Conditioning and Aggregating Uncertain Data Streams: Going Beyond Expectations

Thanh T. L. Tran, Andrew McGregor, Yanlei Diao, Liping Peng, Anna Liu†
Department of Computer Science †Department of Mathematics and Statistics
University of Massachusetts, Amherst
{ttran,mcgregor,yanlei,lppeng}@cs.umass.edu †anna@math.umass.edu

ABSTRACT

Uncertain data streams are increasingly common in real-world deployments and monitoring applications require the evaluation of complex queries on such streams. In this paper, we consider complex queries involving *conditioning* (e.g., selections and group by's) and *aggregation* operations on uncertain data streams. To characterize the uncertainty of answers to these queries, one generally has to compute the full probability distribution of each operation used in the query. Computing distributions of aggregates given conditioned tuple distributions is a hard, unsolved problem. Our work employs a new evaluation framework that includes a general data model, approximation metrics, and approximate representations. Within this framework we design fast data-stream algorithms, both deterministic and randomized, for returning approximate distributions with bounded errors as answers to those complex queries. Our experimental results demonstrate the accuracy and efficiency of our approximation techniques and offer insights into the strengths and limitations of deterministic and randomized algorithms.

1. INTRODUCTION

Uncertain data streams have arisen in a growing number of environments, such as traditional sensor networks [7], GPS systems for locationing [12], RFID networks for object tracking [20], radar networks for severe weather monitoring [13], and telescope surveys for astrophysical pattern detection [17]. As more applications are developed on such streams, there is a growing demand to support complex queries for real-time tracking and monitoring despite various kinds of data uncertainty. Consider the following two examples.

RFID Tracking and Monitoring. RFID readers deployed in a storage area return readings of the tagged objects. Techniques for RFID data cleaning and inference [20] can translate noisy raw RFID data into a location tuple stream ($time$, tag_id, $weight$, x^p), where the x location, a continuous-valued attribute, is probabilistic in nature (denoted by the letter p) due to the use of inference. (For simplicity, we omit the y and z locations in this example.) A fire monitoring application could use the RFID deployment to detect violations of a fire code: storage of flammable merchandise shall not exceed 200 pounds in each unit area. Query Q1 detects such violations on the location tuple stream: It keeps the most recent location tuple for each object in the query window and groups the tuples in the window by the unit area to which they belong (where the $AreaId()$ function retrieves the id of the area given the object location and the unit area length). For each group, it computes the total weight of objects and reports the group if the weight exceeds 200 pounds. The query is written as if the x location were precise.

```
Q1:  Select group_id, sum(S.weight)
     From Locations S [Partition By tag_id Rows 1]
     Group By AreaId(S.x, AreaLength) as group_id
     Having sum(S.weight) > 200
```

Computational Astrophysics. There have been several recent initiatives to apply relational techniques to computational astrophysics. As detailed in a recent workshop paper [17], massive astrophysical surveys will soon generate observations of 10^8 stars and galaxies at nightly data rates of 0.5TB to 20TB. The observations are inherently noisy as the objects can be too dim to be recognized in a single image. However, repeated observations (up to a thousand times) allow scientists to model the location, brightness, and color of objects using appropriate distributions, represented as (id, $time$, $(x,y)^p$, $luminosity^p$, $color^p$). Then queries can be issued to detect dynamic features, transient events, and anomalous behaviors. Query Q2 below detects regions of the sky of high luminosity from the observations in the past hour. Similar to Q1, it groups the objects into the predefined regions and for the regions with the maximum luminosity above a threshold it reports the maximum luminosity.

```
Q2:  Select group_id, max(S.luminosity)
     From Observations S [Range 1 hour]
     Group By AreaId(S.(x,y), AreaDef) as group_id
     Having max(S.luminosity) > 20
```

There are several commonalities between the above two examples. First, the uncertain attributes are continuous-valued and usually modeled by a probability density functions (pdf). Unfortunately, as noted in recent workshop papers [1, 17], such attributes have been under-addressed in the probabilistic databases and data streams literature. Second, both queries involve complex relational operations on continuous-valued uncertain attributes. In particular, group by's are a form of conditioning operations that restrict the pdf of an uncertain attribute to a region specified in the group condition. Then an aggregate is applied to the tuples in each group with conditioned distributions. The aggregate result of each group can be further filtered using the `Having` clause (another form of conditioning operation). Third, such complex operations are performed in real-time as tuples arrive. These commonalities characterize the problem we address in this paper: *to support conditioning and aggregation operations on data streams involving continuous-valued uncertain attributes*.

Challenges. The most salient challenge arises from the fact that to characterize the uncertainty of query results, one generally has to

Proceedings of the VLDB Endowment, Vol. 3, No. 1

compute the *probability distributions* of uncertain attributes in both intermediate and final query results. Take Query Q2 for example. Without knowing the distribution of the maximum luminosity in each region of the sky, it is impossible to evaluate the predicate, $max(S.luminosity) > 20$, to any quantity that characterizes the confidence of the query result. (The paper [17] made a similar argument for the need to compute distributions.) However, computing distributions of query results under conditioning and aggregation operations raises a host of issues.

First, even if the input stream contains only continuous-valued uncertain attributes, which are modeled by continuous random variables, conditioning operations can introduce uncertainty about the tuple existence, which needs to be modeled by discrete random variables. Hence, for complex queries involving conditioning and aggregation, we must compute distributions for both continuous and discrete random variables. (This aspect is detailed in Section 2.)

Second, given tuples with conditioned distributions, computing distributions of aggregates is a hard, unsolved problem. In the discrete setting, it is easy to truncate a discrete distribution and add the tuple existence probability as a special value in the distribution. However, computing the distribution of an aggregate (e.g., sum) of n discrete random variables may require enumerating an exponential number (e.g., 2^n) of possible worlds, hence intractable for large n. In the continuous setting, if data uncertainty is modeled by Gaussian Mixture Models (GMMs), our previous work [19] provides exact, closed-form solutions to aggregates. Conditioning operations, however, result in truncated GMMs with an existence probability. Extending solutions in this case remains an open problem.

Third, given a mix of conditioning operations (e.g., filters and group by's) and aggregates in a query, offering query answers with bounded errors is crucial for the utility of the processing system. State-of-the-art systems compute distributions of complex queries using Monte Carlo simulation [8, 9, 16] without bounded errors.

Fourth, to support monitoring queries on data streams, query processing needs to employ incremental computation as tuples arrive and be efficient for high-volume data streams.

Relationship to Previous Work. Previous work on computing aggregates in probabilistic data streams was restricted to considering expectations of `max` and `min` [10, 11, 4], the expectation and variance of `sum`, and some higher moments of `count` [4]. In contrast, our work aims to characterize final query answers with full distributions of these aggregates. Furthermore, just knowing a few moments of an aggregate at intermediate stages of query processing may not be enough to answer queries accurately. Take Query Q2 for example. The state-of-the-art data-stream algorithm [10] returns an estimate of the expectation of `max` . However, to evaluate the `Having` clause, the expectation, $\mu = \mathbb{E}[\text{max(S.luminosity)}]$, only allows us to conclude that $\mathbb{P}[\text{max(S.luminosity)} > 20]$ is in a wide range, $[0, \min(1, \mu/20)]$, using Markov's inequality. Even if the variance of an aggregate can be obtained, as in the case of `sum`, the probability for the `Having` clause can still take a large range of values according to the Chebyshev bound. We demonstrate the poor accuracy of using only the moments in our performance study.

In the literature of probabilistic databases, the most relevant work is estimating the probability of a predicate aggregate in the `Having` clause for uncertain data modeled by discrete random variables [5, 15]. Besides the restriction to discrete random variables, this work returns only the expectations of the uncertain attributes in query results. Regarding the `Having` clause, it evaluates the predicate aggregate to a probability using Monte Carlo simulation, whereas our work explores a wider range of algorithms, both deterministic and randomized, and demonstrates the benefits of deterministic algorithms over randomized ones in most cases.

Contributions. In this paper, we present a probabilistic data stream system that evaluates queries involving conditioning operations (filters and group by's) and aggregates. Our contributions include:

An Evaluation Framework. To handle queries described above, we propose an evaluation framework that includes three components: (*i*) Our data model characterizes uncertainties associated with both attributes in a tuple, modeled by an arbitrary mix of continuous and discrete random variables, and the tuple existence probability. (*ii*) Our approximation metrics based on the Kolmogorov-Smirnov (KS) distance offer a unified theoretical foundation for bounding errors of both deterministic and randomized approximation algorithms. (*iii*) We further employ two data types for approximate representations of probability distributions. They work well with the KS-based approximation metrics and can capture important aspects of distributions in practice as we show in our performance study.

Approximation Algorithms for Aggregates. Within our framework we develop stream-speed approximation algorithms with guaranteed error bounds. We first devise such algorithms for the aggregates, `max`, `min`, `sum`, `count`, and `avg`, given tuples with conditioned distributions. The `max`/`min` algorithm uses a splitting scheme to efficiently maintain an approximate distribution as the stream is processed, and bounds the total error regardless of the number of tuples processed. The `sum`/`count` algorithm employs repeated rounding of our approximate representations as tuples are processed and further optimizes this process using advanced statistical theory. We also offer a general randomized algorithm based on Monte Carlo simulation and bound the error for all five aggregates.

Approximate Answers to Complex Queries. We consider approximate answers to complex queries that involve a mix of conditioning and aggregation operations. We quantify the errors of intermediate and final query results by keeping track of errors associated with both the attribute distributions and the tuple existence probability. We further develop a query planning approach that given a query accuracy requirement, provisions each operator with an appropriate error bound. To the best of our knowledge, our work is the first to guarantee error bounds for such complex queries.

Our experimental results show that for the class of queries considered, our system can meet arbitrary accuracy requirements while achieving throughput of thousands of tuples per second. In addition, our deterministic algorithm for `max`/`min` always outperforms the randomized algorithm, whereas our deterministic algorithm for `sum`/`count` works better given high accuracy requirements, which are desirable in most cases. Finally, using the only expectation and variance of an aggregate yields poor accuracy even if we are only concerned with the existence probabilities of query answers.

2. DATA MODEL AND OVERVIEW

In this section, we define our data model and discuss the implications of this model on relational processing.

2.1 Data Model

Input model. An uncertain data stream is an infinite sequence of tuples that conform to the schema $\mathbf{A}^d \cup \mathbf{A}^p$. The attributes in $\mathbf{A}^d$ are deterministic attributes, like those in traditional databases. The attributes in $\mathbf{A}^p$ are continuous-valued uncertain attributes, such as the location of an object and the luminosity of a star. In each tuple, the m attributes in $\mathbf{A}^p$ are modeled by a vector of continuous random variables, $\mathbf{X}$, that have a joint pdf, $f_{\mathbf{A}^p}(\mathbf{x})$, defined on $\mathbb{R}^m$. The joint pdf may be further partitioned if attributes are independent.

