \frac{s+t}{2} \rfloor$.
3 Call CalcPolys($\mathcal{P}, s, r$) to calculate
$h_{s,r}(x)$ and $P_{s,r}(x)$.
4 Call CalcPolys($\mathcal{P}, r+1, t$) to calculate
$h_{r+1,t}(x)$ and $P_{r+1,t}(x)$.
5 **RETURN**
$h_{s,t}(x) = h_{s,r}(x) \cdot P_{r+1,t}(x) + h_{r+1,t}(x) \cdot P_{s,r}(x)$
and $P_{s,t}(x) = P_{s,r}(x) \cdot P_{r+1,t}(x)$.

Figure 1: Pseudocode to calculate $h_{s,t}$ and $P_{s,t}$.

3.1 An Exact Algorithm to Compute AVG As a demonstration of the power of the generating function technique we design an algorithm that computes AVG exactly using the integral given in Theorem 3.2.

In this section, we describe an algorithm to compute AVG exactly when we have sufficient memory. Specifically, we give an algorithm that works in time $O(n \lg^2 n)$, but that uses $O(n)$ memory. Although this is an impractical data streaming algorithm, it still represents a vast improvement over the previously best-known exact algorithm, which took time $O(n^3)$ [BDJ+05]. This improvement comes as a consequence of the generating-function view of AVG.

Our algorithm is similar in spirit to the dynamic programming algorithm of [BDJ+05] which implicitly calculates the coefficients of $h_{\mathsf{AVG}}(x)$. Once this is known, computing the value of the integral is trivial. However, by grouping the terms of the product in a better way, we get a much faster algorithm. Let $h_{s,t}(x) = \sum_{i=s}^{t} \alpha_i x \cdot \prod_{\substack{j=s \\ j \neq i}}^{t} (1 - p_j + p_j x)$ and $P_{s,t}(x) = \prod_{i=s}^{t} (1 - p_i + p_i x)$. We use a recursive algorithm to compute both polynomials simultaneously, shown in Figure 1. Note that $h_{1,n} = h_{\mathsf{AVG}}$. Since multiplication of two polynomials of degree k can be done in time $O(k \lg k)$, (e.g., see [Lip81]), the running time $T(k)$ for inputs of size k of this algorithm satisfies the recurrence $T(k) = 2T(k/2) + O(k \lg k)$ Solving for T gives the following theorem.

THEOREM 3.3. *There is an algorithm that computes the exact value of* AVG *taking time* $O(n \lg^2 n)$.

3.2 Data stream algorithm for AVG First, we show a lower bound on the space used by any one-pass data stream algorithm that computes AVG exactly.

THEOREM 3.4. *Every one-pass data stream algorithm for exactly computing* AVG *requires* $\Omega(n)$ *space.*

An interesting aspect of the proof of this theorem is that it uses the generating function characterization of AVG to argue that enough information about the coefficients of $h_{\mathsf{AVG}}(x)$ must be sent by any such algorithm. The proof is deferred to Appendix B.

Thus, we resort to efficiently evaluating the integral $\int_0^1 h_{\mathsf{AVG}}(x)\,dx$ given by Theorem 3.2 in order to obtain a good approximation for AVG. This is a standard problem in numerical analysis for which there are several methods such as the trapezoidal rule or the Simpson's rule [PTVF92]. Typically, one chooses evenly spaced points in the interval in these methods. This approach suffers from 2 drawbacks: (1) it is inefficient since the number of evaluations becomes too large and (2) it produces a poor estimate because h_{AVG} is a high-degree polynomial with large derivatives, so most of the area is concentrated close to $x = 1$.

To remedy this, we will specify the set of points in $[0, 1]$ in a careful manner in order to get the estimate to within the desired accuracy. Here is our main result:

THEOREM 3.5. *For any* $\varepsilon \leq 1$, *there is a* $O(\lg n)$*-pass probabilistic data stream algorithm that is guaranteed to approximate* AVG *within* $(1 + \varepsilon)$. *The algorithm uses memory* $O(\frac{1}{\varepsilon} \lg n(\lg n + \lg R))$ *and has update time* $O(\frac{1}{\varepsilon} \lg n)$ *per block.*

We give an overview of these arguments; since they are rather technical, the details are deferred to Appendix A.

