vldb manuscript No.
(will be inserted by the editor)

CLARO: Modeling and Processing Uncertain Data Streams

Thanh T. L. Tran · Liping Peng · Yanlei Diao · Andrew McGregor · Anna Liu

the date of receipt and acceptance should be inserted later

Abstract Uncertain data streams, where data are incomplete and imprecise, have been observed in many environments. Feeding such data streams to existing stream systems produces results of unknown quality, which is of paramount concern to monitoring applications. In this paper, we present the CLARO system that supports stream processing for uncertain data naturally captured using continuous random variables. CLARO employs a unique data model that is flexible and allows efficient computation. Built on this model, we develop evaluation techniques for relational operators by exploring statistical theory and approximation. We also consider query planning for complex queries given an accuracy requirement. Evaluation results show that our techniques can achieve high performance while satisfying accuracy requirements, and outperform state-of-the-art sampling methods.

1 Introduction

Uncertain data management has become increasingly important in a wide range of applications. Much research in the literature has been motivated by traditional applications including data integration, information extraction, and sensor networks (e.g., [13, 20, 26, 30, 31]). More recently, research has shown that uncertain data management also plays an important role in large-scale scientific applications such as severe weather monitoring [10] and computational astrophysics [24]. Most scientific data that results from real-world measurements is inherently noisy and uncertain. Capturing uncertainty from input data to query output becomes a key component of scientific data management systems. Below we discuss several concrete applications that motivate our work.

T. T. L. Tran · L. Peng · Y. Diao · A. McGregor · A. Liu
University of Massachusetts, Amherst
E-mail: {ttran,lppeng,yanlei,mcgregor}@cs.umass.edu; anna@math.umass.edu

RFID Tracking and Monitoring. RFID readers deployed in a storage area return readings of tagged objects. RFID cleaning and inference methods [26] can translate noisy raw RFID data into location tuples (*time*, *tag_id*, *weight*, $(x, y)^p$), where x and y locations, two continuous-valued attributes, are probabilistic in nature (denoted by the letter p) due to the use of inference. A fire monitoring application could use the RFID deployment to detect violations of a fire code.

Query Q1 below triggers an alert when a flammable object is exposed to a high temperature. This query takes two inputs: a location stream as described above for flammable objects, and a temperature sensor stream with attributes (*time*, *sensor_id*, (x, y), *temp*), and joins them based on the location. The query is written as if the x and y locations were precise.

```
Q1: Select Rstream(R.tag_id, R.x, R.y, T.temp)
    From   FlammableObject [Now] As R,
           Temperature [Partition By sensor_id Rows 1] As T
    Where  T.temp > 60 °C and
           R.x = T.x and R.y = T.y
```

Another example query triggers an alert when the storage of flammable merchandise exceeds 200 pounds in each unit area. Query Q2 detects such violation 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 area to which they belong (where $AreaId()$ returns the id of the area given the object location and the area definition). For each group, Q2 computes the total weight of objects and raises an alert if the weight exceeds 200 pounds.

```
Q2: Select group_id, sum(S.weight)
    From Locations S [Partition By tag_id Rows 1]
    Group By AreaId(S.(x,y), AreaDef) as group_id
    Having sum(S.weight) > 200
```

Computational Astrophysics. Massive astrophysical surveys will soon generate observations of 10^8 stars and galaxies at nightly data rates of 0.5TB to 20TB [24]. 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 continuous distributions, denoted by (id, $time$, $(x, y)^p$, $luminosity^p$, $color^p$). Then queries can be issued to detect dynamic features, transient events, and anomalous behaviors. Query Q3 below detects regions of the sky of high luminosity from the observations in the past hour. Similar to Q2, it groups the objects into predefined regions and reports the regions where the maximum luminosity exceeds a threshold.

```
Q3: 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
```

Problem statement. In this paper, we address uncertain data stream processing for data naturally modeled by *continuous random variables*, such as many types of sensor data, scientific data, and financial data. We aim to support relational query processing on such uncertain data by fully characterizing the distribution of each tuple produced. Such distributions, called *result tuple distributions*, can be exact or approximate. If approximate distributions are returned for performance reasons, we further consider the interaction of the operators in a complex query and seek a query plan that meets an arbitrary given accuracy requirement.

Challenges. Uncertain data stream processing as described above raises two challenges: First, it is computationally difficult to obtain result distributions when input tuples are modeled using continuous random variables. Such computation often involves multivariate integrals or requires new algorithms to be designed. Second, such computation must be performed for high-volume data streams. While approximation is a common approach to improving efficiency, it must be able to achieve a small bounded error while meeting stringent performance requirements.

Despite a flurry of recent work on uncertain data management, the above two challenges have not been sufficiently addressed for several reasons. First of all, most probabilistic databases, e.g., [2,3,7,22,30], and probabilistic stream systems, e.g., [6,15,16], model tuples using *discrete* random variables and evaluate queries using the possible worlds semantics. The continuous nature of our data, however, precludes the use of these techniques as the possible values of a continuous random variable cannot be enumerated.

Second, the state-of-the-art techniques for continuous random variables employ either multivariate integration, discretization, or Monte Carlo simulation. The integral-based approach to aggregation [5] performs n-1 integrals to compute the sum of n tuples. Its computation is too slow for stream processing. The discretization approach [1,23] approximates continuous distributions using discrete ones, and uses the possible worlds semantics to evaluate queries. This approach, however, does not offer any accuracy guarantee and can often be inefficient, as we will show in our experiments. The Monte Carlo approach [11,13] samples from the input distributions and computes the result distribution from the samples. While it is a simple and general approach, a large number of samples is often needed to achieve high accuracy, making it slow for stream processing, as we will also show in our performance evaluation.

Third, conditioning operations (e.g., selections, group-bys) on uncertain attributes complicate the problem further. These operations can introduce uncertainty about the tuple existence. This tuple existence, indicating whether a tuple is present in a relation, is modeled by a discrete random variable (e.g., a Bernoulli distribution). Hence, for complex queries we must handle both continuous and discrete random variables. Moreover, tuples of conditioned distributions make it harder to compute the distributions of aggregates. Previous work on aggregates in probabilistic databases was restricted not only to discrete distributions but also to just the moments of result distributions, e.g., the mean of `max` and `min` [14,15,6], the mean and variance of `sum`, and some higher moments of `count` [6]. In contrast, our work aims to characterize full distributions of these aggregates. Moreover, just knowing a few moments of aggregates is not enough to answer queries accurately, as we will demonstrate in our performance study.

Contributions. In this paper, we present a probabilistic data stream system, called CLARO, that supports relational processing of uncertain data streams modeled by continuous random variables.[1] CLARO provides a systems framework for continuous uncertain data, including a data model, formal semantics, evaluation techniques of relational operators, and query planning for complex queries. Our main contributions include:

Data model. The foundation of CLARO is a unique data model, named *mixed-type model*. In this model, continuous uncertain attributes follow *Gaussian mixture distributions*, which can model complex real-world distributions [18]. They also allow us to develop efficient solutions for many relational operators. Besides the attribute-level uncertainty captured by such distribu-

[1] Our techniques can also be applied to probabilistic databases by using a sequential scan to produce a stream of tuples.

tions, the mixed-type model can also capture tuple-level uncertainty regarding the existence of a tuple.

Formal semantics. In like manner that the possible worlds semantics (PWS) [7] laid the foundation for query processing on discrete uncertain data, we propose formal semantics for relational processing under our model for continuous uncertain data. Our formal semantics, based on measure theory, is shown to be equivalent to PWS when used in the discrete case.

Joins. Our choice of data model enables efficient evaluation techniques for joins. We propose two types of joins to suit different application semantics. The first type models equi-joins on continuous-valued uncertain attributes as a join of an input stream and a probabilistic view. CLARO supports such joins with regression techniques to construct the view and then gives a *closed-form* solution for result distributions. The second type pairs tuples from two inputs by a cross-product. We show that there are also *closed-form* result distributions for such joins.

Aggregates of Gaussian mixture distributions. Our data model also empowers us to design efficient techniques for aggregates such as `sum` and `avg`. When the tuple existence is certain, we show that there are *exact* result distributions of aggregates, which eliminate the use of integrals. In workloads when the exact solution is slow, we derive *approximate* distributions with bounded errors to improve efficiency. These techniques, when used as a hybrid solution, can meet arbitrary accuracy requirements while achieving high speed.

Aggregates under the mixed-type model. We observe that conditioning operations (e.g., selection) can introduce uncertainty regarding tuple existence, which complicates the computation for aggregates. We propose an *approximate* evaluation framework for the mixed-type model that includes tuple existence probabilities. Within this framework, we develop deterministic and randomized approximation algorithms with error bounds for common aggregates like `max`, `min`, `sum`, and `count`.

Query planning. A unique aspect of CLARO is its ability to meet arbitrary accuracy requirements even for complex queries. Given a complex query, we arrange the operators to first apply the closed-form solutions and then approximation algorithms if needed. Starting from the first approximate operator in the query plan, we quantify the errors of this operator as well as all subsequent operators. Based on the above results, we can provision an error bound for each operator to meet an overall query accuracy requirement.

Performance evaluation. We perform a thorough evaluation of our techniques and compare them with state-of-the-art sampling techniques. Our main results are as follows: (i) When the tuple existence is certain, our algorithms for joins and aggregates consistently outperform histogram-based sampling [11]. (ii) When the tuple existence is uncertain, our deterministic algorithm for `max` is faster than our randomized algorithm by orders of magnitude. For `sum`, there is a tradeoff between the two, depending on the workloads. (iii) For complex queries involving conditioning and aggregation, our evaluation used real data and queries from the applications of object tracking and computational astrophysics. Results show that our system can meet any given accuracy requirement while achieving throughput of thousands of tuples per second or higher for most workloads tested.

2 Data Models

We now present our data model, called mixed-type model, for relational processing in CLARO.

2.1 Gaussian Mixture Model

Our choice for modeling continuous-valued attributes is Gaussian mixture models, abbreviated GMMs [18]. As an instance of probability mixture models, a GMM describes a probability distribution using a convex combination of Gaussian distributions.

Definition 1 A Gaussian mixture model for a continuous random variable X is a mixture of m Gaussian variables $X_1, X_2, \cdots, X_m$. The probability density function (pdf) of X is:

$$f_X(x) = \sum_{i=1}^{m} p_i f_{X_i}(x), \qquad f_{X_i}(x) = \frac{1}{\sigma_i\sqrt{2\pi}} e^{-\frac{(x-\mu_i)^2}{2\sigma_i^2}},$$

where $0 \le p_i \le 1$, $\sum_{i=1}^{m} p_i = 1$, and each mixture component X_i is a Gaussian distribution with mean μ_i and variance σ_i^2, or abbreviatedly $X_i \sim \mathcal{N}(\mu_i, \sigma_i^2)$.