Attribute distributions can be generated from real-world data in many ways, including Kalman filters to estimate object speeds from GPS data [12], particle filters to estimate object locations from RFID data [20], and density estimation from repeated measurements

[17] or samples for time series data [19]. Such distributions often follow Gaussian distributions, e.g., the luminosity of a star [17], multivariate Gaussian distributions, e.g., the x and y positions of an object [19], or Gaussian mixture models, e.g., the radial velocity of a tornado in a specific area [19].

Mixed-type model for relational processing. To support relational processing of uncertain data in our input model, we propose a richer model that characterizes the uncertainty associated with tuples in intermediate and final query results. Our model, called the *mixed-type* model, essentially states that with probability p, the tuple exists and when it exists, the deterministic attributes take their original values and the uncertain attributes follow a joint distribution.

Definition 1 *Given a tuple with m continuous uncertain attributes, denoted by $\mathbf{A}^x$, n discrete uncertain attributes, denoted by $\mathbf{A}^y$, and other deterministic attributes $\mathbf{A}^d$, its* mixed-type distribution *g is a pair (p, f): $p \in [0,1]$ is the tuple existence probability (TEP), and f is the joint density function for all uncertain attributes, defined as $f(\mathbf{x},\mathbf{y}) = f_{\mathbf{A}^x|\mathbf{A}^y}(\mathbf{x}|\mathbf{y}) \cdot \mathbb{P}[\mathbf{A}^y = \mathbf{y}]$. Further, g characterizes a random vector $(\mathbf{X},\mathbf{Y},\mathbf{Z})$ over $(\mathbb{R}^m \times \mathbb{U}^n \times \mathbf{A}^d) \cup \{\perp\}$, where*

$$\mathbb{P}[(\mathbf{X},\mathbf{Y},\mathbf{Z}) = \perp] = (1-p),$$

$$\mathbb{P}\left[\mathbf{X} \subseteq I, \mathbf{Y} = \mathbf{y}, \mathbf{Z} = \mathbf{A}^d\right] = p \cdot \int_I f(\mathbf{x},\mathbf{y}) d\mathbf{x},\ I \subseteq \mathbb{R}^m, \mathbf{y} \in \mathbb{U}^n.$$

Note that the input model is a special case of the above definition where $p = 1$ and $n = 0$.

We make several notes on the mixed-type model. First, it combines the tuple-level uncertainty (i.e., TEP) with the attribute-level uncertainty. In fact, the TEP requires *every* attribute of the tuple, when used in query processing, to be modeled by a random variable: if an attribute was deterministic before, it is now modeled by a Bernoulli variable for taking its original value with probability p and $\perp$ otherwise; for the uncertain attributes, their random variables now model the joint event that the tuple exists and the attributes follow a distribution. Second, discrete uncertain attributes can emerge as derived attributes in relational processing, e.g., as the result of aggregating a set of Bernoulli variables. Third, we have a general definition of the joint attribute distribution. In any implementation, it can be factorized based on the independence among attributes and each individual distribution can be described by a known parametric distribution like Gaussian mixture models [19] or an approximate representation as we propose in the following sections.

Our current data model does not handle correlations among tuples. Inter-tuple correlations can be handled using lineage [3] and Monte Carlo simulation [9]. This paper focuses on the simpler case where tuples are independent of each other and explores stream-speed approximation in this setting. Our work can be viewed as an optimization of the general systems mentioned above when query processing does not produce correlated intermediate results.[1]

2.2 Relational Operations under the Model

We next consider relational operations under the mixed-type model. This model is especially designed for conditioning operations that commonly arise in relational processing. Formally, we define a conditioning operation as follows:

Definition 2 *Given a tuple t with a mixed-type distribution $g = (p, f)$, let S be the support of $f(\mathbf{x},\mathbf{y})$ such that S is a subset of the domain of f, and $f(\mathbf{x},\mathbf{y}) \neq 0$ for any $(\mathbf{x},\mathbf{y})$ in S. A* conditioning operation, $\mathbf{C}$, *applies a range predicate I to one of the uncertain attributes in t. Let $\bar{t}$ denote the result tuple. Then its distribution $\bar{g} = (\bar{p}, \bar{f})$ is defined as: $\bar{f}(\mathbf{x},\mathbf{y}) = f(\mathbf{x},\mathbf{y})/q$ with the support $\bar{S} = S \cap I$, $\bar{p} = p \cdot q$, and $q = \int_{S\cap I} f(\mathbf{x},\mathbf{y}) d\mathbf{x} d\mathbf{y}$.*

[1]The class of queries we support appears to be broader than safe queries defined in [5] as we can handle operations with exponential complexities (e.g., sum), which are not safe, using fast approximation.

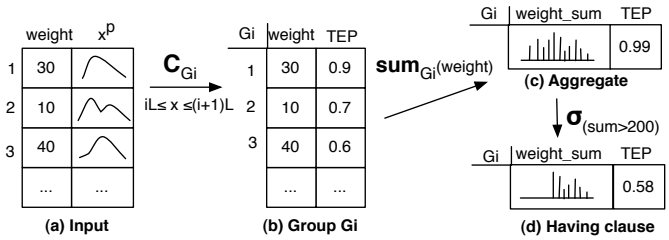


Figure 1: Execution of Q1 in the mixed-type model.

The above definition states that a conditioning operation applies a range predicate I to a tuple. It yields a truncated joint attribute distribution whose support is restricted to the intersection of the original support S and the predicate range I, but normalized. Furthermore, it reduces the tuple existence probability by the factor equal to the probability mass covered by the truncated distribution. In relational algebra, both selections and group by's are conditioning operations, as described below using our running examples.

Example: Execution of Q1 in the mixed-type model. Fig. 1(a) shows three input tuples to the query, where the *weight* is a deterministic attribute, and the x location is a continuous-valued uncertain attribute. The group by operation involves repeated conditioning operations on the input tuples, with a different condition for each group. For instance, the condition of the i-th group is $x \in [iL, (i+1)L]$, where L denotes the length of a unit area. The conditioning operation for the i-th group results in the table depicted in Fig. 1(b): The truncated distribution for the x attribute is omitted since it is not used later in the query, but the probability mass covered by the truncated distribution in each tuple becomes its existence probability (i.e., TEP) in this group. The TEP translates the aggregate, $\texttt{sum}(\textit{weight})$, into a weighted sum of Bernoulli variables. The aggregate result includes a discrete distribution of the weight sum and the TEP of this result, as shown in Fig. 1(c). Finally, the `Having` clause, modeled by a selection in relational algebra, conditions the tuple in Fig. 1(c) with the predicate $\texttt{sum}(\textit{weight}) > 200$. This will yield the reduced support of the distribution of the weight sum and reduced TEP of the aggregate result, as shown in Fig. 1(d).

Example: Execution of Q2 in the mixed-type model is similar to that of Q1, with the main difference that the aggregate, $\texttt{max}(S.\textit{luminosity})$, is the max of a set of continuous random variables. Q1 and Q2 show that aggregates and post-aggregate operations must support both continuous and discrete variables.

Other Operators. We have formally defined the semantics of other relational operators under the mixed type model. Due to space constraints, we leave such formal semantics to our technical report [18]. Regarding evaluation, our recent work [19] has shown that in the absence of conditioning operations, there are exact closed-form solutions to the result distributions of joins, projections, and aggregates. By delaying selections based on commutativity, we see that a crucial set of relational algebra where the closed-form solutions do not apply is the *aggregation of tuples with conditioned distributions*. (The interested reader can find further details in Appendix A.) Hence, supporting aggregates and post-aggregate operations given conditioned tuple distributions is a main focus of our paper.

3. DISTRIBUTIONS OF AGGREGATES

In this section, we present an approximation framework and

devise fast algorithms for aggregation of tuples in a probabilistic data stream with guaranteed error bounds.

3.1 Approximation Framework

Since aggregate functions are applied to a single attribute, the approximation framework presented below concerns a single random variable that can be discrete or continuous.

Representations. We employ cumulative distribution functions (CDF's) to approximate distributions of aggregates due to their two desirable properties: (1) they are non-decreasing functions ranging from 0 to 1, and (2) they are defined at any point in the real domain; e.g., the CDF of a discrete random variable is a step function. We use two specific CDF functions, namely StepCDF and LinCDF.

Definition 3 *Given a set of points* $P = \{(x_1, y_1), \ldots, (x_k, y_k)\}$ *where* $x_1 \leq x_2 \leq \ldots \leq x_k$ *and* $0 \leq y_1 \leq \ldots \leq y_k = 1$, $StepCDF_P$ *is the piecewise constant function that interpolates between the points whereas* $LinCDF_P$ *is a piecewise linear function that interpolates between the points:*

$$StepCDF_P(x) = \begin{cases} 0 & \text{if } x < x_1 \\ y_i & \text{if } x_i \leq x < x_{i+1} \\ 1 & \text{if } x \geq x_k \end{cases}$$

$$LinCDF_P(x) = \begin{cases} 0 & \text{if } x < x_1 \\ y_i + \frac{x - x_i}{x_{i+1} - x_i}(y_{i+1} - y_i) & \text{if } x_i \leq x < x_{i+1} \\ 1 & \text{if } x \geq x_k \end{cases}.$$

Metric. Our approximation metric is based on a standard measure in statistics, called the Kolmogorov-Smirnov distance, for quantifying the distance between two distributions over the real domain.

Definition 4 *The Kolmogorov-Smirnov (*KS*) distance between two one-dimensional cumulative distribution functions* $F, \tilde{F} : \mathbb{R} \to [0,1]$ *is defined as* $\mathrm{KS}(F, \tilde{F}) = \sup_x |F(x) - \tilde{F}(x)|$. *We say that a (randomized) algorithm returns an* (ϵ, δ) *approximation if the* KS *distance between the approximate distribution and its corresponding exact distribution is at most* ϵ *with probability* $1 - \delta$.

This metric offers two key benefits: First, it offers a unified theoretical foundation for us to bound the errors of both deterministic and Monte-Carlo algorithms as will be shown next (while the state-of-the-art Monte-Carlo approach [8, 9, 16] currently lacks guaranteed bounded errors). Second, this metric is particularly suitable for answering questions like "what is the probability that an uncertain attribute is in the range I", which commonly arises with selection operations. We return to this aspect in Section 4.

Objectives. Given the above approximate representations and metric, we devise algorithms that construct approximate distributions of aggregates over probabilistic data. We consider processing a series of tuples and define Y_t to be a random variable that characterizes the aggregate attribute in the t-th tuple. Our goal is to approximate the distribution of the random variable $A_t = A(Y_1, \ldots, Y_t)$, where A is a real-valued aggregate. If F_t^A is the cumulative distribution of A_t, we seek an algorithm that maintains an approximation $\tilde{F}_t^A$ incrementally as data arrives while satisfying a given error bound.