In the first step, we state certain properties of h_{AVG} that are useful in this regard. By analyzing the derivatives of the function h_{AVG}, we can infer that h_{AVG} is a degree n polynomial with non-negative coefficients, increasing with x, and has $h_{\mathsf{AVG}}(0) = 0$. Furthermore, $\frac{1}{n+1} h_{\mathsf{AVG}}(1) \leq \mathsf{AVG} \leq h_{\mathsf{AVG}}(1)$. (cf. Lemma A.1.)

Fix an $\varepsilon < 1$, and set $t = \lceil \frac{2 \lg n}{\lg(1+\varepsilon)} \rceil = O(\frac{1}{\varepsilon} \lg n)$. Next, we show that there exist ideal points $x_0, \ldots, x_t$ such that the values $h_{\mathsf{AVG}}(x_i) = (1 + \varepsilon)^{i-t} h_{\mathsf{AVG}}(1)$ increase geometrically, for all i. Moreover, these points obtain a good upper and lower rectangle approximation of the integral. In particular, define $H = \sum_{i=1}^{t} (x_i - x_{i-1}) h_{\mathsf{AVG}}(x_{i-1})$. Then, $H \leq \int_0^1 h_{\mathsf{AVG}}(x)\,dx \leq (1 + 2\varepsilon)H$. (cf. Lemma A.2.)

In the third step, we show that estimating the ideal points to within some small additive error still allows us to obtain an accurate estimate of AVG. In particular, if $\widehat{x}_0, \ldots, \widehat{x}_t$ are such that $|\widehat{x}_j - x_j| \leq \frac{\varepsilon}{8nt}$ for all j, then replacing x_j by $\widehat{x}_j$ in the expression for H results in an estimate that is within $(1 + 6\varepsilon)$ of AVG. (cf. Lemma A.3.)

Our algorithm proceeds by finding estimates of the x_i values. It begins with poor estimates of the x_i, and then iteratively updates these estimates with better and better approximations using binary search. Although

we do not do so here, it is possible to prove that this algorithm is guaranteed to take at most $O(\lg n)$ passes. Since we have upper and lower bounds on the integral, we are able to stop as soon as the two bounds are within $(1+\varepsilon)$ of each other. In practice, this generally occurs far sooner than the theoretical guarantee. The pseudocode is given in the appendix.

In our experiments, we also considered Newton-Raphson, which is known to have faster convergence than binary search if the derivatives are bounded. Unfortunately, the derivatives of h_{AVG} are large, so we can only guarantee linear convergence. Our experimental analysis compares these two algorithms.

4 MIN/MAX

Let $\vartheta_1, \ldots, \vartheta_n$ be the input probabilistic stream, and let $X_i \sim \vartheta_i$, where we make the convention that $X_i = R$ if $\perp$ is realized. Our goal in this section is to estimate

$$\mathsf{MIN} = \mathbb{E}[\min_i X_i]$$

All of our work applies similarly to estimating MAX. We begin by stating our main result.

THEOREM 4.1. *For any $\varepsilon > 0$, the value of* MIN *[respectively,* MAX*] can be approximated within $(1+\varepsilon)$ by a one-pass deterministic algorithm (given in Appendix D) on a probabilistic stream. This algorithm uses $O(\frac{1}{\varepsilon}\lg R)$ memory and has update time $O(\ell \lg \ell)$, where ℓ is the arity of the tuples describing the pdfs in the stream. (That is, the size of the support for each pdf.)*

Our algorithm for approximating MIN uses a binning technique. Recall that each $X_i \in [1, R]$.

Define $\mathsf{Bin}_i = [(1+\varepsilon)^i, (1+\varepsilon)^{i+1})$ for $i = 0, 1, \ldots, t$, where $t = \lfloor \lg R / \lg(1+\varepsilon) \rfloor$. If we knew $\Pr[X_i = \min_j X_j]$ for each i, then we could calculate the exact value of MIN. Instead, we will calculate, for all $i = 0, 1, \ldots, t$, the probability that $\min_j X_j$ is in Bin_i. Since all items in the same bin are within a $(1+\varepsilon)$ factor of each other, this will allow us to approximate the value of MIN within a $(1+\varepsilon)$ factor.

Specifically, let $X_{\mathsf{min}} = \min_j X_j$. For $i = 0, 1, \ldots, t$, define $P_i = \Pr[X_{\mathsf{min}} \in \mathsf{Bin}_i]$. The following lemma says that discretizing the P_i's in these bins only loses a factor of $(1+\varepsilon)$.