Definition 2 A multivariate Gaussian mixture model for a random vector $\mathbf{X}$ naturally follows from the definition of multivariate Gaussian distributions:

$$f_{\mathbf{X}}(\mathbf{x}) = \sum_{i=1}^{m} p_i f_{\mathbf{X}_i}(\mathbf{x}),$$
$$f_{\mathbf{X}_i}(\mathbf{x}) = \frac{1}{(2\pi)^{k/2}|\Sigma_i|^{1/2}} e^{-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu}_i)^T \Sigma_i^{-1}(\mathbf{x}-\boldsymbol{\mu}_i)},$$

where k is the size of the random vector, and each mixture component $\mathbf{X}_i$ is a k-variate Gaussian distribution with mean $\boldsymbol{\mu}_i$ and covariance matrix Σ_i.

CLARO adopts GMMs due to several key benefits of these models. First, theoretical results have shown that GMMs can approximate any continuous distribution arbitrarily well [18]. Hence, they are suitable for

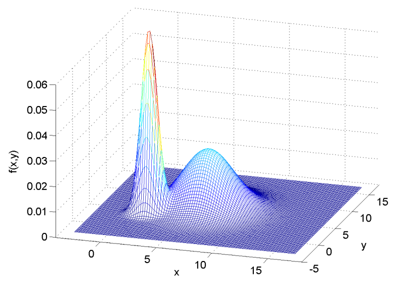


Fig. 1 Location distribution of a recently moved object detected using RFID readers.

modeling complex real-world distributions. Consider an example in the RFID tracking application. Fig. 1 shows the inferred location distribution of a recently moved object [26]. Here, the bivariate, bimodal GMM represents the possibilities of the old and new locations using two mixture components; each component is a bivariate Gaussian modeling the joint distribution of x and y locations. The second benefit of GMMs is efficient computation based on Gaussian properties and statistical theory. The mean and variance of GMMs can be computed directly from those of the mixture components: $E[X] = \sum_{i=1}^{m} p_i E[X_i]$, and $Var[X] = \sum_{i=1}^{m} p_i (Var[X_i] + (E[X_i])^2) - (E[X])^2$. Other computational benefits of GMMs, such as the characteristic functions and product distributions are described in the relevant later sections.

2.2 Mixed-type Model for Relational Processing

Input model. We now describe the model of the input streams. 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 $\mathbb{R}^m$. In our model, $\mathbf{X}$ are captured by a multivariate GMM. If the attributes are independent, this joint pdf may be further partitioned into independent (univariate) GMMs.

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 3 Given a tuple with m continuous uncertain attributes, denoted by $\mathbf{A}^x$, n discrete uncertain attributes $\mathbf{A}^y$, 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 $f(\mathbf{x},\mathbf{y}) = f_{\mathbf{A}^x|\mathbf{A}^y}(\mathbf{x}|\mathbf{y}) \cdot \mathbb{P}[\mathbf{A}^y = \mathbf{y}]$. Then, g characterizes a random vector $(\mathbf{X},\mathbf{Y},\mathbf{Z})$ over $(\mathbb{R}^m \times \mathbb{U}^n \times \mathbf{A}^d) \cup \{\perp\}$ ($\perp$ denotes the non-existence case that is all-or-none over the variables in $(\mathbf{X},\mathbf{Y},\mathbf{Z})$, where

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

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

for $I \subseteq \mathbb{R}^m, \mathbf{y} \in \mathbb{U}^n$.

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

Consider a simple example of applying a selection predicate $a \leq X \leq b$ on an attribute X characterized by a pdf $f(x)$. Given a value x drawn from the pdf, the selection returns x if it satisfies the predicate, and $\perp$ otherwise. The probability that a value is returned is $\int_a^b f(x)dx$, which is the TEP of the result tuple. The returned distribution is truncated to the selection interval $[a,b]$, and then normalized to remain a proper distribution. The result tuple is then characterized by two components, a TEP and a pdf, hence mixed-type. In the context of a query, Fig. 2 illustrates the execution of the group-by operator in Q2 under the mixed-type model. (For simplicity, we consider the dimension x and omit y.) Each group corresponds to a selection on the x location using the interval associated with that group. Then, for a given group and a distribution of x, group-by is performed as in the above example.

An important note on the mixed-type model is that 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, e.g., the attribute "weight" after group-by in Fig. 2 is Bernoulli. If an attribute was uncertain and modeled by a continuous distribution, it is now modeled by a mixed-type distribution capturing the joint event that the tuple exists and the attribute follows the continuous distribution. Also, discrete uncertain attributes can emerge as derived attributes, e.g., as the result of aggregating Bernoulli variables. Therefore, in general, relational processing needs to consider both discrete and continuous random variables under the mixed-type model.

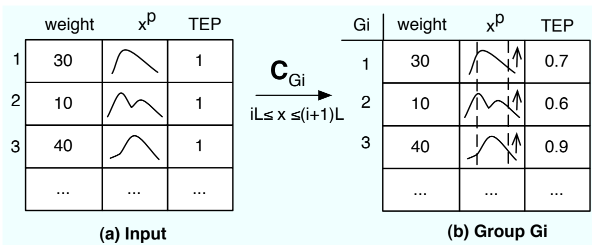


Fig. 2 Group-bys in Q2 under the mixed-type model.

Our current data model does not handle correlations among tuples. Inter-tuple correlations can be handled using lineage [3] and Monte Carlo simulation [13]. Our work can be viewed as an optimization of these general systems when query processing does not produce correlated intermediate results.

3 Formal Semantics of Relational Processing

We now propose the formal semantics of relational operations under our mixed-type data model. (Note that for mixed-type tuples, a continuous uncertain attribute can follow any distribution, not restrictedly a Gaussian mixture model.) The formal semantics is crucial because it states the intended answer of each operation under the chosen data model, hence ensuring the *correctness* of query processing. A key observation is that the possible worlds semantics (PWS) does not apply to continuous random variables. First, the values of a continuous random variable are uncountable. Second, the probability of each possible world is simply zero. Hence, we cannot construct possible worlds by enumerating values of a continuous random variable and merge the results of the possible worlds to get the result distribution. To address this issue, we propose to use *measure theory* to quantify the probabilities associated with subsets of values taken from the domain of a random variable. We first state the definition of probability space [4].

Definition 4 Probability Space. In measure theory, a probability space of a random variable X is a triple $(S_X, \mathcal{F}_X, P_X)$ where S_X is the sample space consisting of all possible values of X, $\mathcal{F}_X$ is the σ-field over S_X, and P_X is the probability measure capturing the probability of any set in the σ-field.

A σ-field over S_X is a nonempty collection of subsets of S_X that contains the empty set, is closed under complementation and countable unions of its members. There can be many σ-fields associated with a sample space. For probability space of a random variable, we only concern with the smallest one that contains all of the open sets in the sample space S. For example, if S_X is the real line, then $\mathcal{F}_X$ is chosen to contain all sets of the form $[a, b]$, $(a, b]$, (a, b), and $[a, b)$, for all real numbers a and b (the closed intervals are due to complementation). The measure of the entire sample space is 1, or $P_X(S_X) = 1$.

We now define the probability space of our mixed-type distributions. To focus on the main idea, we first omit discrete random variables and discuss the extension to them near the end of this section.

Definition 5 Probability Space of Mixed-type Distributions. Consider a random vector $\mathbf{X}$ described by a mixed-type distribution (p, f) where p is the existence probability and f is the density function over $\mathbb{R}^m$. The probability space for $\mathbf{X}$ is characterized by: (1) the sample space $S_\mathbf{X} = \mathbb{R}^m \cup \{\perp\}$, where $\perp$ denotes the non-existence case, (2) the σ-field $\mathcal{F}_\mathbf{X}$ over $S_\mathbf{X}$, and (3) the probability measure $P_\mathbf{X}$ such that given any set A in the σ-field $\mathcal{F}_\mathbf{X}$, $P_\mathbf{X}(A) = (1-p)\mathbb{1}(\perp \in A) + p\int_{A\setminus\{\perp\}} f(\mathbf{x})d\mathbf{x}$.

We next use measure theory to define the semantics of the relational operations. More precisely, given the probability space of input tuples, we define the probability space of each output tuple.

3.1 Projection

Let $(\mathbf{X}, \mathbf{Y})$ be a random vector following a mixed-type distribution $(p, f_{\mathbf{XY}})$. Consider the projection of $(\mathbf{X}, \mathbf{Y})$ onto $\mathbf{Y}$, i.e., projecting out $\mathbf{X}$. Suppose that the domain of $\mathbf{X}$ is $\mathbb{R}^{|\mathbf{X}|}$, and the domain of $\mathbf{Y}$ is $\mathbb{R}^{|\mathbf{Y}|}$.

The probability space of the projection result has three items, the sample space $S_\mathbf{Y} = \mathbb{R}^{|\mathbf{Y}|} \cup \{\perp\}$, the σ-field $\mathcal{F}_\mathbf{Y}$ over $S_\mathbf{Y}$, and the probability measure defined for any set A in $\mathcal{F}_\mathbf{Y}$ as follows.

1. If $A = \{\perp\}$, then $P_\mathbf{Y}(A) = 1 - p$.
2. If $A \subset \mathbb{R}^{|\mathbf{Y}|}$, then
$$P_\mathbf{Y}(A) = p\int_A \int_{\mathbb{R}^{|\mathbf{X}|}} f_{\mathbf{XY}}(\mathbf{x}, \mathbf{y})d\mathbf{x}d\mathbf{y}.$$
3. For any set A that contains both $\perp$ and a subset of the domain, its probability is the sum of the probabilities of the two cases.

In fact, the third case, as a property of measure theory, holds for all other operations; hence we will not mention it explicitly hereafter.

3.2 Selection

Let $\mathbf{X}$ be a random vector following a mixed-type distribution $(p, f_\mathbf{X})$ with the probability space $(S_\mathbf{X}, \mathcal{F}_\mathbf{X}, P_\mathbf{X})$, where $S_\mathbf{X} = \mathbb{R}^{|\mathbf{X}|} \cup \{\perp\}$. Let $\bar{\mathbf{X}}$ be the output of the selection $\mathbf{X} \in I$, where I is the selection region. The probability space of $\bar{\mathbf{X}}$ has the same sample space and σ-field as that of $\mathbf{X}$. To define the probability measure, we first define the *selection probability*, q, to be the probability mass of f under the region I, i.e., $q = \int_{\mathbb{R}^{|\mathbf{X}|}\cap I} f_\mathbf{X}(\mathbf{x})d\mathbf{x}$. Consider a set A in $\mathcal{F}_{\bar{\mathbf{X}}}$.