3.2 Bounded-Error Monte-Carlo Simulation

Our randomized algorithm is based on Monte-Carlo simulation. In contrast to prior work, we establish accuracy guarantees in our evaluation framework. We consider any aggregate A for which there exists an efficient stream algorithm Φ for computing $A(y_1, \ldots, y_t)$ given the deterministic stream $\langle y_1, \ldots, y_t \rangle$. The algorithm to compute an (ϵ, δ) approximate distribution Φ^* proceeds as follows:

- On seeing the t-th tuple, generate $m \geq \ln(2\delta^{-1})/(2\epsilon^2)$ values $y_t^1, \ldots, y_t^m$ independently from the distribution of Y_t.
- Run m copies of Φ: run the i-th copy on the stream $\langle y_1^i, \ldots, y_t^i \rangle$ and compute $a_i = A(y_1^i, \ldots, y_t^i)$, $1 \leq i \leq m$.
- Return $\tilde{F}_t^A(x) = \frac{1}{m} \sum_{i \in [m]} 1_{[a_i, \infty)}(x)$.

Theorem 3.1 *For any aggregate* A *for which there exists an exact algorithm* Φ *for computing aggregate* A *on a non-probabilistic stream, the proposed randomized algorithm* Φ^* *computes an* (ϵ, δ) *approximation of the distribution of* A *on a probabilistic stream. The space and update time used by* Φ^* *is a factor* $O(\epsilon^{-2} \log \delta^{-1})$ *greater than the space and update time required by* Φ.

The proof of the theorem is shown in Appendix B.3. We see that this theorem directly applies to aggregates such as `sum`, `count`, `avg`, `min`, and `max`. This theorem subsumes existing work based on Monte Carlo sampling [8, 9, 16] since it can determine the number of samples sufficient for meeting an accuracy requirement, in contrast to taking the number of samples as an input parameter to the algorithm. The Monte Carlo simulation in [15] only estimates the probability of an aggregate predicate in the `Having` clause, but does not compute the full distribution of an aggregate.

3.3 Distributions of MAX and MIN

In this section, we present a deterministic algorithm to compute approximate distributions of `max` and `min`. Since the algorithm is similar for both aggregates, our discussion below focuses on `max`.

We define the random variable $M_t = \max(Y_1, \ldots, Y_t)$ where Y_t is the random variable corresponding to the t-th tuple, and let F_t^M be the corresponding CDF. To provide a uniform solution for both discrete and continuous random variables, we first consider inputs modeled by discrete distributions and later extend to the continuous case. We assume that each Y_t takes λ values from a finite universe of size $\mathbb{U}$, without loss of generality, $[1, n]$, or $[n]$ for short.

A useful property of `max` is that $F_t^M(x)$ can be easily computed for any specific value of x, if x is known ahead of time, because $F_t^M(x) = \prod_{i \in [t]} \mathbb{P}[Y_i \leq x]$. Consequently, it suffices for the algorithm to maintain a value c_x, initially 1, for each x in the universe, and on processing the t-th tuple we update c_x with $c_x \cdot \mathbb{P}[Y_t \leq x]$. This computes the exact distribution of `max` with the update cost per tuple $O(\mathbb{U})$, which is inefficient for stream processing. Probabilistic databases compute the distribution of `max` based on the *extensional semantics* [5], with the total cost of $O(t\mathbb{U})$ for a relation of t tuples; further, this is not an incremental algorithm.

A natural attempt to turn the above observation into an algorithm that returns a good approximation $\tilde{F}_t^M$ for F_t^M would be to evaluate $F_t^M(x)$ for a fixed set of values of $x_0, x_1, \ldots, x_k$ and then define $\tilde{F}_t^M$ to be the k piecewise linear function that interpolates between these values. Unfortunately, this approach does not work because it is impossible to choose appropriate values of $x_0, x_1, \ldots, x_k$ without first processing the stream. For example, if we space the values evenly, i.e., $x_i = i \cdot n/k$, and observe that every Y_j takes values in the range $[2, n/k]$, then our algorithm determines that $F_t^M(x_0) = 0$ and $F_t^M(x_1) = \ldots = F_t^M(x_k) = 1$. Consequently, the interpolation $\tilde{F}_t^M$ does not satisfy the necessary approximation guarantees.

The main idea of our algorithm is to dynamically partition the universe into consecutive intervals. For each interval, we maintain the estimates of the cumulative probabilities of its two ends. Because the CDF is non-decreasing, if the cumulative probability estimates of the two ends are sufficiently close, either of these estimates is a good estimate for all the intermediate points.

Approximate Representation with Invariants. We employ an approximate representation based on StepCDF for $\tilde{F}_t^M$. The universe

is partitioned into consecutive intervals: $[1,n] = \cup_i [a_i, b_i]$, where $a_{i+1} = b_i + 1$. For each interval $[a,b]$, we maintain c_a and c_b to be the *estimates of cumulative probabilities* at a and b. Each interval $[a,b]$ is then viewed as a *broad step*, which contains a straight line from a to $b-1$ and possibly a jump at b if $c_b \neq c_a$, as illustrated in intervals I_1 and I_3 in Fig. 2(a). This yields a StepCDF defined over the point set $\{a_1, b_1, a_2, b_2, \ldots\}$.

The algorithm has the following invariants. At any point, given any interval $[a_i, b_i]$ and a constant parameter ϵ' (see Theorem 3.2 on how to set ϵ' as a function of the accuracy requirement ϵ) , we have:

$$(1)\ \ c_{b_i} \leq c_{a_i}(1+\epsilon'), \qquad (2)\ \ c_{a_{i+1}} \geq c_{a_i}\sqrt{1+\epsilon'}$$

Invariant 1 guarantees that the estimates of the two ends of an interval are close, so the estimate errors for the points in between can be bounded. Invariant 2 ensures that the estimates of any two adjacent intervals are separated by at least a certain factor. Given the range $[0,1]$ of CDF's, the number of intervals to be maintained is hence bounded, which in turn gives an upper bound on the time and space required for the algorithm.

MAX Algorithm. This algorithm computes the approximate distribution of `max` incrementally. The algorithm first initializes $\tilde{F}_t^M(x)$ with one interval, $\mathcal{I} = \{[1..n]\}, c_1 = c_n = 1$. When a new tuple arrives, the algorithm proceeds by updating the intervals in $\mathcal{I}$, subpartitioning and adjusting some intervals when necessary. When an approximation is required, a StepCDF based on the intervals and estimates is returned. Below are the main steps performed per tuple. The pseudocode is available in Appendix B.4.

0. Preprocessing: Construct a CDF from λ values in the tuple Y_t.

1. Updating and Pruning: For each interval $I = [a,b]$ in the current max distribution, update its estimates with the new tuple: $c'_a = c_a \cdot \mathbb{P}[Y_t \leq a]$ and $c'_b = c_b \cdot \mathbb{P}[Y_t \leq b]$ (see Fig 2b & c). If after updating, $c'_b < \epsilon$, discard the interval. Note that after updating, the ratio between the estimates of the two ends can only increase.

2. Subpartitioning: This step is performed to ensure that Invariant 1 is satisfied. If updating with the new tuple results in $c'_b > c'_a(1+\epsilon')$ for some interval $I = [a,b]$, we subpartition that interval into subintervals $I_1 = [a_1, b_1], \ldots, I_k = [a_k, b_k]$ with $a_1 = a$, $a_{i+1} = b_i + 1$, so that Invariant 1 holds (see Fig 2d). The implementation ensures that the interval is not partitioned excessively. Then, for each $x \in \{a_1, b_1, a_2, b_2, \ldots, b_k\}$, we update c_x as $c_x \mathbb{P}[Y_t \leq x]$.

3. Adjusting: This step deals with a subtle issue regarding the efficiency of the algorithm. If, among the intervals after subpartitioning, there exists an interval I_i, whose width is greater than half of the width of the original interval I, we split it into two intervals I_{i1}, I_{i2} with equal width. This step ensures that each new interval is at most half the width of I. However, this results in I_{i1} and I_{i2} having the same estimates; to ensure Invariant 2, one of the interval is shifted by a factor $\sqrt{1+\epsilon'}$. Fig. 2e illustrates this step.

Analysis. We define two properties for any interval: The *generation* g of an interval is the number of splits made to generate that interval. Note that the algorithm starts with one interval having $g = 0$. The *net shifting effect* s of an interval is the net number of times the interval has been shifted. s is incremented by 1 when the interval is shifted up, and decremented by 1 when shifted down. The proofs of the following lemmas and theorem are deferred to Appendix B.4.

Lemma 3.1 *For any interval $I = [a,b]$ of generation g and net shifting effect s, after t tuples have been processed, for $v \in \{a,b\}$,*

$$F_t^M(v) \in [c_v/(\sqrt{1+\epsilon'})^s, c_v/(\sqrt{1+\epsilon'})^s \cdot (1+\epsilon')^g] .$$

Furthermore, for any $x \in [a,b]$,

$$F_t^M(x) \in [c_a/(\sqrt{1+\epsilon'})^s, c_b/(\sqrt{1+\epsilon'})^s \cdot (1+\epsilon')^g] .$$

Lemma 3.2 *At any step in the algorithm, the number of intervals is bounded as follows: $|\mathcal{I}| \leq 2\log(\epsilon^{-1})/\log(1+\epsilon')$.*

Lemma 3.3 *The maximum generation of an interval is $\log \mathbb{U}$.*

Theorem 3.2 *The algorithm for `max` maintains an $(\epsilon, 0)$ approximation for F_t^M where $\epsilon' = \epsilon(1+0.5\epsilon e^{\epsilon})^{-1}(\log \mathbb{U}+1)^{-1}$. The space use is $O(\epsilon^{-1}\log \mathbb{U} \ln \epsilon^{-1})$ and the update time per-tuple is $O(\min(\lambda t, \epsilon^{-1}\log \mathbb{U} \ln \epsilon^{-1}) + \lambda)$.*

Supporting Continuous Distributions. When input tuples are modeled by continuous random variables, e.g., Gaussian distributions for object locations, a general approach is to consider a real universe of size 2^{64}. The complexity is then proportional to $\log \mathbb{U} = 64$. In most applications, the universe size depends on the range and precision of measurements, often with smaller values of $\mathbb{U}$ and the number of values per tuple λ further less than $\mathbb{U}$. This combined effect can yield a fast algorithm (as shown in Section 5.1).

3.4 Distributions of SUM and COUNT

In this section, we consider the aggregates `sum` and `count`. Since `count` is a special case of `sum`, we focus on `sum` in the discussion below. We define the random variable $S_t = \sum_{i\in[t]} Y_i$ and let F_t^S be the corresponding CDF, where Y_i is the random variable corresponding to the i-th tuple. If the mean and variance of each Y_i are bounded, then the Central Limit Theorem (CLT) states that the distribution of S_t tends towards a Gaussian distribution as t goes to infinity. Later, we quantify the rate at which the distribution converges and use this to achieve an algorithmic result when there are a sufficiently large number of tuples. But for many applications, this asymptotic result cannot be applied. In the probabilistic databases where input tuples are modeled by discrete distributions, the exact distribution of `sum` can be computed using possible worlds semantics, which has an exponential complexity in the number of tuples [5]. We instead present a deterministic algorithm that efficiently computes the approximate distribution of `sum`.