LEMMA 4.1. *Define* $\widetilde{\mathsf{MIN}} = \sum_{i=0}^{t} P_i (1+\varepsilon)^i$. *Then* $\widetilde{\mathsf{MIN}} \le \mathsf{MIN} < (1+\varepsilon)\widetilde{\mathsf{MIN}}$.

Now we need to calculate the P_i's in an efficient way. We first describe a straightforward update rule that requires $O(\ell + t)$ time per pdf. After verifying that this indeed works, we produce an update rule that takes just $O(\ell \ln \ell)$ time per pdf.

For each pdf, ϑ_r, define for $k = 0, 1, \ldots, t$ the value $q_{r,k} = \Pr[X_r \in \mathsf{Bin}_k]$. Notice that for a fixed r, these values can easily be computed in time $O(\ell + t)$: Say each ϑ_r is represented as $\{\langle a_{r,1}, p_{r,1}\rangle, \ldots, \langle a_{r,\ell}, p_{r,\ell}\rangle\}$. Initialize each $q_{r,k}$ to 0. Then for each $j = 1, \ldots, \ell$, find the k' such that $a_{r,j} \in \mathsf{Bin}_{k'}$, and update $q_{r,k'} \leftarrow q_{r,k'} + p_{r,j}$.

We now compute the P_i's in an iterative fashion. After seeing $r \ge 1$ items in the stream, define $X_{\mathsf{min}}^{(r)} = \min_{i \le r} X_i$. Notice that $X_{\mathsf{min}}^{(n)} = X_{\mathsf{min}}$. Let

$$\begin{aligned} P_i^{(r)} &= \Pr[X_{\mathsf{min}}^{(r)} \in \mathsf{Bin}_i], \\ Q_i^{(r)} &= \Pr[X_{\mathsf{min}}^{(r)} \notin \mathsf{Bin}_1 \cup \cdots \cup \mathsf{Bin}_i]. \end{aligned}$$

The following lemma gives a recurrence relation (and thus, an iterative algorithm) to compute the P_i values.

LEMMA 4.2. *Fix any bin i. Set $P_i^{(0)} = 0$ and $Q_i^{(0)} = 1$. For $r \ge 1$,*

$$\begin{aligned} P_i^{(r)} &= (1 - q_{r,1} - \ldots - q_{r,i-1}) P_i^{(r-1)} + q_{r,i} Q_i^{(r-1)}, \\ Q_i^{(r)} &= (1 - q_{r,1} - \ldots - q_{r,i}) Q_i^{(r-1)}. \end{aligned}$$

Proof. Before seeing the stream, $X_{\mathsf{min}}^{(0)} = \infty$, so $P_i^{(0)} = 0$ and $Q_i^{(0)} = 1$. For $r \ge 1$, the event $X_{\mathsf{min}}^{(r)} \in \mathsf{Bin}_i$ happens if either (i) $X_{\mathsf{min}}^{(r-1)} \in \mathsf{Bin}_i$ and $X_r \notin \mathsf{Bin}_1 \cup \cdots \cup \mathsf{Bin}_{i-1}$ or (ii) $X_{\mathsf{min}}^{(r-1)} \notin \mathsf{Bin}_1 \cup \cdots \cup \mathsf{Bin}_i$ and $X_r \in \mathsf{Bin}_i$. Secondly, the event $X_{\mathsf{min}}^{(r)} \notin \mathsf{Bin}_1 \cup \cdots \cup \mathsf{Bin}_i$ happens if $X_{\mathsf{min}}^{(r-1)} \notin \mathsf{Bin}_1 \cup \cdots \cup \mathsf{Bin}_i$ and $X_r \notin \mathsf{Bin}_1 \cup \cdots \cup \mathsf{Bin}_i$. The recurrence relations follow. ■

Implementing the above update algorithm will give a correct answer. However, it requires updating every P_i as each new pdf comes in. Through some algebraic manipulation, we can reduce the number of updates to just $O(\ell)$ per pdf, taking a total of $O(\ell \lg \ell)$ time.

First, notice that all but at most ℓ of the $q_{r,k}$ are equal to 0. Let $\mathsf{BinsOf}(\vartheta_r)$ be the set of indices k for which $a_{r,j} \in \mathsf{Bin}_k$ for some j. Notice that $q_{r,k} = 0$ for all $k \notin \mathsf{BinsOf}(\vartheta_r)$. Computing the set $\mathsf{BinsOf}(\vartheta_r)$ is easy to do in $O(\ell \lg \ell)$ time: By sorting the values $a_{r,1}, \ldots, a_{r,\ell}$ appearing in ϑ_r, grouping them into bins becomes simple since all values in the same bin appear consecutively in the sorted list. It is also not hard to compute the value of $q_{r,k}$ for all $k \in \mathsf{BinsOf}(\vartheta_r)$ once we have the sorted list of values from ϑ_r. The pseudocode for this algorithm appears in Appendix D.