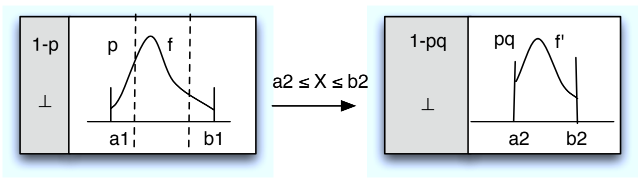


Fig. 3 Selection under the mixed-type model

1. If $A = \{\perp\}$, then
$$P_{\bar{\mathbf{X}}}(A) = (1-p) + p\int_{\mathbb{R}^{|\mathbf{X}|}\setminus I} f_{\mathbf{X}}(\mathbf{x})d\mathbf{x} = 1 - pq.$$
2. If $A \subset \mathbb{R}^{|\mathbf{X}|}$, then $P_{\bar{\mathbf{X}}}(A) = p\int_{A\cap I} f(\mathbf{x})d\mathbf{x}$.

Fig. 3 illustrates the result of a selection of a tuple X following a mixed-type distribution (p, f), whose support of f is $[a_1, b_1]$. The selection on X using the condition $a_2 \leq X \leq b_2$ results in another distribution with reduced support $[a_2, b_2]$ and reduced TEP as defined above in the probability measure. In §4.1, we will describe the steps to obtain the result distributions.

3.3 Cross Product

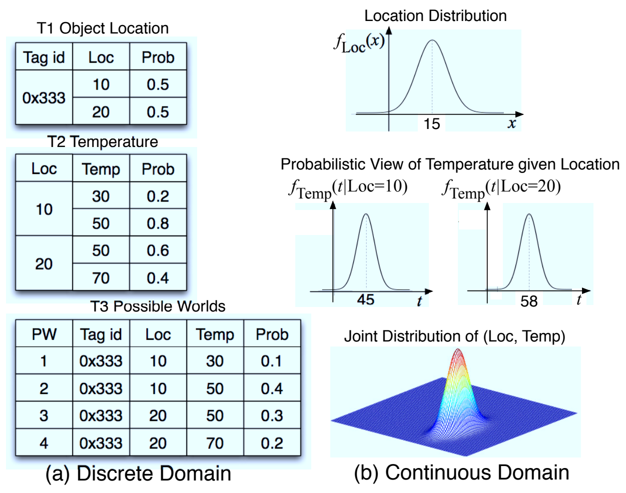
Consider two independent random vectors, $\mathbf{X}$ and $\mathbf{Y}$. Let their probability spaces be $(S_{\mathbf{X}}, \mathcal{F}_{\mathbf{X}}, P_{\mathbf{X}})$, and $\mathbf{Y}$ be $(S_{\mathbf{Y}}, \mathcal{F}_{\mathbf{Y}}, P_{\mathbf{Y}})$, respectively.

The cross product of $\mathbf{X}$ and $\mathbf{Y}$ corresponds to the joint distribution of the pair $(\mathbf{X}, \mathbf{Y})$. We now characterize the probability space of $\mathbf{X} \times \mathbf{Y}$, denoted as $(S_{\mathbf{XY}}, \mathcal{F}_{\mathbf{XY}}, P_{\mathbf{XY}})$. The probability space $S_{\mathbf{XY}}$ is $(\mathbb{R}^{|\mathbf{X}|} \times \mathbb{R}^{|\mathbf{Y}|}) \cup \{\perp\}$. Due to our convention of all-or-none existence among the variables, we define that the cross product exists when both $\mathbf{X}$ and $\mathbf{Y}$ exists. Therefore,

1. If $A = \{\perp\}$, then $P_{\mathbf{XY}}(A) = 1 - p_{\mathbf{X}}p_{\mathbf{Y}}$.
2. For any $A \subset (\mathbb{R}^{|\mathbf{X}|} \times \mathbb{R}^{|\mathbf{Y}|})$,
$$P_{\mathbf{XY}}(A) = p_{\mathbf{X}}p_{\mathbf{Y}}\iint_A f_{\mathbf{X}}(\mathbf{x})f_{\mathbf{Y}}(\mathbf{y})d\mathbf{x}d\mathbf{y}.$$

3.4 Join using Probabilistic View

We first introduce the concept of a probabilistic view. Consider the scenario when some attributes $\mathbf{Y}$ depend on some other attributes $\mathbf{X}$ as follows. For a given $\mathbf{x}$, there is a distribution of $\mathbf{Y}$, $f_{\mathbf{Y}}(\mathbf{y}|\mathbf{X} = \mathbf{x})$. Then we say that $\mathbf{Y}$ *view-depends* on $\mathbf{X}$ and the collection of these distributions for all values of $\mathbf{x}$ is a *probabilistic view*. The formal definition is given in §4.3.1. We denote the existence of the view with $p_{\mathbf{Y}|\mathbf{X}=\mathbf{x}}$. For $\mathbf{x}$ where the view is defined, $p_{\mathbf{Y}|\mathbf{X}=\mathbf{x}} = 1$; otherwise, $p_{\mathbf{Y}|\mathbf{X}=\mathbf{x}} = 0$. Now given a tuple with the attributes $\mathbf{X}$ following a distribution $(p_{\mathbf{X}}, f_{\mathbf{X}})$, the join of this tuple with the


T1 Object Location
T2 Temperature
T3 Possible Worlds
Location Distribution
$f_{\mathrm{Loc}}(x)$
15
x
Probabilistic View of Temperature given Location
$f_{\mathrm{Temp}}(t|\mathrm{Loc}=10)$
$f_{\mathrm{Temp}}(t|\mathrm{Loc}=20)$
45
t
58
t
Joint Distribution of (Loc, Temp)
(a) Discrete Domain
(b) Continuous Domain

Tag id	Loc	Prob
0x333	10	0.5
	20	0.5

Loc	Temp	Prob
10	30	0.2
	50	0.8
20	50	0.6
	70	0.4

PW	Tag id	Loc	Temp	Prob
1	0x333	10	30	0.1
2	0x333	10	50	0.4
3	0x333	20	50	0.3
4	0x333	20	70	0.2

Fig. 4 Compare equi-joins in the discrete domain (using PWS) and in the continuous domain (using a probabilistic view).

view is characterized by the joint distribution that pairs each value of $\mathbf{X}$ with the corresponding distribution of $\mathbf{Y}$ from the view.

Let the probability space for the random vector $\mathbf{X}$ of the join be $(S_{\mathbf{X}}, \mathcal{F}_{\mathbf{X}}, P_{\mathbf{X}})$, and the mixed-type distribution of $\mathbf{X}$ be $(p_{\mathbf{X}}, f_{\mathbf{X}})$. Let the probability space for the random variable $\mathbf{Y}$ given $\mathbf{X} = \mathbf{x}$ (in the probabilistic view) be $(S_{\mathbf{Y}|\mathbf{x}}, \mathcal{F}_{\mathbf{Y}|\mathbf{x}}, P_{\mathbf{Y}|\mathbf{x}})$ and its distribution be $(p_{\mathbf{Y}|\mathbf{X}=\mathbf{x}}, f_{\mathbf{Y}|\mathbf{X}=\mathbf{x}})$. Then the joint probability space for $(\mathbf{X}, \mathbf{Y})$ is characterized with the sample space $S_{\mathbf{X}} \times S_{\mathbf{Y}|\mathbf{x}}$, the σ-field $\mathcal{F}_{\mathbf{XY}}$, and the probability measure $P_{\mathbf{XY}}$, where for $A \in \mathcal{F}_{\mathbf{XY}}$:

1. If $A = \{\perp\}$, then $P_{\mathbf{XY}}(A) = 1 - p_{\mathbf{X}}q$, where $q = \int_{\mathbb{R}^{|\mathbf{X}|}} p_{\mathbf{Y}|\mathbf{X}=\mathbf{x}} f_{\mathbf{X}}(\mathbf{x})d\mathbf{x}$.
2. If $A \subset (\mathbb{R}^{|\mathbf{X}|} \times \mathbb{R}^{|\mathbf{Y}|})$, then
$$P_{\mathbf{XY}}(A) = p_{\mathbf{X}}\iint_A p_{\mathbf{Y}|\mathbf{X}=\mathbf{x}} f_{\mathbf{X}}(\mathbf{x}) f_{\mathbf{Y}|\mathbf{X}=\mathbf{x}}(\mathbf{y})d\mathbf{x}d\mathbf{y}.$$

Consider the example query Q1. In this query, there are two input streams, the location stream and the temperature reading stream. Fig. 4 illustrates the execution of this query for both discrete and continuous domains, assuming one-dimensional location x. The known possible worlds semantics is used for the discrete case. Now we consider the continuous case.

Let $(p_i, f_{Xi}(x))$ be the mixed-type distribution for object i, where $f_{Xi}(x)$ is the distribution of its location. Assume that at location x, a temperature sensor obverses the temperature $f_{T|x}(t)$. (In §4.3.1, we will discuss techniques to construct this distribution from temperature readings.) The collection of all of these observations forms a probabilistic view of temperature given object location. In general, a probabilistic view can be characterized with both a distribution and an existence probability $p_{T|x}$. Depending on implementation choice,

$p_{T|x}$ can be set to 1, if there are enough observations for the view. Then, the query computes the temperature of each object in the location stream, which is a join with the probabilistic view. Using the above definition, we can quantify the probability space of the joint distribution of location and temperature for each object i with existence probability p_i and $f_i(x,t) = f_{Xi}(x) \cdot f_{T|x_i}(t)$. For the general case when the view may not exist for some locations (i.e., $p_{T|x} \leq 1$), let $q_i = \int_{\mathbb{R}} f_{Xi}(x) p_{T|x} dx$; then q_i denotes the existence probability of the view given object i. In this case, the new TEP of the join result is $p'_i = p_i q_i$.

3.5 Aggregation

SUM. Let $Y = X_1 + X_2$. Consider the simple case where X_1 and X_2 are univariate and independent. X_i follows a mixed-type distribution (p_{X_i}, f_{X_i}) and has the probability space $(\mathbb{R} \cup \{\perp\}, \mathcal{F}_{X_i}, P_{X_i})$, $i = 1, 2$. Then, the sum Y has probability space characterized with the same sample space $S_Y = \mathbb{R} \cup \{\perp\}$. Note that since X_1 and X_2 can either exist or not, there are four combinations of how X_1 and X_2 contribute to the sum. The probability measure is hence defined for any $A \in \mathcal{F}_Y$ as follows.

1. If $A = \{\perp\}$, then $P_Y(A) = (1 - p_{X_1})(1 - p_{X_2})$.
2. If $A \subset \mathbb{R}$, then, $P_Y(A) =$

$$p_{X_1}(1 - p_{X_2}) \cdot \int_{x \in A} f_{X_1}(x)dx + (1 - p_{X_1})p_{X_2} \cdot \int_{x \in A} f_{X_2}(x)dx$$
$$+ p_{X_1}p_{X_2} \cdot \int_{x_1 + x_2 \in A} f_{X_1}(x_1) f_{X_2}(x_2) dx_1 dx_2.$$

In general, `sum` of n independent random variables can be obtained using induction.