Approximate Representation using Quantiles. We use StepCDF and LinCDF with the set of points based on the quantiles of a distribution. For some $0 < \epsilon < 1$, a particularly useful set of $k = \lceil 1/\epsilon \rceil$ points are those corresponding to *uniform quantiles*, or shortly *quantiles*, of the distribution, denoted by $Q(\epsilon)$, such that:

$$P_{Q(\epsilon)}(F) = \{(x_1, \epsilon), (x_2, 2\epsilon), \ldots (x_k, 1)\} .$$

where each $x_i = F^{-1}(i\epsilon)$. It is easy to show that

$$\mathrm{KS}(F, \mathrm{LinCDF}_{P_{Q(\epsilon)}(F)}) \leq \epsilon \ , \ \mathrm{KS}(F, \mathrm{StepCDF}_{P_{Q(\epsilon)}(F)}) \leq \epsilon .$$

SUM Algorithm. We now present a deterministic algorithm for maintaining a good approximation of F_t^S. We assume that each Y_t takes values from a finite set V_t of size at most λ, where the universe size is still $\mathbb{U}$. We treat the non-existence value $\perp$ as if 0 since this does not affect the value of `sum`. In this case, it is easy to see that F_t^S satisfies $F_t^S(x) = \sum_{v\in V_t} F_{t-1}^S(x-v)\mathbb{P}[Y_t = v]$. Unfortunately even when $\lambda = 2$, the complexity of exactly representing F_t^S is exponential in t. Hence, to achieve space and time efficiency, we use approximate representations using quantiles as introduced above. The challenge is to quickly update the point set when each tuple arrives. We focus on the LinCDF representation with quantiles but the following algorithm also applies to StepCDF. (We observed empirically that LinCDF typically performed better.)

Our algorithm processes each new tuple in two conceptual steps *Update* and *Simplify*. In update, we combine our approximation for

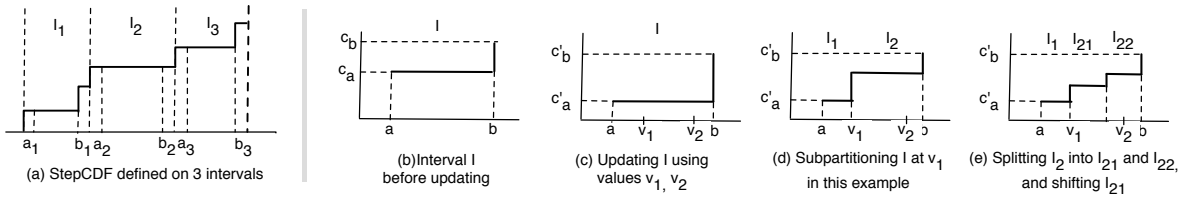


Figure 2: StepCDF and illustration of the basic steps of the MAX algorithm

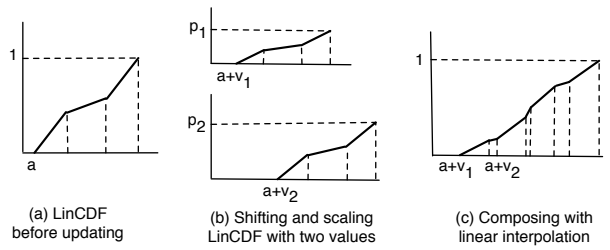


Figure 3: Updating step of the SUM algorithm

F^S_{t-1} with Y_t to produce an intermediate approximation F for F^S_t:

$$F(x) = \sum_{v \in V_t} \text{LinCDF}_{P_{t-1}}(x - v)\mathbb{P}[Y_t = v] \quad (1)$$

In this step, for each $v \in V_t$, we shift the point set P_{t-1} for the previous sum distribution by v and scale it by $\mathbb{P}[Y_t = v]$. We then compose these new point sets into λk points, in particular, using linear interpolation for the LinCDF representation. See Fig. 3 for an illustration of this step. Now F contains a set of λk points, which we call $\bar{P}_t$. Next, we simplify F to ensure efficiency in later processing while meeting the error bound ϵ' provisioned for this tuple (Theorem 3.3 shows how to set ϵ' by default, which is further optimized in our implementation.) To do this, we compute the k quantiles of F and return LinCDF_{P_t} where $P_t = \{(F^{-1}(i\epsilon'), i\epsilon') : 1 \le i \le k\}$.

However, it is inefficient to perform these steps sequentially: why compute the set of λk points for F when ultimately we are only concerned with k points? To avoid this we compute $F^{-1}(i\epsilon')$ for each i by doing a binary search for the closest pair $x_a, x_b \in \bar{P}_t$ such that $F(x_a) \le i\epsilon' \le F(x_b)$. This results in the following theorem.

Theorem 3.3 *We can maintain an $(\epsilon, 0)$ approximation for F^S_t using $O(\frac{1}{\epsilon'})$ space and $O(\frac{\lambda}{\epsilon'}\log(\frac{\lambda}{\epsilon'}))$ time per tuple, where $\epsilon' = \epsilon/t$.*

Optimizations. We further develop three optimizations of the basic algorithm: 1) *Adaptive number of quantiles.* We observe empirically that the number of quantiles, k, needed at each step to satisfy the error bound, ϵ', is smaller than the proven bound, $1/\epsilon'$. Hence, we consider a variant of the algorithm that computes the updated set of λk points, then computes the k quantiles, and then reduces the number of quantiles, e.g., by half, if the error bound ϵ' is still met. 2) *Biased quantiles.* For distributions that are close to Gaussian, we observe that using a set of *biased quantiles* gives a better approximation. However, to meet a KS requirement, we theoretically need more biased quantiles than uniform quantiles. We propose to use both sets of quantiles in the algorithm. 3) *Central Limit Theorem.* For sufficiently large t, the distribution of F^S_t can be approximated by a Gaussian distribution. To exploit this, we just need to compute a few moments of each input distribution and check if the asymptotic result holds. Further details can be found in Appendix B.5.

Supporting Continuous Distributions. When the input distributions are continuous, we propose to discretize and represent these distributions by StepCDF or LinCDF. When discretized with λ quantiles, the KS error is $\epsilon_1 = 1/\lambda$. We show in the appendix that if the KS error incurred when adding this tuple to `sum` is ϵ_2, the total error when processing this tuple is $\epsilon_1 + \epsilon_2$.

4. APPROXIMATE QUERY ANSWERS

We next consider approximate answers to complex queries involving conditioning and aggregation operations. Our work supports a `Select-From-Where-Group by-Having` block and a single aggregate predicate in the `Having` clause if present. To quantify errors of intermediate and final query results, we extend our approximation framework to account for errors in both the attribute distributions and the tuple existence probability. We then develop a query planning approach that given a query accuracy requirement, provisions each operator with appropriate error bounds. (See Appendix C for further discussion.)

Extended Approximation Metric. We first extend the KS distance to quantify the distance between two mixed type distributions.

Definition 5 *Let G=(p, F) and $\tilde{G}$=$(\tilde{p}, \tilde{F})$ be two mixed-type distributions where F and $\tilde{F}$ are the cumulative distributions of an uncertain attribute. We define the* mixed type KS*, termed* KSM*, as:*

$$\text{KSM}(G, \tilde{G}) = \max(|p - \tilde{p}|, \sup_x |p \cdot F(x) - \tilde{p} \cdot \tilde{F}(x)|, \sup_x |p \cdot (1 - F(x)) - \tilde{p} \cdot (1 - \tilde{F}(x))|).$$

As a special case, if $p = \tilde{p} = 1$, $\text{KSM}(G, \tilde{G}) = \text{KS}(F, \tilde{F})$.

For example, given a random variable X with its G and $\tilde{G}$ distributions, $\text{KSM}(G, \tilde{G}) = \epsilon$ means that all quantities such as $\mathbb{P}[x \ne \perp]$, $\mathbb{P}[x \ne \perp \wedge x \le 5]$, and $\mathbb{P}[x \ne \perp \wedge x > 5]$, when computed using G or $\tilde{G}$, will not differ by more than ϵ. The second and third components of the KSM definition ensure symmetric results for range predicates (e.g., for $<, >$) when using the KS-based distance.

To handle multiple uncertain attributes, the KSM definition can be extended to multi-dimensional CDF's [14]. In our work, since errors in query execution start to occur only at the first aggregate operator that uses approximation, the KSM has non-zero values only for the derived aggregate attributes. For derived attributes (not in the base tuples), we currently focus on their marginal distributions and bound the corresponding errors using the one-dimensional KSM. Computing joint distributions of correlated attributes derived from aggregates is a hard problem and is subject to future work.

Query Approximation Objective. We next introduce our notion of approximate answers of a query. As is known, the evaluation of a relational query results in an answer set; when given infinite resources or time, we could compute the exact answer set. We then define an approximate answer set against such an exact answer set.

Definition 6 *An approximate query answer set, $\tilde{S}$, is called (ϵ, δ) - approximation of the exact query answer set, S, if $\tilde{S}$ and S contain the same set of tuples*[2]*, and for any tuple in $\tilde{S}$, the* KSM *between*

[2]A tuple in $\tilde{S}$ and its corresponding tuple in S can be identified based on lineage [3], i.e., the same derivation from the same set of base tuples.

any of its uncertain attributes and the corresponding attribute in the corresponding tuple in S is at most ϵ with probability $1-\delta$.

Query Planning. The goal of query planning is to find a query plan that meets the (ϵ, δ) approximation objective for a given query. We first perform a bottom-up analysis of a query plan, focusing on how errors arise and propagate through operators. In our query plans, errors occur at the first aggregation that applies (ϵ, δ)-approximation as proposed in §3 (the existence probability of the aggregate result can still be computed exactly as shown in Appendix B.2). For post-aggregate operations, the earlier approximation error now affects the estimates of both the tuple existence probability and distributions of derived attributes. Below, we focus on selections and leave the discussion of other operations to our technical report [18]

Proposition 4.1 *Selection on an attribute with (ϵ, δ)-approximation using a range condition ($x \leq u$, $x \geq l$, or $l \leq x \leq u$) is $(2\epsilon, \delta)$-approximation. If the selection uses a union of ranges, the approximation error is the sum of error, $2\epsilon_i$, incurred for each range i.*

Given a query accuracy requirement, the above proposition allows us to provision error bounds for individual operators in a top-down fashion. Take Q1 whose query plan is shown in Fig. 1. Given a target error bound ϵ for the entire query, Proposition 4.1 implies that we should provision $\frac{\epsilon}{2}$ for the approximation of `sum` while allowing the error to double (in the worst case) in the subsequent selection.

5. PERFORMANCE EVALUATION

In this section, we evaluate our approximation algorithms for aggregates and complex queries in both efficiency and accuracy.

5.1 Approximation Algorithms for Aggregates

We first use simulated uncertain data streams with controlled properties to evaluate our algorithms for aggregates. The experimental setup is detailed in Appendix D. The parameters used in this study are: the accuracy requirement (ϵ, δ), the (tumbling) window size W, the number of values per tuple λ including the non-existence case (by default $\lambda = 3$), and the universe size $\mathbb{U}$ (by default, $\mathbb{U}$=10^6).

Evaluation of MAX. We evaluate the performance of both the deterministic algorithm for `max`, $\mathrm{D_{max}}$, where δ=0, and the generic randomized algorithm, Rand, where 1-δ=0.9, 0.95, or 0.99.