We will see how to utilize this in a moment. But first, we give the following lemma which allows us to calculate P_i using a different update rule.

LEMMA 4.3. *Fix any bin i. Recursively define $U_i^{(\cdot)}$ and $V_i^{(\cdot)}$ as follows: Let $U_i^{(0)} = 0$ and $V_i^{(0)} = 1$. For $r > 0$, let*

$$U_i^{(r)} \quad U_i^{(r-1)} + \frac{q_{r,i}}{1 - q_{r,1} - \ldots - q_{r,i-1}} V_i^{(r-1)},$$
$$V_i^{(r)} \;=\; \frac{1 - q_{r,1} - \ldots - q_{r,i}}{1 - q_{r,1} - \ldots - q_{r,i-1}} V_i^{(r-1)}.$$

Then for all $r \geq 1$,

$$P_i^{(r)} \;=\; U_i^{(r)} \prod_{j=0}^{i-1} V_j^{(r)} \quad \textit{and} \quad Q_i^{(r)} \;=\; \prod_{j=0}^{i} V_j^{(r)}.$$

Proof. Both the assertions follow easily by induction on r. ■

The key property of the update rule from Lemma 4.3 is that $U_k^{(r)} = U_k^{(r-1)}$ and $V_k^{(r)} = V_k^{(r-1)}$ for all $k \notin \mathsf{BinsOf}(\vartheta_r)$. So we only need to apply the update rule for $k \in \mathsf{BinsOf}(\vartheta_r)$, which changes at most ℓ values. The pseudocode is given in Appendix D.

5 Experiments

We implemented our algorithms for AVG and MIN and evaluated their performance over various probabilistically generated data sets. We omitted SUM and COUNT from our experiments since, as proved in Section 2, there are extremely fast one-pass algorithms for estimating these operators exactly, which require no more effort than simple additions and multiplications.

5.1 Experimental setup The algorithms were coded in C, and compiled with `gcc` version 3.4.4 without the optimization option on a Sun X4100 server with dual-core, 2.2GHz Opteron 275 processors (four 64-bit processing cores) with 16 GB RAM running the CentOS distribution of Linux. We note, however, that the algorithm utilized only 1 processor in its execution. To simulate an actual database system, the probabilistic data stream was read off a file on disk in blocks, and the total memory usage was capped at 1 MB [1].

We implemented the AVG algorithm using 3 different root finding algorithms: we implemented Binary Search, Newton-Raphson, and Regula Falsi (also known as the False Position method). We optimized the search binary algorithm further by two additional heuristics, which allowed us to avoid repeated function evaluations of the same point.

We used a probabilistic generative model to generate test data. The model assumes that the elements in the data stream are generated from a distribution (such as Zipf, Gaussian, uniform, etc.) over the domain $\mathcal{B}$. Associated with each element sampled from the distribution is some uncertainty, which we model as an additive Gaussian noise. Further, we assume there is a "masking" process, which we model as a randomly chosen probability p_0 that the element is omitted from the data stream (i.e., the output is $\perp$).

For the sake of brevity, we only give a few highlights of the experimental results; more detailed results can be found in the forthcoming full version of the paper.

5.2 Experimental results for AVG Of all the root-finding methods used for AVG, the best was our optimized Binary Search algorithm, which took an average of 0.47 seconds to run over a stream of size 100,000, with $\varepsilon = 0.04$. Other methods needed on the order of hundreds of seconds for the task. So in the following, we only discuss the performance of the optimized Binary Search algorithm.

We tested our algorithm under varying scenarios of changing generative distributions and masking probabilities and the algorithm was very consistent in having almost the same running time, for a fixed stream size. The running time scales linearly with stream size and with $\frac{1}{\varepsilon}$ so for a stream with 1,000,000 entries, the algorithm took around 5 seconds for $\varepsilon = 0.04$ and around 17 seconds for $\varepsilon = 0.01$.

5.3 Experimental results for MIN Our one-pass algorithm for MIN ran extremely fast in all settings, under 0.7 seconds for a stream of size $1,000,000$ for $\varepsilon = 0.01$. The running time scaled linearly with the stream size. Just like the algorithm for AVG, the algorithm for MIN had very consistent in having the same running time under different generative distributions and masking probabilities, for a fixed stream size.