COUNT. In our model, `count` is equivalent to the `sum` of Bernoulli random variables. The above semantics for `sum` can be directly adapted to `count` by replacing the probability density functions (pdfs) with the probability mass functions (pmfs), and replacing integration with summation. Also, the sample space of `count` is the set of natural numbers $\mathbb{N}$, i.e., does not include $\perp$ — `count` is 0 when none of the tuples exists. This is the same as the possible worlds semantics.

MIN and MAX. The semantics for these aggregates are defined similarly to that for `sum`; the only difference is the integration region in the last term of the probability measure. For example, for `max`, the integration region is $\texttt{max}(x_1, x_2) \in A$.

AVG. Since `avg=sum/count`, and `count` is probabilistic due to the uncertainty of tuple existence, `avg` is more complicated than `sum` and cannot be defined using induction. Generally, it is defined by enumerating all combinations of the input tuples' existence. Consider $Y = \texttt{avg}(X_1, X_2, X_3)$. Given a set A in the σ-field of Y,

1. If $A = \{\perp\}$, $P_Y(A) = (1 - p_{X_1})(1 - p_{X_2})(1 - p_{X_3})$
2. If $A \subset \mathbb{R}$, then

$$P_Y(A) = p_{X_1}(1 - p_{X_2})(1 - p_{X_3}) \int_{x \in A} f_{X_1}(x_1)dx_1$$
$$+(1 - p_{X_1})p_{X_2}(1 - p_{X_3}) \int_{x \in A} f_{X_2}(x_2)dx_2$$
$$+(1 - p_{X_1})(1 - p_{X_2})p_{X_3} \int_{x \in A} f_{X_3}(x_3)dx_3$$
$$+p_{X_1}p_{X_2}(1 - p_{X_3}) \int_{(x_1+x_2)/2 \in A} f_{X_1}(x_1)f_{X_2}(x_2)dx_1dx_2$$
$$+p_{X_1}(1 - p_{X_2})p_{X_3} \int_{(x_1+x_3)/2 \in A} f_{X_1}(x_1)f_{X_3}(x_3)dx_1dx_3$$
$$+(1 - p_{X_1})p_{X_2}p_{X_3} \int_{(x_2+x_3)/2 \in A} f_{X_2}(x_2)f_{X_3}(x_3)dx_2dx_3$$
$$+p_{X_1}p_{X_2}p_{X_3} \int_{(x_1+x_2+x_3)/3 \in A} f_{X_1}(x_1)f_{X_2}(x_2)f_{X_3}(x_3)dx_1dx_2dx_3$$

If there are more than 3 random variables, the semantics is defined similarly by enumerating a number of terms exponential in the number of input tuples.

3.6 Group-by Aggregation

Group-by aggregation involves repeated selections, with a different condition per group. If selections involve deterministic attributes, then the participation of a tuple in a group is certain, and when it is a group, the TEP remains the same. Aggregation of a set of tuples in a group is defined as above. If selections involve uncertain attributes, we will first use the definition of selection to obtain the selection results, and then use the definition of aggregation to obtain the results of group-by aggregation.

Now consider queries Q2 and Q3, which are group-by aggregation queries. Since they are similar, we focus on Q3 here. Each object in the input stream is characterized with the distributions of its location $f_X(x)$ and luminosity $f_Y(y)$; the tuple existence probability p is assumed to be 1. The objective is to define the probability space of `max(S.luminosity)` for each group. Let the condition of the i-th group be $x \in [iL, (i+1)L]$, where L denotes the group length. For object i, the selection probability is $q_i = \int_{x \in [iL,(i+1)L]} f_{Xi}(x)dx$. The new TEP of object i in this group is $p_i = q_i$. The result distribution of X_i has a probability space defined as in §3.2. Then we can characterize the distribution of `max`(S.luminosity), which is $M_i = \texttt{max}_k(f_{Yk}(y) \cdot \mathbb{1}(k \in Group(i)))$, using the semantics for `max`.

3.7 Equivalence to Possible Worlds Semantics

For discrete random variables characterized by probability mass functions (pmfs), instead of probability density functions (pdfs), the above definitions can still apply by replacing integration with summation in the formulas for probability measures. This is the same as the possible worlds semantics (PWS), which has been defined for discrete random variables. Hence, our proposed semantics is consistent with the PWS. In general, for mixed-type distributions involving both discrete and continuous attributes, the formulas for probability measures use both integration and summation.

4 Relational Processing under Mixed-type Model

In this section, we consider the processing of relational operations under the mixed-type model. We show that for a substantial subset of operations, there are exact, closed-form solutions. For other operations, we devise approximate result distributions with bounded errors.

4.1 Selections

We first consider selections under the mixed-type model, which involve applying a condition on some attributes of mixed-type tuples. In §3.2, we define the semantics of selection by characterizing its result distribution using probability space. We now state how to obtain this result distribution.

Let t be a tuple following a mixed-type distribution $g = (p, f)$, and let S be the support of $f(\mathbf{x})$ such that S is a subset of the domain of f, and $f(\mathbf{x}) \neq 0$ for any $\mathbf{x}$ in S. Consider a selection that applies a range condition $\mathbf{x} \in I$ to (one or many) uncertain attributes in t. Let $\bar{t}$ denote the result tuple. Then, its distribution $\bar{g} = (\bar{p}, \bar{f})$ is computed as follows.

1. Compute the selection probability q, which is the the probability mass of f in the selection range I, $q = \int_{S \cap I} f(\mathbf{x}) d\mathbf{x}$.
2. Compute the new tuple existence probability, $\bar{p} = p \cdot q$.
3. Truncate the joint distribution so that its support is restricted to the intersection of the original support S and the selection range I, $\bar{S} = S \cap I$.
4. Normalize the truncated distribution, $\bar{f}(\mathbf{x}) = f(\mathbf{x})/q$.

Note that a group-by operation applies repeated selections with a different condition for every group, hence the above process can be applied to compute the result distribution of each tuple in a group. We will mention *conditioning* operations when referring to both selections and group-bys later.

4.2 Projections

A projection is equivalent to integrating over the attributes that are projected out, or not in the projection list. For example, if an attribute x_i from the attributes $\mathbf{x}$ is projected out, the new distribution is $f'(\mathbf{x} \backslash \{x_i\}) = \int_{\mathbb{R}} f(\mathbf{x}) dx_i$. If f is a GMM, this is a marginalization of a multivariate GMM to get a GMM of lower dimension. Under the mixed-type model, the result tuple follows a mixed-type distribution with the same tuple existence probability while the continuous distribution is the result of marginalization.

4.3 Joins

We next consider efficient evaluation for joins under our data model. We propose two types of joins of continuous random attributes to suit different application semantics. The first type deals with the complexities associated with *equi-joins* on attributes modeled by continuous random variables, which is the focus of this section. Our system employs a *probabilistic view* to facilitate such joins and offers a closed-form solution for inputs characterized by Gaussian mixture models (GMMs), and more general mixed-type models. The other (traditional) type of join pairs tuples from two inputs for *inequality* comparison and is modeled by a cross-product followed by a selection [23]. We also show that cross-products have exact result distributions under our data model.

4.3.1 Joins using Probabilistic Views

We now discuss equi-joins for continuous random variable. Revisit the query Q1, which, retrieves the temperature for any given object location. The two common join attributes (X, Y) in the object location stream are uncertain and modeled by continuous random variables (in contrast, the join attributes in the temperature stream are the fixed sensor locations and hence deterministic). This kind of joins is inherently difficult to support, since given a tuple i, the value of (X_i, Y_i) at a specific location has probability 0, any join result that pairs a specific value of the location tuple and a temperature tuple also has probability 0.

To attain proper result distributions for such joins, we use the notion of **probabilistic views**, as introduced in §3.4. In Q1, a probabilistic view on the temperature stream is the distribution of *Temp* given (x, y), denoted by $f_{Temp}(t|x, y)$. Then, for each tuple i in the location stream, the process of iterating all possible values of (X_i, Y_i) and retrieving the corresponding temperature distribution can be compactly represented by $f_{X_i,Y_i}(x, y) f_{Temp}(t|x, y)$, yielding a joint distribution $f_{X_i,Y_i,Temp}(x, y, t)$. Below we give formal definitions of such equi-joins using probabilistic views.

Definition 6 We denote a stream by $S(\mathbf{A}^*, \mathbf{B}^*, \bar{\mathbf{S}})$, where $\mathbf{A}$ is a vector of attributes that are deterministic or probabilistic (denoted by $\mathbf{A}^* = \mathbf{A}$ or $\mathbf{A}^p$), $\mathbf{B}$ is another vector of deterministic or probabilistic attributes related to attributes in $\mathbf{A}$, and $\bar{\mathbf{S}}$ is a vector for the rest of the attributes. The probabilistic view of $\mathbf{B}$ as a function of $\mathbf{A}$, denoted by $V_{\mathbf{B}|\mathbf{A}}(S)$, is a distribution of $\mathbf{B}$ for a given value of $\mathbf{A}$, characterized by $f_{\mathbf{B}}(\mathbf{b}|\mathbf{A} = \mathbf{a})$.

Given an equi-join query, the suitable probabilistic view can be recognized by the query compiler based on the following observation: as in traditional databases, equi-joins only include one copy of the join attributes, say $R.(X, Y)$, in the output; the other copy of the join attributes, $S.(X, Y)$, is used only for comparison but suppressed from the output. This implies a probabilistic view for the input, S, and the view is defined for the S attributes included in the output, e.g., $S.Temp$, dependent on the join attributes.

Definition 7 Given two independent streams $R(\mathbf{A}^p, \bar{\mathbf{R}})$ and $S(\mathbf{A}^*, \mathbf{B}^*, \bar{\mathbf{S}})$ with the probabilistic view $V_{\mathbf{B}|\mathbf{A}}(S)$, an equi-join of R and S on A using the probabilistic view, denoted by $R \bowtie^v_A S$ is a join of R and $V_{\mathbf{B}|\mathbf{A}}(S)$: For any tuple i in R, denoted by $(\mathbf{A_i}, \bar{\mathbf{R}}_\mathbf{i})$, the join combines the tuple with the view $V_{\mathbf{B}|\mathbf{A}}(S)$ and outputs a tuple $(\mathbf{A_i}, \bar{\mathbf{R}}_\mathbf{i}, \mathbf{B})$ with the joint distribution defined as:

$$f_{\mathbf{A_i},\bar{\mathbf{R}}_\mathbf{i},\mathbf{B}}(\mathbf{a}, \bar{\mathbf{r}}, \mathbf{b}) \equiv f_{\mathbf{A_i},\bar{\mathbf{R}}_\mathbf{i}}(\mathbf{a}, \bar{\mathbf{r}}) \cdot f_{\mathbf{B}}(\mathbf{b}|\mathbf{S.A} = \mathbf{a}).$$

In this definition, the join preserves each tuple in R and extends it with the attributes in $\mathbf{B}$ from S.