We first vary the error bound ϵ in a common range [0.01, 0.1]. W is uniformly sampled from [10, 1000]. Fig. 4(a) shows the throughput of the algorithms. The deterministic algorithm, $\mathrm{D_{max}}$, is 10 to 1000 times faster than the randomized algorithm, Rand, for all ϵ values tested. This is because $\mathrm{D_{max}}$ can use a small number of intervals to approximate the distribution (e.g., 20-50), whereas Rand uses hundreds to tens of thousands samples, hence worse performance. We also observe that $\mathrm{D_{max}}$ is more accurate than Rand (as shown in Fig. 6(a)), which we explain in Appendix D.

We next study the effect of the number of values per tuple, λ. We vary λ from 2 to 200, and set $W = 100$ and $\epsilon = 0.01$. Fig. 4(b) shows the throughput results. As expected, the cost of $\mathrm{D_{max}}$ increases with λ due to the costs of the first two steps of $\mathrm{D_{max}}$ depending on λ. However, the number of intervals in the approximate `max` distribution does not increase linearly in λ—it is bounded according to Theorem 3.2. Overall, the throughput of $\mathrm{D_{max}}$ is better than that of Rand by at least one order of magnitude.

Evaluation of SUM. We evaluate the performance of the deterministic algorithm for `sum`, $\mathrm{D_{sum}}$, using the optimizations shown in Section 3 and Appendix B, and the randomized algorithm, Rand.

We vary W from 10 to 1000 for two values of ϵ, 0.01 and 0.05. Figs. 4(c) and 4(d) show the throughput of both algorithms. For $\epsilon = 0.01$, $\mathrm{D_{sum}}$ is faster than Rand in all settings because Rand uses a number of samples increasing quadratically in $1/\epsilon$, but $\mathrm{D_{sum}}$ uses much less. The throughput of $\mathrm{D_{sum}}$ decreases with W because the additive error bound of $\mathrm{D_{sum}}$ requires provisioning error bounds to W tuples. For $\epsilon = 0.05$, $\mathrm{D_{sum}}$ is slightly slower than Rand for $W \leq 600$ due to the reduced benefit from ϵ. However, for larger values of W, CLT applies, yielding a high throughput of millions of tuples per second. If we keep increasing ϵ, CLT starts to apply earlier, e.g., when $W = 150$ for $\epsilon = 0.1$.

We then vary ϵ from 0.01 to 0.1. W is uniformly taken from $[1, 100]$, so that CLT cannot be applied. Fig. 4(e) shows the throughput (Fig. 6(b) in the appendix shows the accuracy). $\mathrm{D_{sum}}$ is faster than Rand for the high-precision range [0.01, 0.02]. This confirms that to gain high accuracy, Rand needs a very large number of samples and hence degrades the performance quickly. When we do not require high accuracy, Rand can be used for good throughput.

See Appendix D for other experiments for `sum` including the optimization with quantiles and varying number of values per tuple.

5.2 Approximate Query Answers

We now study the performance of two queries shown in Section 1. We also compare with alternative methods such as [10] using moments to evaluate the `Having` predicates. See Appendix D for details about the datasets and additional results not included below.

Q1. This query computes the `sum` of object weights per group and checks if it exceeds 200 (see Fig. 1 for the query plan). The `sum` is computed for Bernoulli variables, or $\lambda = 2$, which is common for aggregation of a deterministic attribute in the presence of TEP. Given a query accuracy requirement ϵ, the predicate `sum` > 200 requires assigning an error bound $\epsilon/2$ to the algorithm for `sum`.

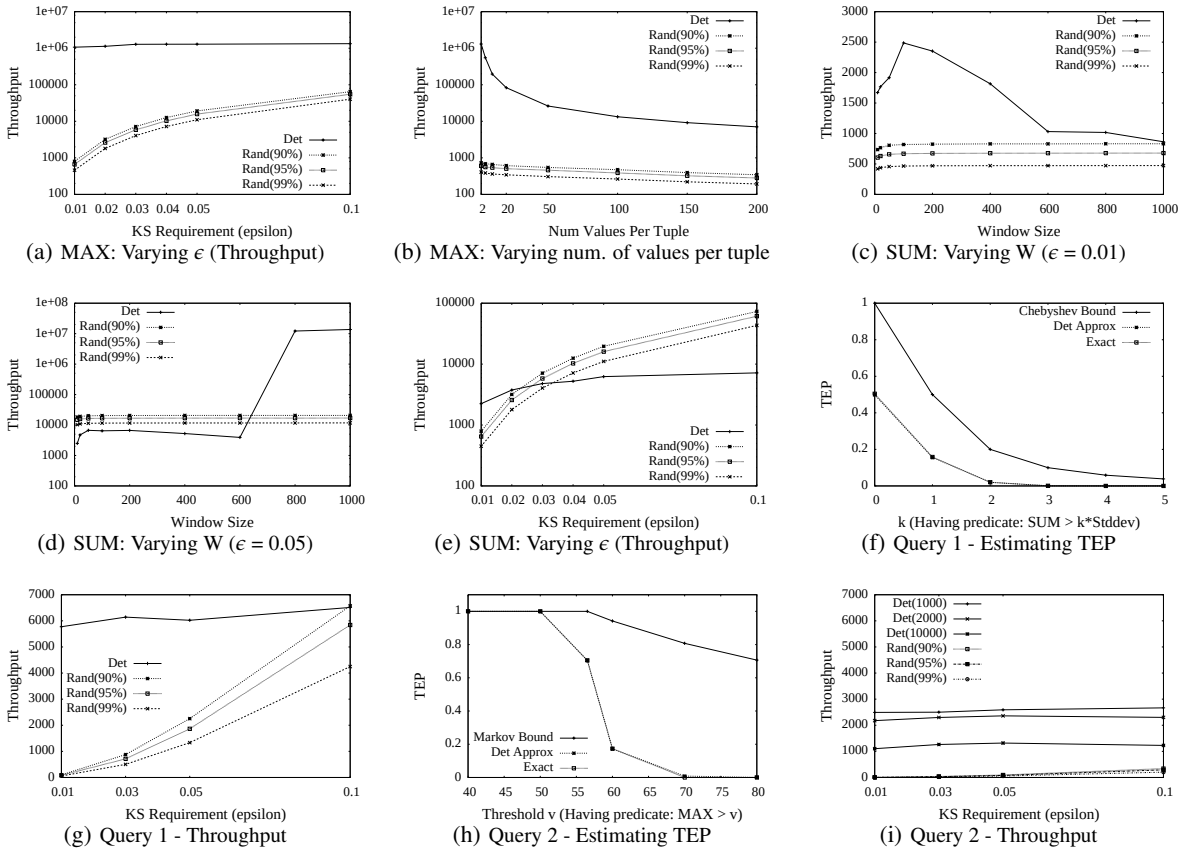
We first compare our deterministic algorithm, $\mathrm{D_{sum}}$ (with $\epsilon = 0.05$) with a method that uses only the moments of distributions to estimate the TEP given the `Having` predicate `sum` $> v$. This method cannot return the distribution of `sum` so the comparison is done for the TEP of the result tuples only. Since the mean and variance of `sum` can be computed from the input tuples using the linearity property, we use the Chebyshev's inequality to derive an upper bound of the TEP. Fig. 4(f) shows these estimates as we vary the threshold v. As seen, this method can be very inaccurate, thus confirming the need to use the `sum` distribution to compute the TEP.

We next compare the performance of $\mathrm{D_{sum}}$ and Rand on computing query result distributions. Fig. 4(g) shows the throughput. $\mathrm{D_{sum}}$ is faster than Rand due to the provisioning of smaller error bounds to the aggregate algorithm in order to account for the `Having` predicate, which causes Rand to use more samples. Also, since λ is 2 in this query, the cost of $\mathrm{D_{sum}}$ is smaller compared to Fig. 4(e).

Q2. As we are unable to obtain an astrophysical data set for this query, we use a similar query and data trace generated using the Linear Road benchmark [2]. The query considered computes the distribution of the maximum speed per road segment, and selects segments where $\mathtt{max}(speed) < 40$. The main difference from Q1 is that the aggregate attribute, *speed*, is a continuous attribute. Hence, we consider three sizes of the universe, $\mathbb{U}$ = 1000, 2000, and 10000, in the deterministic algorithm for `max`.

We again consider an alternative method that estimates the TEP of result tuples based on the moments of the `max` distribution. Since the state-of-the-art technique [10] can only compute the mean of `max`, we use the Markov's inequality to derive an upper bound of the TEP. We vary the value v in the `Having` predicate `max` $> v$ as for Q1. Fig. 4(h) shows that using this method can give inaccurate estimates, e.g., the error of the TEP can be as high as 0.6.

We now compare the performance of $\mathrm{D_{max}}$ and Rand on this query. $\mathrm{D_{max}}$ outperforms Rand under all these settings as shown in Fig. 4(i), which confirms that the former performs well for



(a) MAX: Varying ϵ (Throughput) (b) MAX: Varying num. of values per tuple (c) SUM: Varying W (ϵ = 0.01)

(d) SUM: Varying W (ϵ = 0.05) (e) SUM: Varying ϵ (Throughput) (f) Query 1 - Estimating TEP

(g) Query 1 - Throughput (h) Query 2 - Estimating TEP (i) Query 2 - Throughput

Figure 4: Experimental results of algorithms for MAX, SUM and 2 queries

large numbers of values per tuple. For the largest universe, the performance of D_{max} degrades due to the $\log \mathbb{U}$ complexity as expected. The decrease in throughput of both algorithms, compared to Fig. 4(a), is due to the group by-aggregation, as opposed to a scalar aggregate, in which an update to a group triggers the processing of all tuples in the group, e.g., 20 to 30 tuples.

6. CONCLUSIONS

In this paper, we presented an evaluation framework and approximation techniques that return distributions with bounded errors for complex queries that perform conditioning and aggregation operations on probabilistic data streams. Our work was the first in the literature to guarantee accuracy for such queries and evaluate them on data streams with demonstrated performance. In future work, we plan to support a wider range of aggregates, capture certain correlation among derived attributes, and explore query optimization to find the cheapest plan while meeting the accuracy requirement.

Acknowledgements. This work was supported in part by the National Science Foundation under the grants IIS-0746939, IIS-0812347, and CCF-0953754, and by the National Security Agency under the grant H98230-09-1-0044.

7. REFERENCES

[1] P. Agrawal and J. Widom. Continuous uncertainty in trio. In *MUD Workshop*, 2009.

[2] A. Arasu et al. CQL: A language for continuous queries over streams and relations. In *DBPL*, pages 1–19, 2003.

[3] O. Benjelloun et al. ULDBs: Databases with uncertainty and lineage. In *VLDB*, pages 953–964, 2006.

[4] G. Cormode and M. Garofalakis. Sketching probabilistic data streams. In *SIGMOD*, pages 281–292, 2007.

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

[6] A. DasGupta. *Asymptotic theory of statistics and probability*. Springer Verlag, 2008.