6 Conclusions and open problems

In this paper, we gave deterministic algorithms for accurately estimating various aggregation algorithms on probabilistic data. Our algorithms work in the data stream framework and are efficient in terms of running time, space requirements, and number of passes over the data. Our algorithms have good theoretical guarantees, and optimized versions of our algorithms have even better experimental performance on large data sets. Experiments also show the scalability and robustness of our algorithms under different input settings.

A number of open questions remain: our algorithm for estimating the integral has a theoretical bound of $O(\log n)$ on the number of passes. Can this be improved to a constant, under the restriction of using only

[1] Increasing memory did not give any significant performance gains.

logarithmic space? The Newton-Raphson algorithm is known to converge quadratically under suitable conditions, which can potentially yield a $O(\log \log n)$ pass algorithm. This was our hope, but in the current setup only a linear convergence of the Newton-Raphson algorithm can be guaranteed.

Preliminary results indicate that our generating function technique can be extended to give even faster algorithms for estimating AVG. Although this extension does not produce a general method for integral estimation, it does provide a one-pass algorithm for calculating both AVG and the variance of a probabilistic data stream, using small memory.

It would be interesting to extend the lower bound results for exact computation of AVG to randomized computation and multiple passes, and to address the complexity of the *exact* computation of MIN and MAX.

References

[ABC] Marcelo Arenas, Leopoldo E. Bertossi, and Jan Chomicki. Consistent Query Answers in Inconsistent Databases. In *PODS 1999*.

[ABC+03] Marcelo Arenas, Leopoldo E. Bertossi, Jan Chomicki, Xin He, Vijay Raghavan, and Jeremy Spinrad. Scalar Aggregation in Inconsistent Databases. *Theor. Comput. Sci.*, 3(296):405–434, 2003.

[AKG87] Serge Abiteboul, Paris C. Kanellakis, and Gösta Grahne. On the Representation and Querying of Sets of Possible Worlds. In *SIGMOD*, 1987.

[BBD+02] Brian Babcock, Shivnath Babu, Mayur Datar, Rajeev Motwani, and Jennifer Widom. Models and issues in data stream systems. In *PODS*, pages 1–16, 2002.

[BDJ+05] Doug Burdick, Prasad M. Deshpande, T. S. Jayram, Raghu Ramakrishnan, and Shivakumar Vaithyanathan. Olap over uncertain and imprecise data. In *VLDB '05: Proceedings of the 31st international conference on Very large data bases*, pages 970–981. VLDB Endowment, 2005.

[BGL96] David A. Bell, J. W. Guan, and Suk Kyoon Lee. Generalized Union and Project Operations for Pooling Uncertain and Imprecise Information. *Data Knowl. Eng.*, 18(2):89–117, 1996.

[CCT96] Arbee L. P. Chen, Jui-Shang Chiu, and Frank Shou-Cheng Tseng. Evaluating Aggregate Operations over Imprecise Data. *IEEE TKDE*, 8(2):273–284, 1996.

[CKP03] Reynold Cheng, Dmitri V. Kalashnikov, and Sunil Prabhakar. Evaluating Probabilistic Queries over Imprecise Data. In *SIGMOD*, 2003.

[CP87] R. Cavallo and M. Pittarelli. The Theory of Probabilistic Databases. In *VLDB*, 1987.

[DS98] Debabrata Dey and Sumit Sarkar. PSQL: A Query Language for Probabilistic Relational Data. *Data Knowl. Eng.*, 28(1):107–120, 1998.

[DS04] Nilesh N. Dalvi and Dan Suciu. Efficient query evaluation on probabilistic databases. In *VLDB*, pages 864–875, 2004.

[FR97] Norbert Fuhr and Thomas Roelleke. A probabilistic relational algebra for the integration of information retrieval and database systems. *ACM Trans. Inf. Syst.*, 15(1):32–66, 1997.

[GMP92] H. Garcia-Molina and D. Porter. The Management of Probabilistic Data. *IEEE TKDE*, 4:487–501, 1992.

[Lip81] J. D. Lipson. *Elements of algebra and algebraic computing*. Addison-Wesley Publishing Company, 1981.

[LLRS97] Laks V. S. Lakshmanan, Nicola Leone, Robert Ross, and V. S. Subrahmanian. ProbView: A Flexible Probabilistic Database System. *ACM TODS*, 22(3):419–469, 1997.