Closed-form result distributions in GMMs. Given the above definitions, we seek a closed-form solution to join results under our data model. Recall that CLARO describes uncertain attributes using mixed-type distributions, where (multivariate) GMMs are used to model continuous distributions. We propose a special model for the probabilistic view, which we call *order-1 linear regression*, that allows us to obtain join result distributions also in GMMs. While the assumption of order-1 linearity may seem restrictive, it can be applied to the view at either a global or local scale, allowing implementation choices for both accuracy and efficiency.

We now consider the case when the tuples in R follow GMMs (i.e., existence probabilities are 1), while the join attributes $\mathbf{A}$ and the view attributes $\mathbf{B}$ in S are deterministic. We present a theorem that offers result distributions in GMMs for these inputs and then discuss its extensions. (The proofs are available in [27].)

Theorem 1 *Given* $R(\mathbf{A}^p, \bar{\mathbf{R}})$, $S(\mathbf{A}, \mathbf{B}, \bar{\mathbf{S}})$ *with the view* $V_{\mathbf{B}|\mathbf{A}}(S)$, *and the join* $R \bowtie^v_A S$, *assume order-1 linear regression to model the view:*

$$\mathbf{B} = \mathbf{A}\boldsymbol{\beta} + \mathbf{E}, \quad (1)$$

where $\mathbf{A}$, $\mathbf{B}$, *and* $\mathbf{E}$ *are row vectors,* $\boldsymbol{\beta}$ *is a parameter matrix, and* $\mathbf{E}$ *is normally distributed with mean* $\mathbf{0}$ *and covariance matrix* Σ_E. *For each tuple* i *in* R, *if* $(\mathbf{A_i}, \bar{\mathbf{R}}_\mathbf{i})$ *follows a GMM, a mixture of* m *components identified by the parameters* (p_c, $\boldsymbol{\mu}_c$, Σ_c), ($c = 1..m$), *then based on Definition 7, the join of tuple* i *and* $V_{\mathbf{B}|\mathbf{A}}(S)$ *yields a distribution of* $(\mathbf{A_i}, \bar{\mathbf{R}}_\mathbf{i}, \mathbf{B})$ *which also follows a GMM, a mixture of* m *components with parameters* (p_c, $\boldsymbol{\mu}'_c$, Σ'_c), ($c = 1..m$):

$$\boldsymbol{\mu}'_c = (\boldsymbol{\mu}_c, \boldsymbol{\mu}_c\bar{\boldsymbol{\beta}}), \bar{\boldsymbol{\beta}} = \begin{pmatrix} \boldsymbol{\beta} \\ \mathbf{0} \end{pmatrix}, \Sigma'_c = \begin{pmatrix} \Sigma_c & \Sigma_c\bar{\boldsymbol{\beta}} \\ \bar{\boldsymbol{\beta}}^T\Sigma_c & \Sigma_E + \bar{\boldsymbol{\beta}}^T\Sigma_c\bar{\boldsymbol{\beta}} \end{pmatrix}.$$

Theorem 1 fully characterizes each join result distribution with all of its parameters. In practice, $\boldsymbol{\beta}$ and Σ_E are unknown. They can be estimated using regression over the tuples in S, denoted by $(\mathbf{A_j}, \mathbf{B_j}, \bar{\mathbf{S}}_\mathbf{j})$, ($j = 1..n$). The least squares estimates of $\boldsymbol{\beta}$ and Σ_E are:

$$\boldsymbol{\beta} = (\tilde{\mathbf{A}}^T\tilde{\mathbf{A}})^{-1}\tilde{\mathbf{A}}^T\tilde{\mathbf{B}} \quad (2)$$

$$\Sigma_E = \tilde{\mathbf{B}}^T(\mathbf{I}_n - \tilde{\mathbf{A}}(\tilde{\mathbf{A}}^T\tilde{\mathbf{A}})^{-1}\tilde{\mathbf{A}}^T)\tilde{\mathbf{B}}/(n-k), \quad (3)$$

where $\tilde{\mathbf{A}} = (\mathbf{A}_1^T, ..., \mathbf{A}_n^T)^T$, $\tilde{\mathbf{B}} = (\mathbf{B}_1^T, ..., \mathbf{B}_n^T)^T$, $\mathbf{I}_n$ is the identity matrix of size n, and k is the size of vectors $\mathbf{A}_i$.

We next discuss three extensions for Theorem 1 that handle mixed-type tuples in R and S.

First, we consider the case when the existence of the tuples in R is uncertain. If each tuple in R follows a mixed-type distribution (p, f), where p is the tuple existence probability (TEP) and f is a GMM as before, then the join result tuple also follows a mixed-type distribution (p', f'), where $p' = p$, and f' follows a GMM as stated in the theorem.

Second, we extend to the case when the attributes in S are uncertain. We can show that when the view attributes $\mathbf{B}$ follow GMMs, there is a closed-form solution to the join result, which is also in GMMs (see [27]). When the join attributes $\mathbf{A}$ in B are uncertain, (e.g., modeled by GMMs), many smoothing techniques are available for creating the view. In this work, we opt to sample from the distributions of those uncertain attributes, collect the samples into a new input S', and apply Theorem 1 to the original input R and the new view input S'. This method allows us to obtain a closed-form solution to the join result distribution and guarantees that it is also in the form of GMMs. Note that sampling here is on a GMM, hence easy and fast. This is different from a sampling approach that constructs the full join result distribution directly from the samples—the lack of a model-based view and the need to sample over the joint space makes this approach less desirable, as we will show in §6.1.1.

Third, for the general case when the existence of tuples in S is uncertain, we also propose to sample these tuples. The difference here is that sampling may return $\perp$, indicating that a tuple does not exist. We then handle

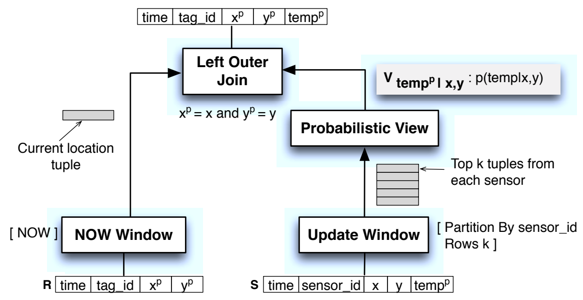

time | tag_id | xp | yp | tempp
Left Outer Join
xp = x and yp = y
V tempp | x,y : p(temp|x,y)
Probabilistic View
Current location tuple
Top k tuples from each sensor
[NOW]
NOW Window
Update Window
[Partition By sensor_id Rows k]
R time | tag_id | xp | yp
S time | sensor_id | x | y | tempp

Fig. 5 Query plan for the join using a probabilistic view (Q1).

this by omitting these samples and sampling for more data points to run regression as above.

Evaluation Techniques. The query plan for Q1 with a join and a probabilistic view is depicted in Fig. 5. The plan first applies the query-specified windows to the inputs: A *Now* window feeds each location tuple as the left input to the join. An *Update* window, [Partition By sensor_id Rows k], contains the most recent k temperature tuples from each sensor; the probabilistic view $V_{temp|x,y}$ is maintained over the update window and is the right input to the join. The join extends each location tuple with the temperature presented by the view, and returns a joint distribution.

Given the closed-form result distribution, the main implementation issue is the view construction using regression. While recent research has applied regression to build models and views for sensor data [12,9], our work differs by exploring the tradeoffs of using order-1 linear regression at a global versus local scale.

Global regression applies regression equations (Eq. 2 and Eq. 3) to all S tuples in the current update window to construct the view. As proposed in recent work [12], the view can be maintained incrementally by updating intermediate matrices for $\boldsymbol{\beta}$, e.g., $\tilde{\mathbf{A}}^T\tilde{\mathbf{A}}$ and $\tilde{\mathbf{A}}^T\tilde{\mathbf{B}}$ in Eq. 2. Then, when an R tuple arrives, the view is refreshed by completing the matrix operations for $\boldsymbol{\beta}$ and Σ_E. A fundamental limitation of global regression is that the order-1 linear assumption may not hold over the entire view. For Q1, the temperature may not be a linear function of the location but rather, say, a quadratic function. In that case, global regression may be inaccurate when its assumption fails to hold.

Local regression is motivated by the statistical theory that a smooth function can be approximated by a low degree polynomial, e.g., a linear function, in the neighborhood of any point. We design a local regression method as follows: Given each R tuple following a GMM, use the means and the variances of the components of the GMM to define a sufficient *local regression region* (LRR). As a simple example, the LRR for an R tuple that follows $\mathcal{N}(\mu, \sigma^2)$ can be $[\mu - m\sigma,\ \mu + m\sigma]$ with $m \geq 2$. Then, retrieve the subset of S tuples that reside in the LRR and apply regression to these tuples.

A key advantage of this method is that it does not require the assumption of global linearity, hence allowing more accurate view construction. However, a very small set of tuples in the LRR may not have enough data points to achieve the accuracy (which is a data problem, not a model problem). When this problem occurs, we can collect more data points by adjusting the LRR appropriately (as shown experimentally in §6.1.1). Computation-wise, regression is applied to a small set of tuples, hence with a low cost.

4.3.2 Joins using Cross-Product

We now summarize the main result for joins using cross-product, which is straightforward under our data model. It can be applied in the scenarios where a join is modeled using a cross-product followed by a selection [23]—in this case, only inequality predicates are allowed to avoid join results of zero probability. An example is a query in the RFID domain that detects the co-location of peanut and peanut-free food: it compares every pair of objects for proximity in location. Our system supports such joins with closed-form result distributions under the mixed-type model. More specifically, if two join attributes are uncertain and follow mixed-type distributions (p_1, f_1) and (p_2, f_2), then the join result follows a mixed-type (p, f) where $p = p_1 \cdot p_2$, $f = f_1 \cdot f_2$. If f_1 and f_2 are GMMs, then f is a multivariate GMM. Any subsequent selection for a specific region of f can be performed as detailed in §4.1.

4.4 An Evaluation Framework for Aggregation

In this section, we address aggregation of continuous-valued uncertain tuples. The nature of computation for aggregates such as `sum` and `avg` is multivariate integration, which is inherently expensive. Fig. 6 shows an example of `avg` of continuous random varibles. A state-of-the-art approach is *integral-based* [5], which integrates two variables at a time, resulting in the use of n-1 integrals to aggregate n variables. An alternative sampling-based approach [11,23] generates samples from the input distributions and computes aggregates from these samples. However, it is hard to know the right number of samples to exploit the tradeoff between accuracy and performance, as we will show in the experiments. Another approach is to discretize continuous distributions and use existing algorithms for discrete distributions to compute `sum` and `avg` [16]. This has a time complexity $O(nD^3)$, where n is the number of tuples and D is the domain size of each tuple, hence becoming inefficient for large domains like what continuous variables require.