[7] A. Deshpande et al. Model-driven data acquisition in sensor networks. In *VLDB*, pages 588–599, 2004.

[8] T. Ge and S. B. Zdonik. Handling uncertain data in array database systems. In *ICDE*, pages 1140–1149, 2008.

[9] R. Jampani et al. MCDB: a monte carlo approach to managing uncertain data. In *SIGMOD*, pages 687–700, 2008.

[10] T. S. Jayram et al. Efficient aggregation algorithms for probabilistic data. In *SODA*, pages 346–355, 2007.

[11] T. S. Jayram et al. Estimating statistical aggregates on probabilistic data streams. *ACM Trans. Database Syst.*, 33(4), 2008.

[12] B. Kanagal and A. Deshpande. Online filtering, smoothing and probabilistic modeling of streaming data. In *ICDE*, 1160–1169, 2008.

[13] J. Kurose et al. An end-user-responsive sensor network architecture for hazardous weather detection. In *AINTEC*, pages 1–15, 2006.

[14] R. H. Lopes et al. The two-dimensional kolmogorov-smirnov test. In *Proceedings of the XI International Workshop on Advanced Computing and Analysis Techniques in Physics Research*, 2007.

[15] C. Ré and D. Suciu. The trichotomy of HAVING queries on a probabilistic database. The VLDB Journal, 18(5), 1091–1116, 2009.

[16] S. Singh et al. Database support for probabilistic attributes and tuples. In *ICDE*, pages 1053–1061, 2008.

[17] D. Suciu et al. Embracing uncertainty in large-scale computational astrophysics. In *MUD Workshop*, 2009.

[18] T. Tran et al. Conditioning and aggregating uncertain data streams: Going beyond expectations. Technical report 2010-026, UMass Amherst, 2010.

[19] T. Tran et al. PODS: A new model and processing algorithms for uncertain data streams. In *SIGMOD*, 2010.

[20] T. Tran et al. Probabilistic inference over RFID streams in mobile environments. In *ICDE*, pages 1096-1107, 2009.

APPENDIX

A. RELATIONAL PROCESSING

We provide a brief overview of relational processing, involving selections, projections, joins, aggregates, and group-by aggregation, under our mixed-type data model.

We begin with queries that involve only joins, projections, and aggregates. Our recent work [19] shows that if continuous uncertain attributes are modeled by Gaussian mixture models (GMMs), there are exact closed-form solutions to the distributions of result tuples.

When the above queries are extended with selections, as long as the selections appear after joins, projections, and aggregates in query plans, one can still apply the closed-form solutions and then compute the distribution of a selection strictly based on Definition 2. However, placing selections before joins, projections, and aggregates in a query plan can result in conditioned (more precisely, mixed type) distributions, hence not in GMMs any more. The implications of this on other relational operations depend on *commutativity*. It is known that in traditional databases, projections and joins commute with selections. These results still hold in probabilistic databases with continuous uncertain attributes. Therefore, the GMM-based solutions can still be applied if we postpone selections after the joins and projections in a query plan. However, aggregates do not commute with selections in either traditional or probabilistic databases. The lack of commutativity makes it hard to apply those GMM-based solutions to aggregates. Similarly, group-by aggregation conditions distributions before aggregation, precluding GMM-based solutions.

The above discussion leads to two conclusions: (1) *Aggregation of tuples with conditioned distributions* gives rise to an unsolved problem in relational processing under the mixed-type model. (2) It suggests the arrangement of relational operations in a query plan as follows. When commutativity applies, we compute exact distributions for operations including joins, projections, and aggregates using the the exact algorithms [19], and conditioning operations using Definition 2. Errors start to occur at aggregates following a conditioning operation, when an approximation algorithm is used, as we present in Section 3, and will propagate to the subsequent operators. In Section 4, we show how our system keeps track of such error propagation.

B. DISTRIBUTIONS OF AGGREGATES

B.1 Approximation Framework

We note that the KS distance is related to another common distance function, the variation distance (VD), which is defined as $V(f,g) = \frac{1}{2}\int_{\mathbb{R}} |f(x) - g(x)|dx$, where f and g are the probability density functions (pdf's) of two random variables.

Proposition B.1 *The following relation holds between the KS distance of two CDF's,* KS*(F, G), and the variation distance of the corresponding pdf's,* $V(f,g)$*:* KS*(F, G)* $\leq V(f,g)$*. In some cases,* KS*(F, G) can be arbitrarily smaller than* $V(f,g)$*.*

The proof of this proposition is available in [18]. Since KS(F, G) $\leq V(f,g)$ always holds, any approximation algorithm that satisfies the error bound ϵ using the VD metric can be readily included in our evaluation framework that bounds the KS distance.

B.2 Existence Probabilities of Aggregates

For all standard aggregates, the existence probability of the aggregate result, p, can be computed exactly. Specifically, for `count`, $p = 1$; for `sum`, `avg`, `max` and `min`, the aggregate result exists if one of the input tuples exists; hence, $p = 1 - \prod_t (1 - p_t)$. Therefore, in Section 3, we focus on algorithms that compute (ϵ, δ) approximate distributions when the aggregate results exist.

B.3 Monte Carlo Simulation

PROOF OF THEOREM 3.1. The result follows from the Dvoretsky-Kiefer-Wolfowitz theorem: Given m i.i.d. samples $R_1, \ldots, R_m$ from a distribution F, and the empirical distribution function they define, $\tilde{F}(x) = \frac{1}{m}\sum_{i\in[m]} 1_{[R_i,\infty)}(x)$, then $\mathbb{P}\left[\text{KS}(\tilde{F}, F) > \epsilon\right] < 2e^{-2m\epsilon^2}$. If $m \geq \ln(2\delta^{-1})/(2\epsilon^2)$, this probability is less than δ. □

B.4 Pseudocode and Proof for MAX

Pseudocode of the MAX Algorithm is shown below.

Algorithm 1 MAX: Processing a tuple

Input: Interval $I = [a, b]$, tuple Y_t, constants ϵ, ϵ'.

```
1: c'_a = c_a P[Y_t ≤ a], c'_b = c_b P[Y_t ≤ b].
2: if c'_b < ε then
3:    Discard this interval.
4: else
5:    if c'_a ≥ c'_b/(1+ε') then
6:       Update estimates: c_a ← c'_a, c_b ← c'_b.
7:    else
8:       Subpartition I: T' ← Subpartition(I, Y_t).
9:       for I' = [a', b'] ∈ T' do
10:         c_a' = c_a P[Y_t ≤ a'], c_b' = c_a P[Y_t ≤ b'].
11:         if |I'| > |I|/2 then
12:            Divide I' into two equal width intervals, I'_1 and I'_2.
13:            if I'_1 starts at a then
14:               Shift the estimates of I'_2 up by a factor √(1+ε').
15:            else
16:               Shift the estimates of I'_1 down by a factor √(1+ε').
17:            end if
18:         end if
19:      end for
20:   end if
21: end if
```

Algorithm 2 MAX: Subpartition Procedure

Input: Interval $[a, b]$, tuple Y_t, constant ϵ'.

1: $i = 1, a_i = a$.
2: $b_i = \min\{r : \mathbb{P}[Y \leq r+1] > \mathbb{P}[Y \leq a_i](1+\epsilon')\} \cup \{b\}$.
3: **if** $b_i < b$: $i = i+1$ **then** $a_i = b_{i-1} + 1$; repeat step 2.

PROOF OF LEMMA 3.1. Because a cumulative distribution is non-decreasing, for any $x < y < z$, $F_t^M(x) \leq F_t^M(y) \leq F_t^M(z)$. Consequently if for some α, β, γ, c_x/α and c_z/α are under-estimates for $F_t^M(x)$ and $F_t^M(z)$ such that

$$F_t^M(x) \geq c_x/\alpha \geq F_t^M(x)/\beta \text{ and } F_t^M(z) \geq c_z/\alpha \geq F_t^M(z)/\beta$$

and $c_x \leq c_z \leq \gamma c_x$, then $c_y = c_x$ satisfies

$$\frac{F_t^M(y)}{\beta\gamma} \leq \frac{F_t^M(z)}{\beta\gamma} \leq \frac{c_z}{\gamma\alpha} \leq \frac{c_y}{\alpha} \leq F_t^M(x) \leq F_t^M(y)$$

i.e., we implicitly have an under-estimate for $F_t^M(y)$, i.e., c_x/α, whose multiplicative error is at most $\beta\gamma$.

We proceed by induction on the generation. Clearly for $g = 0$, the result is true because c_1 and c_n are computed exactly. Consider an interval $[a, b]$ at step t, characterized by generation g and net

shifting effect s, and assume that the following inequality holds for v in $\{a,b\}$ before updating with tuple t.

$$F_t^M(v) \geq c_v/(\sqrt{1+\epsilon'})^s \geq F_t^M(v)/(1+\epsilon')^g$$

If updating with tuple t does not trigger subpartitioning, this condition still holds since both c_a and $F_t^M(a)$ are multiplied by the same factor $\mathbb{P}[Y_t \leq a]$. (Similarly for c_b and $\mathbb{P}[Y_t \leq b]$).

If updating requires subpartitioning, then $g' = g+1$. Assuming that no adjustment is needed, after updating $c_a \geq c_b/(1+\epsilon')$; hence, $\gamma = 1+\epsilon'$. Since $\beta = (1+\epsilon')^g$, according to our analysis, the multiplicative error for the estimates of the ends of a new interval is $\beta\gamma = (1+\epsilon')^{g+1} = (1+\epsilon')^{g'}$. If an adjustment is made, s is incremented or decremented so that $c_x/(\sqrt{1+\epsilon'})^s$ remains the same estimate for $F_t^M(x)$ as before adjustment; therefore the given inequality holds for new g and s. By induction, it holds for any generation. This second part of the lemma follows immediately. □

PROOF OF LEMMA 3.2. Suppose $\mathcal{I} = \{I_1, I_2, \ldots, I_m\}$ where $I_i = [a_i, b_i]$. The lemma follows because $\epsilon \leq c_{b_1} \leq c_{a_1}(1+\epsilon')$, $c_{a_m} \leq c_{b_m} \leq 1$ and for all $i \in [m-1]$, $c_{a_{i+1}} \geq c_{a_i}\sqrt{1+\epsilon'}$. □

PROOF OF LEMMA 3.3. We define the *width* of an interval $I = [a,b]$ to be $b-a+1$. Note that the generation 0 interval has width n and that every interval has width at least 1. The lemma follows from the fact that if a generation g interval I is subpartitioned into generation $g+1$ intervals $I_1, I_2, \ldots I_k$ then each $I_i, i \in [k]$, has a width of at most half of the width of I. □

PROOF OF THEOREM 3.2. From Lemma 3.3, for any interval $[a,b]$, if we have compensated for the net shifting effect by $\bar{c}_a = c_a/(\sqrt{1+\epsilon'})^s$ and $\bar{c}_b = c_b/(\sqrt{1+\epsilon'})^s$, then we have:

$$F_t^M(a) \geq \bar{c}_a \geq \frac{F_t^M(a)}{(1+\epsilon')^g} \text{ and } F_t^M(b) \geq \bar{c}_b \geq \frac{F_t^M(b)}{(1+\epsilon')^g}$$

Also, from the algorithm, we have: $\bar{c}_a \leq \bar{c}_b \leq (1+\epsilon')\bar{c}_a$. Therefore, as shown in our analysis in the proof of Lemma 3.1, the multiplicative error is $(1+\epsilon')^{g+1} \leq (1+\epsilon')^{\log \mathbb{U}+1}$. It can be shown using Taylor's theorem that $\epsilon' \leq \epsilon/((1+0.5\epsilon e^{\epsilon})(\log \mathbb{U}+1))$ suffices to ensure that the multiplicative error (and therefore the additive error since all quantities are less than 1) is less than ϵ.