[Mot90] Amihai Motro. Accommodating Imprecision in Database Systems: Issues and Solutions. *SIGMOD Record*, 19(4):69–74, 1990.

[Mot96] Amihai Motro. Sources of Uncertainty, Imprecision and Inconsistency in Information Systems. In *Uncertainty Management in Information Systems*, pages 9–34. 1996.

[MSS01] Sally I. McClean, Bryan W. Scotney, and Mary Shapcott. Aggregation of Imprecise and Uncertain Information in Databases. *IEEE TKDE*, 13(6):902–912, 2001.

[Mut06] S. Muthukrishnan. Data streams: Algorithms and applications. *Foundations and Trends in Theoretical Computer Science*, 1(2), 2006.

[PTVF92] William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. *Numerical Recipes in C: The Art of Scientific Computing*. Cambridge University Press, New York, NY, USA, 1992.

[RB92] Elke A. Rundensteiner and Lubomir Bic. Evaluating Aggregates in Possibilistic Relational Databases. *Data Knowl. Eng.*, 7(3):239–267, 1992.

[RSG05] Robert Ross, V. S. Subrahmanian, and John Grant. Aggregate Operators in Probabilistic Databases. *J. ACM*, 52(1):54–101, 2005.

[SD05] Dan Suciu and Nilesh N. Dalvi. Foundations of probabilistic answers to queries. In *SIGMOD Conference*, page 963, 2005.

[SSD] SSDM. **St**anford st**re**am dat**a** **M**anager. http://www-db.stanford.edu/stream/.

A Proofs of Lemmas used in the proof of Theorem 3.5

LEMMA A.1. *The function h_{AVG} as defined in Theorem 3.2 satisfies the following:*

1. *h_{AVG} is a degree-n polynomial, all of whose coefficients are non-negative.*
2. *h_{AVG} is an increasing function with $h_{\mathsf{AVG}}(0) = 0$.*
3. *$\frac{1}{n+1} h_{\mathsf{AVG}}(1) \leq \mathsf{AVG} \leq h_{\mathsf{AVG}}(1)$.*

Proof. Parts 1 and 2 follow from the definition of h_{AVG}. For part 3, suppose $h_{\mathsf{AVG}}(x) = \sum_{i=0}^{n} c_i x^i$. Therefore,

$$h_{\mathsf{AVG}}(1) = \sum_i c_i.$$

By Theorem 3.2,

$$\mathsf{AVG} = \int_0^1 \sum_i c_i x^i \, dx = \sum_{i=0}^{n} \frac{c_i}{i+1}$$

Comparing the expressions in the 2 equations yields the desired inequalities. ■

LEMMA A.2. *Fix an $\varepsilon < 1$, and set $t = \lceil \frac{2 \lg n}{\lg(1+\varepsilon)} \rceil = O(\frac{1}{\varepsilon} \lg n)$. Then, there exist points $x_0, \ldots, x_t \in [0,1]$ such that*

1. *for all $i = 0, 1, \ldots, t$*

$$h_{\mathsf{AVG}}(x_i) = (1+\varepsilon)^{i-t} h_{\mathsf{AVG}}(1) \tag{1.2}$$

2. *Define $H = \sum_{i=1}^{t} (x_i - x_{i-1}) h_{\mathsf{AVG}}(x_{i-1})$. Then,*

$$H \le \int_0^1 h_{\mathsf{AVG}}(x) \, dx \le (1+2\varepsilon) H \tag{1.3}$$

Proof. First, $h_{\mathsf{AVG}}(0) = 0$ and by Lemma A.1, h_{AVG} is increasing in $[0,1]$, so the equations for points $x_0, \ldots, x_t$ given in Equation 1.2 can be satisfied. This proves part 1.

For part 2, notice that $x_t = 1$, and

$$h_{\mathsf{AVG}}(x_0) = (1+\varepsilon)^{-t} h_{\mathsf{AVG}}(1) \le \frac{1}{n^2} h_{\mathsf{AVG}}(1)$$

For brevity, let A denote AVG. By Lemma A.1, part 3, we obtain

$$h_{\mathsf{AVG}}(x_0) \le \frac{1}{n^2} \cdot (n+1) A \le \frac{\varepsilon}{2} A \tag{1.4}$$