Our work departs from existing approaches by exploring advanced statistical theory to obtain *exact result distributions*, whenever possible, while completely eliminating the use of integrals. When the exact result distributions are complex, we provide an efficient *approximation* technique to simplify their formulas while satisfying given accuracy requirements. In other cases when it is hard to obtain the closed-form solutions, we seek *approximation* algorithms to directly compute the distribution of aggregates with bounded errors.

We next present an evaluation framework including metrics and objectives that we will use in our approximation algorithms for aggregation.

A. Metrics. We introduce two common distance metrics to approximate the distributions of aggregates.

The *point-based Variation Distance (VD)*, similar in idea to the VD in [11], is used as a distance metric for two continuous distributions.

Definition 8 Given two probability density functions (pdf's) $f(x)$ and $g(x)$, the VD is defined as:

$$\mathrm{VD}(f,g) = \frac{1}{2}\int_{\mathbb{R}} |f(x) - g(x)|dx.$$

The constant 1/2 ensures that VD is in [0,1].

Another distance metric based on a standard measure in statistics, is the *Kolmogorov-Smirnov (KS) distance.*

Definition 9 Given two one-dimensional cumulative distribution functions (CDF's) $F, G : \mathbb{R} \to [0,1]$, the KS distance is defined as: $\mathrm{KS}(F,G) = \sup_x |F(x) - G(x)|$.

The following proposition states the relationship between the two distance metrics. (The proof is in [27].)

Proposition 1 *The KS distance of two CDF's,* KS*(F, G) and the variation distance of the corresponding pdf's,* VD(f,g)*, satisfy* KS*(F, G)* $\le$ VD(f,g)*. In some cases,* KS*(F, G) can be arbitrarily smaller than* VD(f,g).

Since KS$(F, G) \le \mathrm{VD}(f,g)$ always holds, any approximation algorithm that satisfies the error bound ϵ using the VD metric also has a KS distance bounded above by ϵ. Therefore, approximation algorithms using the VD metric can be readily included in an evaluation framework that employs the KS distance as the metric.

In the following sections, we derive approximation algorithms using the KS distance. We include the variation distance to compare our techniques with a state-of-the-art technique that uses this metric [11].

B. Approximation Objective. We next state the definition of (ϵ, δ) approximation using KS distance as the metric. (The definition using VD follows directly.)

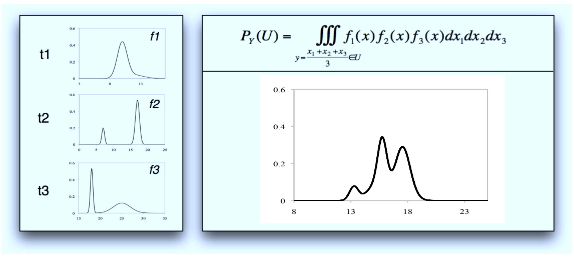


Fig. 6 Aggregation of continuous random variables.

Definition 10 A (randomized) algorithm computes an (ϵ, δ) approximation if the KS distance between the approximate distribution and its corresponding exact distribution is at most ϵ with probability $1 - \delta$.

$\delta = 0$ corresponds to deterministic algorithms.

4.5 Aggregations under GMM (TEP=1)

We now address aggregation of uncertain tuples whose existence is certain, i.e., the existence probabilities are 1, and the tuples follow Gaussian mixture models. This includes our input model, hence a common case in our applications. We focus on `sum` and `avg` because they are crucial to our target applications but have not been sufficiently addressed in the literature.[2]

4.5.1 A Basic Algorithm

We next introduce *characteristic functions* and describe a basic algorithm to derive the result distribution for `sum` of a set of independent tuples. When tuples exist with probabilities 1, `count` is deterministic; thus, the modification to `avg` is straightforward, hence omitted.

In probability theory, characteristic functions (CFs) [4] are used to "characterize" distributions. Specifically, the CF of a random variable U is defined as:

$$\Phi_U(t) = E[e^{iUt}], \tag{4}$$

where E denotes the expected value and i is the complex number $\sqrt{-1}$. The pdf of U then can be obtained by the inverse transformation of the CF:

$$f_U(x) = \frac{1}{2\pi}\int_{-\infty}^{+\infty} e^{-itx}\Phi_U(t)dt. \tag{5}$$

Now let us consider `sum`(A), with the attribute A in n tuples modeled using random variables X_1, ..., X_n. Let $U = X_1 + X_2 + ... + X_n$. The CF of U is:

$$\begin{aligned}\Phi_U(t) &= E[e^{iUt}] = E[e^{i(X_1+X_2+...+X_n)t}] \\ &= \Phi_{X_1(t)}\Phi_{X_2(t)}...\Phi_{X_n(t)} \end{aligned} \tag{6}$$

[2] Our technique for `min` and `max` is similar to that in [5], hence omitted in this paper.

That is, the CF of U can be written as the product of the CFs of the input tuples based on the independence assumption. This suggests a simple algorithm for `sum`: (1) Get the CF of each input tuple and take the product of these functions according to Eq. 6. (2) For a given value x, apply the inverse transformation to yield $f_U(x)$ according to Eq. 5. In particular, we call the inverse transformation in the second step a *parameterized integral* because it takes an argument x.

In the context of Gaussian mixture models (GMMs), the CFs can be expressed in closed form. For example, for a Gaussian mixture of two components:

$$f(x) = p_1\mathcal{N}(\mu_1, \sigma_1^2) + p_2\mathcal{N}(\mu_2, \sigma_2^2),$$

its CF can be directly obtained as:

$$\Phi_X(t) = p_1 e^{i\mu_1 t - \frac{1}{2}\sigma_1^2 t^2} + p_2 e^{i\mu_2 t - \frac{1}{2}\sigma_2^2 t^2}.$$

Thus, Step 1 of the above algorithm does not involve any integration. The only integral required is the one for inverse transformation in Step 2. This analysis holds for all common distributions whose characteristic functions are known. This gives a boost in performance compared to the two-variable convolution method, which requires n-1 parameterized integrals [5].

The main drawback of this approach is that the formula of the result distribution involves an unresolved parameterized integral. To get sufficient knowledge of the result distribution (e.g., calculating its mean and variance), one needs to repeat the inverse transformation for a large number of points. This is expensive when the integration has no closed-form solution and needs resorting to numerical solutions. Moreover, it is unknown if the result distribution is a GMM.

4.5.2 Exact Derivation of Result Distributions

The discussion in the previous section motivated us to seek a solution without using integration. For GMMs, it turns out that we can obtain the **closed-form** solution to the inverse transformation. In addition, when input tuples are GMMs and independent, the result of `sum` over those tuples is also a GMM that can be directly obtained from the input tuples.

Theorem 2 *Let each X_i $(i = 1..n)$ be a mixture of m_i components identified by the parameters $(p_{i_j}, \mu_{i_j}, \sigma_{i_j}), (j = 1..m_i)$. The result distribution for $U = \sum_{i=1}^n X_i$ is a Gaussian mixture of $\prod_{i=1}^n m_i$ components, each of which corresponds to a unique combination that takes one component from each input Gaussian mixture $\{j_i\}(i = 1..n, j = 1..m_i)$ and is identified by (p_k, μ_k, σ_k):*

$$p_k = \prod_{i=1}^n p_{j_i},\ \ \mu_k = \sum_{i=1}^n \mu_{j_i},\ \ \sigma_k = \sqrt{\sum_{i=1}^n \sigma_{j_i}^2}. \qquad (7)$$

This theorem can be proved by manipulation of the inverse transformation formula. The result subsumes the well-known linear property of Gaussian distributions.

This technique gives an exact solution. The computation involved is to enumerate and compute all components of the result Gaussian mixture, which grows exponentially in the number of tuples, n. Hence, this technique is inefficient when n is large.

4.5.3 Approximation of Result Distributions

We next propose to approximate the exact result distribution by performing function fitting in the Characteristic Function (CF) space. This is based on the property that the CF of `sum` can be compactly represented as a product of n individual CFs (Eq. 6), rather than an exponential number of components (Eq. 7). Our goal is to find some Gaussian mixture distribution whose CF best fits this product function.

Algorithm 1 Sketch of the **CF fitting** algorithm for approximation

```
1: Obtain the expression of the CF of the sum, $\Phi_{\texttt{sum}}(t) = \prod_{i=1}^n \Phi_{X_i}(t)$. This is a complex function.
2: Take $P$ points $\{t_i\}, (i = 1..P)$ from the domain of $\Phi_{\texttt{sum}}(t)$, and compute $\{\Phi_{\texttt{sum}}(t_i)\}, (i = 1..P)$.
3: Start with $K = 1$. Consider a Gaussian mixture of $K$ components. The corresponding CF is $\overline{\Phi}(t)$.
4: Run least squares fitting to minimize:
   $\sum_{i=1}^P \left[(Re(\overline{\Phi}(t_i) - \Phi_{\texttt{sum}}(t_i)))^2 + (Im(\overline{\Phi}(t_i) - \Phi_{\texttt{sum}}(t_i)))^2\right]$.
5: Get the fitting residual. If this is smaller than a threshold $\epsilon$, return the fitted Gaussian mixture. Otherwise, increase $K$ by one (by default) and go back to step 3.
```

We devise an approximation algorithm, named **Characteristic Function (CF) fitting**, which is sketched in Algorithm 1. The algorithm starts with one component Gaussian mixture, running the least squares fitting. If the fitting residual is below a threshold, it returns the fitted parameters; otherwise it increases the number of components and repeats fitting. Note that the objective function for fitting contains both *real* and *imaginary* parts, since the CFs are complex functions and both parts contribute to the result pdf via inverse transformation. This algorithm eliminates the exponential cost as for exact derivation, and incurs a cost polynomial in the number of tuples n, the number of components per tuple (Steps 1 and 2), and the size of the result distribution K (Steps 3 and 4).

Optimizations. We further employ a suite of optimizations based on statistical theory to improve performance and accuracy. The first optimization regards the choice of an appropriate range in the domain of the CF $\Phi_{\texttt{sum}}(t)$ for fitting. The formula of $\Phi_{\texttt{sum}}(t)$ indicates that it approaches 0 fast as t moves from the center 0.

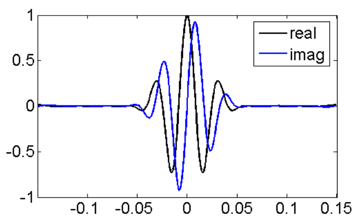


Fig. 7 Example characteristic function for `sum` of 10 tuples.