The running time of the algorithm follows because there are $O(\min(\lambda t, \epsilon^{-1}\log \mathbb{U} \ln \epsilon^{-1})$ intervals and the estimate for each endpoint is updated when a tuple arrives. In addition, running the subpartitioning procedure on an interval I takes time proportional to the number of values taken by Y_t that fall in the interval. Hence, the total time over all intervals is $O(\lambda)$. □

B.5 Proof and Additional Discussion for SUM

PROOF OF THEOREM 3.3. We first consider the error accumulated by repeatedly "rounding" $F(x)$, as defined in Equation 1 in Section 3.4, to construct $\text{LinCDF}_{P_t}(x)$. We first note that for any x,

$$\begin{aligned}
&|F_t^S(x) - F(x)| \\
&= \sum_{v \in V_t} |F_{t-1}^S(x-v) - \text{LinCDF}_{P_{t-1}}(x-v)|\mathbb{P}[Y_t = v] \\
&\leq \sum_{v \in V_t} \text{KS}(\text{LinCDF}_{P_{t-1}}, F_{t-1}^S)\mathbb{P}[Y_t = v] = \text{KS}(\text{LinCDF}_{P_{t-1}}, F_{t-1}^S)
\end{aligned}$$

and hence $\text{KS}(F_t^S, F) \leq \text{KS}(\text{LinCDF}_{P_{t-1}}, F_{t-1}^S)$. Therefore,

$$\begin{aligned}
\text{KS}(\text{LinCDF}_{P_t}, F_t^S) &\leq \text{KS}(\text{LinCDF}_{P_{t-1}}, F_{t-1}^S) + \text{KS}(F, \text{LinCDF}_{P_t}) \\
&\leq \text{KS}(\text{LinCDF}_{P_{t-1}}, F_{t-1}^S) + \epsilon
\end{aligned}$$

and by induction on t, $\text{KS}(\text{LinCDF}_{P_t}, F_t^S) \leq t\epsilon$.

We next consider the running time of the algorithm. Since evaluating $F(x)$ for a given x takes $O(\lambda)$ time, performing a binary search for a quantile value over the set $\bar{P}_t$, where $|\bar{P}_t| \leq \lambda k$, takes $O(\lambda \log \lambda k)$ time. The total time is $O(\lambda k \log \lambda k)$ since we need to find x_i for all $1 \leq i \leq 1/\epsilon$. □

Biased Quantiles. In practice we also observe that a set of points based on *biased quantiles* often gives good empirical results. For some small γ, let $k^* = 1 + \lfloor \log_{1+\epsilon}(1/(2\gamma)) \rfloor$ and let

$$P_{\text{BQ}(\epsilon,\gamma)}(F) = \{(x_1,\delta_1),(x_2,\delta_2),\ldots,(x_{2k^*+2},\delta_{2k^*+2})\}$$

where $x_i = F^{-1}(\delta_i)$ and

$$\delta_i = \begin{cases} (1+\epsilon)^{i-1}\gamma & \text{if } i \in [k^*] \\ 1-(1+\epsilon)^{2k^*+1-i}\gamma & \text{if } i-k^*-1 \in [k^*] \\ 1 & \text{if } i = 2k^*+2 \end{cases}$$

Because $|\delta_i - \delta_{i+1}| \leq \epsilon$ for all $i \in [2k^*+2]$, it is easy to show

$$\text{KS}(F, \text{LinCDF}_{P_{\text{BQ}(\epsilon,\gamma)}(F)}) \leq \epsilon \ , \ \text{KS}(F, \text{StepCDF}_{P_{\text{BQ}(\epsilon,\gamma)}(F)}) \leq \epsilon \ .$$

Fig. 6 shows examples of approximating a cumulative Gaussian distribution using StepCDF and LinCDF with uniform and biased quantiles. We observe that LinCDF does a better job of approximating the true CDF. Furthermore, basing LinCDF on biased quantiles is more accurate than basing it on uniform quantiles because of the areas of higher curvature as we approach the tails of the distribution.

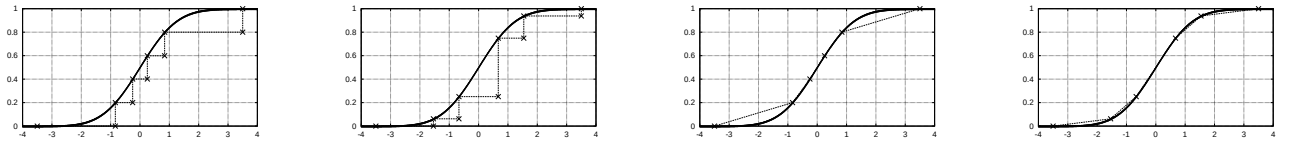
Asymptotic Result for Long Windows. We use the following theorem due to Berry and Esseen (see DasGupta [6] for an overview of the relevant area of statistics) to quantify the rate of convergence: Let $Y_1, \ldots, Y_t$ be independent random variables with finite $\mathbb{E}[Y_i] = \mu_i$, $\mathbb{V}[Y_i] = \sigma_i^2$, and $\beta_i = \mathbb{E}[|Y_i - \mu_i|]$. Let F be the CDF of $Y = \sum_{i\in[t]} Y_i$. Let $\mu_Y = \sum_{i\in[t]} \mathbb{E}[Y_i]$ and $\sigma_Y^2 = \sum_{i\in[t]} \sigma_i^2$. Then,

$$\text{KS}(F, \Phi_{\mu_Y,\sigma_Y^2}) \leq B(\beta,\sigma) := 0.8(\sum_{i\in[t]} \beta_i)(\sum_{i\in[t]} \sigma_i^2)^{-3/2}$$

where $\Phi_{a,b}$ is the cumulative distribution of the Normal distribution with mean a and standard deviation b.

For the algorithmic result, we just need to incrementally compute $\sum_{i\in[t]} \beta_i$, $\sum_{i\in[t]} \sigma_i^2$, $\mu_{S_t} = \sum_{i\in[t]} \mu_i$, and $\sigma_{S_t}^2 = \sum_{i\in[t]} \mu_i$, which is easily achieved in $O(1)$ words of space and $O(\lambda)$ time per tuple. Whenever $B(\beta,\sigma)$ falls below ϵ, we can construct an $(\epsilon,0)$ approximation for F_t^S from the values computed. However, since $B(\beta,\sigma)$ is not necessarily monotonically decreasing in t, it could be the case that it is sufficient to use the Normal approximation for F_t^S whereas the Normal approximation is not sufficiently accurate for F_{t+1}^S. In this case, we switch back to the previous algorithm by first constructing the (biased) quantiles of the Normal approximation.

Implementation Issues. *Provisioning error bounds.* Since our goal is to compute an $(\epsilon,0)$ approximate distribution of `sum`, the algorithm needs to know the number of tuples in the window. If we do not know this number in advance, we can use an infinite sequence to provision error bounds to tuples, e.g., $\epsilon\sum_{i=1}^{\infty}(\frac{1}{2})^i = \epsilon$. On the other hand, if we know an upper bound on the number of tuples based on application knowledge, we can use it to provision error bounds. Another approach is to buffer them until the window closes when we can provision an error bound for each tuple. This incurs some delay in output, but is more efficient than the above two methods. Also, it empirically shows that larger error bounds should be provisioned to the earlier tuples, e.g., 10 or 15 tuples since when more tuples are summed, the distribution becomes smoother and requires fewer quantiles to approximate.



(a) StepCDF (quantiles) (b) StepCDF (biased-quantiles) (c) LinCDF (quantiles) (d) LinCDF (biased-quantiles)

Figure 5: Approximate Representations of Cumulative Distribution using 4 intermediate points.

Execution mode. We now consider the execution mode for `sum`. We know that when the number of tuples is large enough, the condition by Berry-Esseen theorem is satisfied and hence, CLT can be applied. Therefore, we use the execution model as follows. On arrival, tuples are buffered, and the distribution moments specified in the Berry-Esseen theorem are computed. When the window for `sum` closes, check if CLT can be applied. If not, run the deterministic algorithm to compute the distribution of `sum` for the buffered tuples. Since the window size is known then, it is easy to allocate error bounds for tuples. Also, if the data arrival rate is low, when each tuple arrives, we first perform computation for CLT and then for the deterministic algorithm (sometimes partially) until the next tuple arrives (in this case, we use an upper bound of the window size to provision error bounds).

Discretizing Continuous Distributions. Let $F^S_{t-1}(x)$ be the CDF of `sum` at time t and Y_t be the new tuple represented by CDF $\tilde{F}^Y_t(v)$ with KS distance ϵ_1. If we consider $F^S_{t-1}(x)$ to be exact and every point x having probability $\mathbb{P}[s = x]$, then we update the CDF of `sum` by $F^S_t(x) = \sum_{v \in V_t} \mathbb{P}[s = x - v]\, \tilde{F}^Y_t(v)$. Similar to the proof of Theorem 3.3, the KS error of $F^S_t(x)$ is ϵ_1. If we next compute a set of quantiles for this function and incur a KS error of ϵ_2, the total error is $\epsilon_1 + \epsilon_2$ due to the additive property of KS distance.

C. APPROXIMATE QUERY ANSWERS

Our approach leverages previous research [19] by postponing selections to later, obtaining exact closed-form solutions in the early part of a query plan involving joins, projections, and aggregates. Errors only start to occur in the first aggregate operator after a selection or a group-by that uses approximation to handle conditioned tuple distributions—this is the technical context for our discussion.

Queries Supported in Planning. The class of queries that we support in this work follows the template below, which involves a single `Select-From-Where-Group By-Having` block:

```
Select group_id, Aggr(a1), ...
From S(G_a1, G_a2, ..., a1, a2...)  [window def.]
[Where BooleanExpr(G_a1, G_a2, ..., a1, a2...)]
[Group By Fn(G_a1, G_a2, ...,) as group_id]
[Having Aggr(a1) ∈ I]
```

As in relational stream processing, the query window contains a set of tuples from the input stream, each containing a number of continuous-valued uncertain attributes. The `Where` clause, if present, applies conjunctive predicates to the attributes. Then the `Group By` clause assigns tuples into groups based on the group attributes and then computes aggregates for each group. In our current work we consider two cases: (1) The `Having` clause involves a predicate aggregate that is uncertain. If the input tuples for the aggregate in `Having` all have TEP=1, then we can support various aggregates in the `Select` clause with their (marginal) distributions. If the input tuples for the predicate aggregate may have TEP<1, we can only return the distribution of this aggregate in the `Select` clause. This is because multiple aggregates computed from the same set of tuples, even if from independent attributes, are correlated or conditionally independent based on the existence of these tuples. Given the conditioning operation on one of the aggregates in the `Having` clause, simply returning the marginals of other aggregates in the `Select` clause is not correct. (2) The `Having` clause does not involve a predicate aggregate that is uncertain (but can involve other predicates on deterministic attributes), we can compute the marginal distributions of various aggregates in the `Select` clause.