Now, we take a simple Riemann sum using these points. Since h is increasing, we see

$$\begin{aligned} H &= \sum_{i=1}^{t} (x_i - x_{i-1}) h_{\mathsf{AVG}}(x_{i-1}) \\ &\le \int_0^1 h_{\mathsf{AVG}}(x) \, dx \qquad (= A) \\ &\le x_0 h_{\mathsf{AVG}}(x_0) + \sum_{i=1}^{t} (x_i - x_{i-1}) h_{\mathsf{AVG}}(x_i) \end{aligned}$$

By Equation 1.4, $x_0 h_{\mathsf{AVG}}(x_0) \le h_{\mathsf{AVG}}(x_0) \le \frac{\varepsilon}{2} A$. Noting that $h(x_i) = (1+\varepsilon) h(x_{i-1})$ for all i in $1 \ldots t$, we see that

$$A \le \frac{\varepsilon}{2} A + (1+\varepsilon) H$$

Rearranging, we obtain

$$A \le \frac{1+\varepsilon}{1-\frac{\varepsilon}{2}} H \le (1+2\varepsilon) H$$

To summarize, AVG bounds H from above and is no more than a factor $(1+2\varepsilon)$ of H. ■

LEMMA A.3. *Let H and $x_0, \ldots, x_t$ are as in Lemma A.2. Suppose we have points $\widehat{x}_0, \ldots, \widehat{x}_t$ such that $|\widehat{x}_j - x_j| \le \frac{\varepsilon}{8nt}$. Now, define*

$$\widehat{H} = \sum_{i=1}^{t} (\widehat{x}_i - \widehat{x}_{i-1}) h_{\mathsf{AVG}}(x_{i-1}) = \sum_{i=1}^{t} (\widehat{x}_i - \widehat{x}_{i-1})(1+\varepsilon)^{i-1-t}$$

Then,

$$\widehat{H} \le \int_0^1 h_{\mathsf{AVG}}(x) \, dx \le (1+6\varepsilon) \widehat{H}$$

Proof. By the triangle inequality, we see that $|(\widehat{x}_i - \widehat{x}_{i-1}) - (x_i - x_{i-1})| \le \frac{\varepsilon}{4nt}$. Hence, we have

$$\begin{aligned} |\widehat{H} - H| &\le \sum_{i=1}^{t} \frac{\varepsilon}{4nt} h_{\mathsf{AVG}}(x_{i-1}) \le \sum_{i=1}^{t} \frac{\varepsilon}{4nt} h_{\mathsf{AVG}}(1) \\ &= \frac{\varepsilon}{4n} h_{\mathsf{AVG}}(1) \le \frac{\varepsilon}{2} \mathsf{AVG}, \end{aligned}$$

by Lemma A.1, part 3. Combining the above equation with Lemma A.2 yields the desired bound. ■

B Proof of Theorem 3.4

Consider the special case of computing (the smoothed) AVG on $\vartheta_1, \vartheta_2, \ldots, \vartheta_{2n}$, where ϑ_1 is a pdf having support $\{\perp, 1\}$ and for $j > 1$, ϑ_j is a pdf having support $\{\perp, 0\}$. This amounts to computing the reciprocal of 1 plus the count of items over the possible streams. By Theorem 3.2, the expression for this special case of AVG equals

$$\int_0^1 \prod_{j=1}^{2n} (1 - p_j + p_j x) \, dx, \tag{2.5}$$

where $\Pr_{\vartheta_j}[\perp] = 1 - p_j$ for all j.

Let Q denote any set of $m \ge n$ distinct values in $[0,1]$. Our lower bound is based on the following 2-player one-way communication problem: Alice gets the first set of n pdfs in the probabilistic stream as defined above such that the probabilities p_j for $1 \le j \le n$ are *distinct* values from Q. Bob gets the second set of n pdfs where for some $1 \le s \le n$, the first s pdfs equal 0 with probability 1, and the remaining $n - s$ pdfs equal $\perp$ with probability 1.

We will show that given any deterministic one-way protocol for this problem that computes AVG exactly,

Bob can completely recover Alice's input. This implies that Alice has to send distinct messages for distinct inputs. This yields a lower bound of $\log \binom{m}{n}$ on the number of bits in Alice's message, which equals $\Omega(n)$ when $m = 2n$. Via a standard simulation, this yields the same lower bound on the space required by any deterministic data stream algorithm.