Fig. 7 shows an example CF for `sum` of 10 tuples, with both real and imaginary parts. Given this observation, we set the range for fitting to be a small region centered around 0 so that points taken can better capture the shape of the function to be fitted.

The second optimization concerns the initial guess of the parameters of a K-component Gaussian mixture. Due to the oscillating behavior of the CF, the fitting result is quite sensitive to the initial guess and can get stuck in local optima. We use Theorem 2 to precompute a small number of result components whose means are spread out and use them as the initial guess for fitting.

Test Condition for Convergence. We determine whether the fitting result satisfies a KS requirement, ϵ, by approximately computing the distance between the the approximate distribution and the true distribution. To do so, we approximate the inverse transformation to obtain the CDF's by using the points in fitting to estimate the integrals. Specifically, we check if the following condition holds to stop fitting.

$$\sum_{i=1}^{P}[Re(\overline{\Phi}(t_i)-\Phi_{\texttt{sum}}(t_i))+Im((\overline{\Phi}(t_i)-\Phi_{\texttt{sum}}(t_i))]\frac{\Delta t}{t_i}\le\epsilon$$

where t_i ($i = 1..P$) are points used in fitting. This holds because for points outside this range, the values of the CFs are close to 0. A similar condition can also be derived if VD is used as the metric.

Relation to the Central Limit Theorem. The Central Limit Theorem (CLT) is a special case of our CF fitting algorithm. It states that the sum of a sufficiently large number of independent random variables is normally distributed [4]. This gives an asymptotic result but our algorithm dynamically determines when this result can apply. For example, a weather monitoring system sometimes requires a small number of stream segments to be averaged, for which our algorithm determines that the CLT does not apply, whereas when the number of tuples is sufficiently large (e.g., greater than 20), the result distribution starts to become a single smooth Gaussian.

4.5.4 Hybrid Solution

The two algorithms for aggregation, exact derivation and CF fitting, can be combined into a **hybrid** solution to exploit their advantages. When the number of tuples is small, we use exact derivation since it is fast and its formula is not complex. When the number of tuples is large enough, we switch to CF fitting. This way, we take the advantage of each algorithm in the range it performs best. We observe that the switching points among the two mainly depend on the number of tuples and less so on other data characteristics, as shown in §6.1.2.

4.6 Aggregations under Mixed-type Model (TEP $\le$ 1)

We have considered aggregation when the existence of tuples is certain. However, in the presence of conditioning operations, e.g., selections, the existence probabilities of tuples become less than 1, precluding the above closed-form solution and its approximation for aggregation. In this section, we seek to directly devise approximation algorithms to compute aggregates of conditioned tuples.

4.6.1 Approximate Representation for CDFs

We first extend our approximation framework to include a new approximate representation for approximation algorithms to compute aggregates. We employ cumulative distribution functions (CDFs) to approximate distributions of aggregates due to two desirable properties of a CDF: (1) it is a non-decreasing function ranging from 0 to 1, and (2) it can be defined at any point in the real domain; e.g., the CDF of a discrete random variable can be represented as a step function. We employ two specific CDF functions, StepCDF and LinCDF, for approximate representations.

Definition 11 Given a set of points $P = \{(x_1, y_1), \ldots, (x_k, y_k)\}$ where $x_1 \le x_2 \le \ldots \le x_k$ and $0 \le y_1 \le \ldots \le y_k = 1$, 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:

$$\text{StepCDF}_P(x) = \begin{cases} 0 & \text{if } x < x_1 \\ y_i & \text{if } x_i \le x < x_{i+1} \\ 1 & \text{if } x \ge x_k \end{cases}$$

$$\text{LinCDF}_P(x) = \begin{cases} 0 & \text{if } x < x_1 \\ y_i + \frac{x-x_i}{x_{i+1}-x_i}(y_{i+1}-y_i) & \text{if } x_i \le x < x_{i+1} \\ 1 & \text{if } x \ge x_k \end{cases}$$

Objectives. Using these approximate representations, we devise algorithms that construct approximate distributions of aggregates over uncertain data. If F_t^A is the cumulative distribution of aggregate $A_t = A(Y_1, \ldots, Y_t)$, where Y_i's are independent, we seek an algorithm that maintains an approximation $\tilde{F}_t^A$ incrementally as data arrives while satisfying a given error bound.

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`, an aggregate exists if one of the input tuples exists; hence, $p = 1 - \prod_i (1 - p_i)$. Therefore, below we focus on algorithms that compute (ϵ, δ) approximate distributions given that the aggregates exist.

4.6.2 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_i is the random variable corresponding to the i-th tuple, and let F_t^M be the CDF of M_t. Recall that our mixed-type data model incorporates both continuous and discrete variables that arise from tuple existence uncertainty. To provide a uniform solution, we first consider discrete variables and later extend to the continuous case. We assume that each Y_i takes λ values from a finite universe of size $\mathbb{U}$, without loss of generality, $[1, n]$, or shortly $[n]$.

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 $F_t^M(x) = \prod_{i \in [t]} \mathbb{P}[Y_i \leq x]$. Therefore, it is sufficient to maintain $F_t^M(x)$ for every x in the universe and update it when a new tuple arrives. This computes the exact distribution of `max` with the update cost per tuple $O(\mathbb{U})$, hence inefficient for stream processing. Probabilistic databases compute the distribution of `max` based on the *extensional semantics* [7], with the high total cost of $O(t\mathbb{U})$ for a relation of t tuples; further, this is not an incremental algorithm.

An natural attempt given the above observation is to approximate the distribution of `max` by maintaining its values at a set of points. However, it is impossible to choose this set of points in advance to get a good approximation. 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 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. 8(a). This yields a StepCDF defined over the point set $\{a_1, b_1, a_2, b_2, \ldots\}$.

The algorithm has the following two invariants. At any point, given any interval $[a_i, b_i]$ and a constant parameter ϵ' (see Theorem 3 on how to set ϵ' as a function of the accuracy requirement ϵ) , we have:

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

Invariant 1 guarantees that the estimates of the two ends of an interval are close, so the 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 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.

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. 8b & c). If after updating, $c'_b < \epsilon$, discard the interval. Note that the ratio between the updated 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. 8d). 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 intervals is shifted by a factor $\sqrt{1 + \epsilon'}$. Fig. 8e illustrates this step.

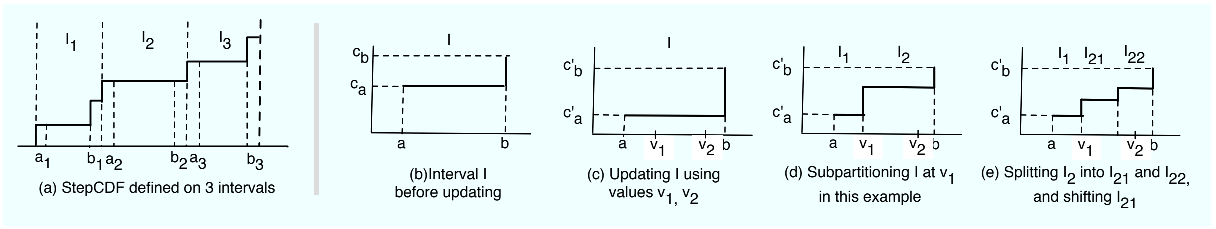

I1
I2
I3
I
a1
b1 a2
b2 a3
b3
(a) StepCDF defined on 3 intervals
cb
ca
a
b
(b)Interval I before updating
c'b
c'a
a v1 v2 b
(c) Updating I using values v1, v2
I1 I2
(d) Subpartitioning I at v1 in this example
I1 I21 I22
(e) Splitting I2 into I21 and I22, and shifting I21

Fig. 8 StepCDF and illustration of the basic steps of the MAX algorithm.

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 available in [28].

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

Lemma 3 *The maximum generation of an interval is* $\log \mathbb{U}$.

Theorem 3 *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 §6.2.1).

4.6.3 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, the Central Limit Theorem (CLT) states that the distribution of S_t tends towards a Gaussian distribution as t goes to infinity. But for many applications, this asymptotic result cannot apply. 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 [7]. We instead present a deterministic algorithm that computes the approximate distribution of `sum` more efficiently.

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 $\mathrm{Q}(\epsilon)$, such that:

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

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

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

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. 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 is often more accurate.)

Our algorithm processes each new tuple in two conceptual steps *Update* and *Simplify.* In update, we combine our approximation for F_{t-1}^S with Y_t to produce an intermediate approximation F for F_t^S:

$$F(x) = \sum_{v \in V_t} \mathrm{LinCDF}_{P_{t-1}}(x - v)\mathbb{P}[Y_t = v] \tag{8}$$

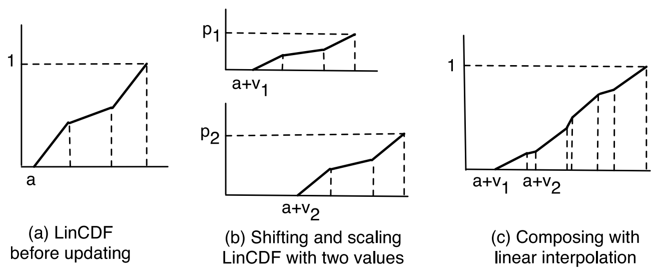

1
a
(a) LinCDF before updating
p1
a+v1
p2
a+v2
(b) Shifting and scaling LinCDF with two values
1
a+v1 a+v2
(c) Composing with linear interpolation

Fig. 9 Updating step of the SUM algorithm.

That is, 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 by using linear interpolation for the LinCDF. See Fig. 9 for an illustration of this step. Now F contains a set of λk points, which we call $\bar{P}_t$. Next, simplify F to ensure efficiency in later processing while meeting the error bound ϵ' provisioned for this tuple. (Theorem 4 shows how to set ϵ'.) To do this, we compute the k quantiles of F and return LinCDF_{P_t} where $P_t = \{(F^{-1}(i\epsilon'), i\epsilon') : 1 \leq i \leq k\}$. Performing these steps sequentially may be inefficient since we compute λk points but then only use 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 $F(x_a) \leq i\epsilon' \leq F(x_b)$. This results in the following theorem.

Theorem 4 *We can maintain an $(\epsilon, 0)$ approximation for F_t^S 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 for 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 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*, which has more points near the two ends of a distribution, gives a better approximation. However, to meet a KS requirement, we need more biased quantiles than uniform quantiles theoretically. We hence propose to use both sets of quantiles, named mixed quantiles. (3) *Central Limit Theorem.* For sufficiently large t, the distribution of F_t^S is approximately 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 according to the Berry-Esseen theorem [8].