Proof of Proposition 4.1 for Selections.

PROOF OF PROPOSITION 4.1. We consider a tuple t having a mixed type distribution $(\tilde{p}_t, \tilde{F}_t)$, which is an $(\epsilon, 0)$ approximation of the exact distribution (p_t, F_t). Let $\bar{t}$ denote the output tuple after applying a selection on t using a range condition. Again, the approximate distribution of $\bar{t}$ is denoted by $(\tilde{p}_{\bar{t}}, \tilde{F}_{\bar{t}})$, while the corresponding exact distribution is $(p_{\bar{t}}, F_{\bar{t}})$.

First, consider the selection condition, $x \leq u$. The KSM of the result distribution may come from the error of the new tuple existence probability (TEP) or the approximation of the CDF of the tuple attribute. The approximate TEP after selection is $\tilde{p}_{\bar{t}} = \tilde{p}_t \tilde{F}_t(u)$ while the exact TEP after selection is $p_{\bar{t}} = p_t F_t(u)$. The error in TEP incurred is $|\tilde{p}_{\bar{t}} - p_{\bar{t}}| = |\tilde{p}_t \tilde{F}_t(u) - p_t F_t(u)| \leq \epsilon$. (This inequality follows directly from the definition of KSM).

After selection,

$$\tilde{F}_{\bar{t}}(x) = \frac{\tilde{F}_t(x)}{\tilde{F}_t(u)} \text{ and } F_{\bar{t}}(x) = \frac{F_t(x)}{F_t(u)}, \; x \leq u$$

The first error component from the approximate CDF is:

$$|\tilde{p}_{\bar{t}} \tilde{F}_{\bar{t}}(x) - p_{\bar{t}} F_{\bar{t}}(x)| = |\tilde{p}_t \tilde{F}_t(x) - p_t F_t(x)| \leq \epsilon$$

The second error component from the approximate CDF is:

$$\begin{aligned} & |\tilde{p}_{\bar{t}}(1 - \tilde{F}_{\bar{t}}(x)) - p_{\bar{t}}(1 - F_{\bar{t}}(x))| \\ = \; & |\tilde{p}_t(\tilde{F}_t(u) - \tilde{F}_t(x)) - p_t(F_t(u) - F_t(x))| \leq 2\epsilon \end{aligned}$$

Combining all error components gives $(2\epsilon, 0)$-approximation for selection with condition, $x \leq u$. The proof for the range $x \geq l$ or $l \leq x \leq u$ can be shown similarly.

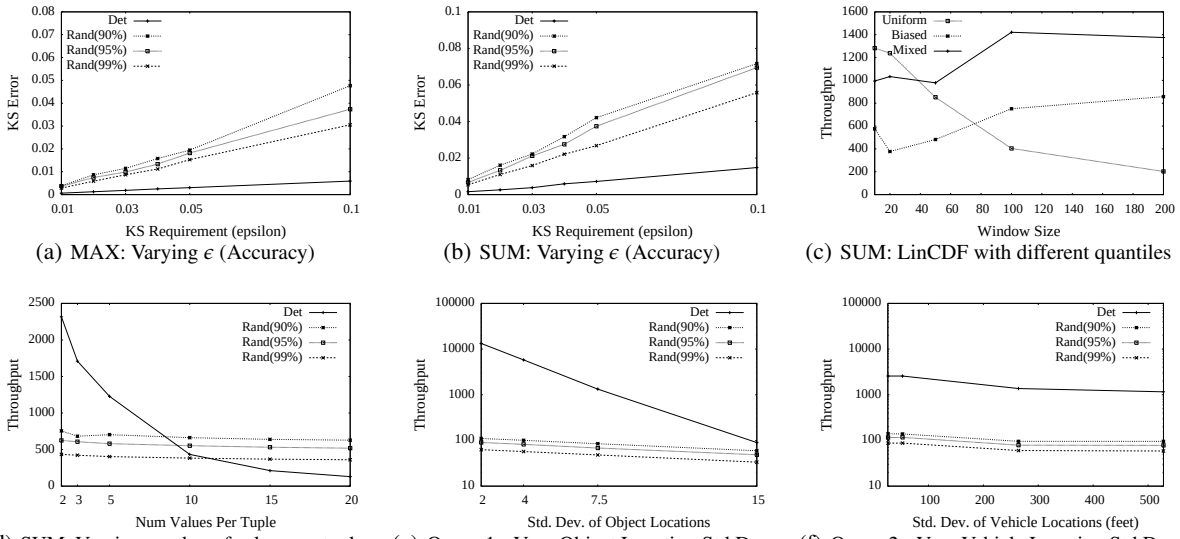
For (ϵ, δ) approximation where $\delta > 0$, we can ensure that selection gives an approximation of $(2\epsilon, \delta)$ since when an instance satisfies the ϵ requirement, its selection result is bounded by 2ϵ.

Finally, the result for the union of ranges is straightforward because selection can be evaluated for one range at a time. □

D. PERFORMANCE EVALUATION

D.1 More Results for Aggregates

Experimental Setup. In our experiments, each tuple has a tuple existence probability p that, by default, is uniformly sampled from [0, 0.5], denoted by $p_{max} = 0.5$. Each tuple, when existing, has two possible real values uniformly sampled from [0, 20]. This way, each tuple corresponds to a mixed type distribution with an existence



(a) MAX: Varying ϵ (Accuracy) (b) SUM: Varying ϵ (Accuracy) (c) SUM: LinCDF with different quantiles

(d) SUM: Varying number of values per tuple (e) Query 1 - Vary Object Location Std Dev (f) Query 2 - Vary Vehicle Location Std Dev

Figure 6: Additional experimental results of algorithms for MAX, SUM and 2 queries

probability and two possible values, or $\lambda = 3$ in our setting. (This data model was used in recent work on aggregates on uncertain data streams [10].) All experiments were run on a server with an Intel Xeon 3GHz CPU and 1GB memory running Java HotSpot 64-Bit server VM 1.6. Each reported result was averaged from 100 batches of tuples with the same setting after warming up the JVM.

We compare the performance of the deterministic and randomized algorithms. The former compute $(\epsilon, 0)$ approximation. The later computes (ϵ, δ) approximation; we use three values for δ, 0.1, 0.05 and 0.01 (or guarantees of 90%, 95% and 99%). From Theorem 3.1, the numbers of samples needed to meet the KS distance ϵ are $1.50/\epsilon^2$, $1.84/\epsilon^2$ and $2.65/\epsilon^2$. Note that for both `max` and `sum`, the result tuple existence probability can be computed exactly; thus the KS error is to quantify the approximate distributions only.

Accuracy Results for MAX. Fig. 6(a) shows that our deterministic algorithm, $\mathrm{D_{max}}$, is more accurate than the randomized algorithm, Rand, in KS error. This is because it sets its parameter ϵ', to meet the worst case scenario (i.e., reaching the maximum generation $\log \mathbb{U}$). In practice, the generations of the intervals are smaller than $\log \mathbb{U}$.

Additional Results for SUM. The accuracy of both algorithms, $\mathrm{D_{sum}}$ and Rand, is shown in Fig. 6(b). Given an ϵ, $\mathrm{D_{sum}}$ is always more accurate because the provisioning of errors to tuples not only guarantees the error bound, but also assumes the worst case when the errors are strictly additive. We observe that for some batches of tuples, Rand violates the accuracy requirement.

We also examine the optimization for SUM with different quantiles by running SUM using LinCDF with three types of quantiles: uniform, biased, and a mix of both. We set $\epsilon = 0.01$ and vary the window size W from 10 to 200. Fig. 6(c) shows the throughput. As W increases, the distribution of `sum` becomes smoother, and uniform quantiles cause worse performance since they require close to the maximum number of quantiles, $\frac{1}{\epsilon}$, to capture the higher curvature at the tails of a smooth distribution. Biased quantiles, on the other hand, require fewer quantiles to meet a given ϵ for a smooth distribution. Mixed quantiles combine the advantages of these two types of quantiles and in fact outperform them when W is large enough, e.g., $W > 40$. Therefore, in the other experiments for SUM, we employ LinCDF with mixed quantiles.

We vary the number of values per tuple λ in the SUM algorithm from 2 to 20. We set $\epsilon = 0.01$, $W = 100$, and $p_{max} = 0.1$. The throughput of both algorithms is shown in Fig. 6(d). The performance of the deterministic algorithm reduces fast as λ increases due to the relatively high cost of updating the LinCDF with linear interpolation, as shown in Theorem 3.3. As observed, under this setting, the randomized algorithm starts to outperform when $\lambda \geq 10$.

D.2 More Results for Query Plans

Q1. This query detects the violations of a fire code where the total weight of objects per area exceeds a threshold. We run inference ([20]) over a raw RFID reading stream to obtain an inferred object location stream. Each event in the output trace is an update of an object location, which is modeled by a Gaussian distribution. The objects are grouped into shelf areas ranging from 10 to 30 objects per shelf on average. The length of each shelf is 100cm and the standard deviation of object locations by default is in the range [3,5].

We vary the standard deviation of object locations from 2%, 4%, 7.5% to 15% of the shelf length. The two larger standard deviations indicate that the traces are highly noisy. In these cases, an object can belong to multiple groups and each reading triggers computation for a large number of objects in the group. We set the accuracy requirement $\epsilon = 0.01$. As expected, the throughput is decreased for both algorithms as shown in Fig. 6(e). The deterministic algorithm still outperforms the randomized algorithm for all settings considered. The cost of the deterministic algorithm for `sum` reduces somewhat faster because as the number of tuples under `sum` increases, the error provisioned for each tuple is reduced, hence the worse throughput.

Q2. This query originates from the Linear Road benchmark [2] for detecting congested freeway segments. The trace from the benchmark reports, at every time step, the most recent location and speed of vehicles. Both attributes are modeled by Gaussian distributions in our experiments. Q2 returns the maximum speed of vehicles on the congested segments, pre-determined to be 1 mile long each.

We vary the standard deviation of vehicle locations from 0.5% to 10% of a segment, which affects the window size for aggregation in each segment. We set $\mathbb{U} = 10000$ and $\epsilon = 0.05$ (the randomized algorithm is too slow for smaller values of ϵ). Fig. 6(f) shows that the deterministic algorithm is about an order of magnitude faster than the randomized one. Also, the result confirms that unlike `sum`, the deterministic algorithm for `max` is quite insensitive to the window size, since its complexity only depends on ϵ and $\mathbb{U}$.