Fix an input $\vartheta_1, \vartheta_2, \ldots, \vartheta_n$ for Alice and recall that $\Pr_{\vartheta_j}[\perp] = 1 - p_j$, where the p_i's are distinct values from Q. For every input of Bob as defined above where the first s pdfs are identical and the remaining $n-s$ are also identical pdfs, we substitute in Equation 2.5 to obtain

$$\mathsf{AVG} = \int_0^1 \prod_{j=1}^{n} (1 - p_j + p_j x) x^s \, dx \triangleq b_s \tag{2.6}$$

Let $A(x)$ denote the expression $\prod_{j=1}^{n}(1-p_j+p_j x)$ within the integral. This is a polynomial of degree n such that $A(1) = 1$. Let $A(x) = \sum_{t=0}^{n} a_t x^t$. Substituting in Equation 2.6, we obtain for all $1 \le s \le n$:

$$b_s = \int_0^1 \sum_{t=0}^{n} a_t x^{s+t} \, dx = \sum_{t=0}^{n} \frac{a_t}{s+t+1}$$

Since $A(1) = 1$ we also obtain $\sum_{t=0}^{n} a_t = 1$. Thus, we have $n+1$ linear constraints defined by the $(n+1)\times(n+1)$ matrix H where $H(s,t) = \frac{1}{s+t+1}$ where $0 \le s, t \le n$. This is the well-known *Hilbert matrix*, and it is a non-singular matrix that can be inverted to solve for the a's. The roots of the polynomial $A(x)$ can then be used to recover the p_i's.

C Pseudocode for estimating AVG

```
EstimateAverage(P)
Initialize y_0 = 0, y_1 = 1/2, y_2 = 1
In one pass, compute h_AVG(y_k) for k = 0, 1, 2
FOR j ← 0 to t: // Recall t = ⌈2 lg n / lg(1 + ε)⌉.
    Find the smallest k ∈ {0, 1, 2} s.t.
            h_AVG(y_k) ≥ (1 + ε)^{i−t} h_AVG(1).
    Set x̂_j ← y_k.
// For binary search, set x̂'_j ← y_{i−1}.
UNTIL convergence criterion is met DO:
// Any root finding method can be used below.
    UPDATE each of the x̂_i
// For binary search, update x̂'_i as well.
    In one pass, compute h_AVG(x̂_i) for all i
    Set Ĥ_0 = x̂_0 · h_AVG(x̂_0)
    Set Ĥ_low = Σ_{i=1}^{t} (x̂_i − x̂_{i−1}) h_AVG(x̂_{i−1}).
    Set Ĥ_high = Ĥ_0 + Σ_{i=1}^{t} (x̂_i − x̂_{i−1}) h_AVG(x̂_i).
    IF Ĥ_high ≤ (1 + ε) Ĥ_low,
    THEN RETURN Ĥ_low.
    ELSE REPEAT.
```

D Pseudocode for estimating MIN

The procedure for computing MIN needs a helper procedure, ComputeBins, whose pseudocode follows:

```
ComputeBins(ϑ_r)
Initialize BinsOf(ϑ_r) ← ∅.
SORT the values a_{r,1}, ..., a_{r,ℓ} of ϑ_r
    to produce b_1 ≤ ··· ≤ b_ℓ
    with resp. probabilities p_1, p_2, ..., p_ℓ.
Set i ← 1, and b_{ℓ+1} ← ∞.
WHILE i ≤ ℓ:
    Set k ← ⌊lg b_i / lg(1 + ε)⌋ and q_{r,k} ← 0.
    Update BinsOf(ϑ_r) ← BinsOf(ϑ_r) ∪ {k}.
    WHILE k = ⌊lg b_i / lg(1 + ε)⌋:
        Update q_{r,k} ← q_{r,k} + p_i.
        Increment i ← i + 1.
RETURN
    BinsOf(ϑ_r) and q_{r,k} for all k ∈ BinsOf(ϑ_r).
```

The pseudocode for estimating MIN follows:

```
EstimateMin(P)
WHILE P is non-empty:
    Read in the next pdf ϑ_r.
    Call ComputeBins(ϑ_r) to find
        BinsOf(ϑ_r) and q_{r,k}.
    Set w ← 1.
// Maintain w = 1 − q_{r,1} − ... − q_{r,k−1}.
    FOR all k ∈ BinsOf(ϑ_r):
    // Iterate through the k in order.
        Update U_k ← (q_{r,k}/w) V_k + U_k
        Update V_k = (1 − q_{r,k}/w) V_k
        Set w ← w − q_{r,k}.
RETURN
    MIN~ = Σ_{i=0}^{t} (1 + ε)^i U_i Π_{j=0}^{i−1} V_j.
```