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 (at most) $\epsilon_1 = 1/\lambda$. We can show that if the KS error incurred when adding this tuple to `sum` is ϵ_2, the total error from this tuple is bounded by $\epsilon_1 + \epsilon_2$ [28]. We next discuss on how to set ϵ_1 and ϵ_2. Recall that the SUM algorithm with optimizations computes λk points and then adaptively chooses a subset of size k' that satisfies the KS error of ϵ_2, where $k' \leq 1/\epsilon_2$. Hence the cost is also proportional to $O(\lambda k') = O(1/\epsilon_1 \cdot 1/\epsilon_2)$. In practice, due to the use of mixed quantiles, k' is often smaller than $1/\epsilon_2$, especially when the distribution becomes smooth. This gives an incentive to set $\epsilon_1 > \epsilon_2$ as we validate empirically.

4.7 Bounded-Error Monte-Carlo Simulation

We next present a general randomized algorithm based on Monte-Carlo simulation to compute aggregates. In contrast to prior work, we establish accuracy guarantees in our evaluation framework. Given a deterministic stream $\langle y_1, \ldots, y_t \rangle$ and an aggregate A for which there exists an efficient stream algorithm Φ to compute $A(y_1, \ldots, y_t)$, the algorithm Φ^* to compute an (ϵ, δ) approximate distribution for an uncertain stream 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 $\langle y_1^i, \ldots, y_t^i \rangle$, $i = 1..m$, and compute $a_i = A(y_1^i, \ldots, y_t^i)$.
– Return $\tilde{F}_t^A(x) = \frac{1}{m}\sum_{i \in [m]} 1_{[a_i, \infty)}(x)$.

Theorem 5 *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 this theorem follows from the Dvoretsky-Kiefer-Wolfowitz theorem in statistics.

We see that this theorem directly applies to aggregates such as `sum`, `count`, `min`, `max`, and `avg`. This theorem subsumes existing work based on Monte Carlo sampling [11,13,23], since it can determine the number of samples sufficient for a given accuracy requirement, in contrast to taking the number of samples as an input parameter. A recent work [21] also uses Monte Carlo simulation to estimate the probability of an aggregate predicate in the `having` clause, but does not compute the full distribution of an aggregate.

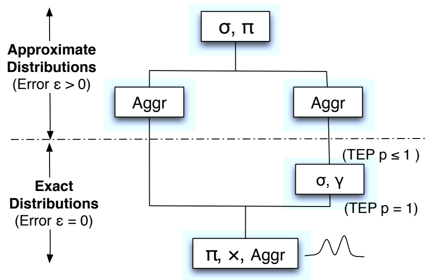

Approximate Distributions (Error ε > 0)
Exact Distributions (Error ε = 0)
σ, π
Aggr
Aggr
(TEP p ≤ 1)
σ, γ
(TEP p = 1)
π, ×, Aggr

Fig. 10 Query plan arrangement in the mixed-type model.

5 Query Planning under Mixed-type Models

In this section, we examine query planning that considers how to handle errors due to the mix of different operations in the context of a complete query.

5.1 Arranging Operators in a Query Plan

We first discuss the arrangement of relational operators in a query plan using our data model. For queries that involve only joins, projections, and aggregates, we have shown that for continuous uncertain attributes modeled by Gaussian mixture models (GMMs), there are exact, closed-form solutions for the result distributions in Section 4. When the above queries are extended with selections, placing selections before joins, projections, and aggregates in a query plan can result in conditioned (or precisely, mixed-type) distributions, hence not in GMMs any more. The implications of this on other relational operations depend on *commutativity.* As in traditional databases, projections and joins commute with selections [19]. 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, hence these solutions cannot be applied to aggregates after selections. Similarly, group-bys condition distributions when evaluating the groups, thus precluding GMM-based solutions for subsequent aggregates. Then, we can resort to the approximations proposed in Section 4 to compute the distributions of aggregates.

The above discussion suggests the arrangement of relational operators in a query plan, as depicted in Fig. 10, where the operators contained in the same box can be arranged in any order. In particular, the bottom part of the query plan computes exact distributions, using the exact algorithms and the definition of conditioning operations, i.e., selections, group-bys (denoted as γ). Errors start to occur at the aggregation operator where an approximation algorithm is used, (see §4.5.3 and §4.6), and will propagate to the subsequent operators.

5.2 Query Planning

We now consider query planning that computes approximate answers with bounded errors for complex queries. Our work supports a `Select-From-Where-Group by-Having` block. We can compute multiple aggregates that are independent by invoking the approximation algorithms separately.[3] More specifically, the cases that we support include: (1) apply selection or group-by on some uncertain attributes and then compute a single aggregate, (2) compute multiple aggregates on independent attributes when tuple existence is certain. In both cases, the aggregates computed can be used in `Having` or returned in `Select`. The examples of the first case are Q2 and Q3, where group-bys introduce tuple existence probabilities (TEPs) and the uncertain attributes become correlated through these TEPs. Then, we can compute the marginal distribution for a single aggregate. The second case includes not having `Group by` or having `Group by` on deterministic attributes (e.g., Q4 below) since this still retains TEPs equal to 1. In this case, the aggregates of the independent attributes are independent and can be computed using our algorithms.

At mentioned above, errors start to occur in the first aggregate computed using an approximation algorithm. These errors can then propagate to the subsequent operations performed on the derived aggregate attributes. To quantify errors of intermediate and final query results, we extend our approximation framework to account for errors associated with both the attribute distributions and the tuple existence probability.

Extended Approximation Metric. We first extend our KS metric to a general case when both the TEP and uncertain attributes in a mixed-type tuple are approximate. The extension, adopted from the KS definition for multi-dimensional CDF's [17], considers all complementary distribution functions. We denote an *ordering* of random variables, $\mathbf{X} = (X_1, X_2, ..., X_n)$, to be a vector $\mathbf{o} = (o_1, o_2, ..., o_n)$, where $o_i = \{\leq, \geq\}$. Given a constant vector, $\mathbf{x} = (x_1, x_2, ..., x_n)$, $\mathbb{P}_{\mathbf{X}}[\langle \mathbf{o}, \mathbf{x} \rangle] = \mathbb{P}[\bigwedge_i (X_i \ o_i \ x_i)]$. $\mathbb{P}_{\mathbf{X}}[\langle \mathbf{o}, \mathbf{x} \rangle]$ can be computed via integration of the joint density function (pdf) of $\mathbf{X}$.

Definition 12 Let $G{=}(p, f)$ and $\tilde{G}{=}(\tilde{p}, \tilde{f})$ be two mixed-type distributions of $\mathbf{X}$ and $\tilde{\mathbf{X}}$, where each contains n attributes, respectively. The *mixed-type* KS, termed KSM, between G and $\tilde{G}$ is defined as:

$$\begin{aligned}&\mathrm{KSM}(G, \tilde{G})\\ &= \max(|p - \tilde{p}|, \max_{\mathbf{o}}(\sup_{\mathbf{x}} |p \cdot \mathbb{P}_{\mathbf{X}}[\langle \mathbf{o}, \mathbf{x} \rangle] - \tilde{p} \cdot \mathbb{P}_{\tilde{\mathbf{X}}}[\langle \mathbf{o}, \mathbf{x} \rangle]|)).\end{aligned}$$

[3] Computing correlated aggregates is a hard problem and left to future work.

This definition considers all of the 2^n orderings $\mathbf{o}$ of n variables. Since KSM computes the maximum of the differences between the probabilities that the variables are in any given range, it ensures symmetric results for range predicates (e.g., for $\leq$, $\geq$). For the two classes of queries discussed above, this general definition can be reduced to:

Remark 1 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 a single attribute. The KSM between G and $\tilde{G}$ becomes:

$$\begin{aligned}\mathrm{KSM}(G, \tilde{G}) = \max(&|p - \tilde{p}|, \ \sup_x |p \cdot F(x) - \tilde{p} \cdot \tilde{F}(x)|, \\ &\sup_x |p \cdot (1 - F(x)) - \tilde{p} \cdot (1 - \tilde{F}(x))|).\end{aligned}$$

For example, if G and $\tilde{G}$ are the true and approximate distributions of an attribute X, $\mathrm{KSM}(G, \tilde{G}) = \epsilon$ means that all quantities such as $\mathbb{P}[x \neq \perp]$, $\mathbb{P}[x \neq \perp \wedge x \leq 5]$, and $\mathbb{P}[x \neq \perp \wedge x \geq 5]$, when computed using G or $\tilde{G}$, will not differ by more than ϵ.

In the second case, where the TEP is exact and equal to 1, and the attributes X_i are independent, the KSM can be rewritten as follows.

Remark 2 Let G and $\tilde{G}$ be the multivariate distributions of $\mathbf{X}$ and $\tilde{\mathbf{X}}$, where each contains n independent attributes. The KSM between G and $\tilde{G}$ is:

$$\begin{aligned}\mathrm{KSM}(G, \tilde{G}) &= \max_{\mathbf{o}}(\sup_{\mathbf{x}} |\mathbb{P}_{\mathbf{X}}[\langle \mathbf{o}, \mathbf{x} \rangle] - \mathbb{P}_{\tilde{\mathbf{X}}}[\langle \mathbf{o}, \mathbf{x} \rangle]|) \\ &= \max_{\mathbf{o}} \sup_{\mathbf{x}} |\prod_i \mathbb{P}[X_i \ o_i \ x_i] - \prod_i \mathbb{P}\left[\tilde{X}_i \ o_i \ x_i\right]|\end{aligned}$$

The following proposition characterizes the KSM of the joint distribution in terms of individual KS's. Its proof is shown in [27], due to space constraints.

Proposition 2 *Let $G{=}(p, f)$ and $\tilde{G}{=}(\tilde{p}, \tilde{f})$ be two mixed-type distributions of attributes $\boldsymbol{X} = (X_1, X_2, ..., X_n)$. If $p = \tilde{p} = 1$, X_i's are independent of each other, and each X_i is bounded with a* KS *error ϵ_i,* $\mathrm{KSM}(G, \tilde{G}) \leq \sum_i \epsilon_i$.

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 as follows.

Definition 13 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 and the KSM between any tuple in $\tilde{S}$ and its corresponding tuple in S is at most ϵ with probability $1 - \delta$.

Query Planning: Error Propagation. 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 begin at the first aggregation that applies (ϵ, δ)-approximation as proposed in Section 4. For post-aggregate operations, the earlier approximation errors now affect the estimates of both the tuple existence probability and distributions of derived aggregate attributes. Below, we focus on selection and projection as post-aggregation operators.

Selection. We quantify the approximation errors propagated through selections, e.g., in the `Having` clause in the next proposition. The proof is available in [28].

Proposition 3 *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.*

When selections are applied for multiple independent aggregates, the above proposition applies for each selection independently. Note that in this case, the TEP would be factorized into these attributes, and its KSM error is bounded by the sum of KSM error of the independent attributes (this is a simple generalization of Proposition 2 that uses KSM instead of KS).

Projection. Projection does not change the tuple existence probability. That is, if a derived attribute whose existence probability is approximate is projected out, its error is transferred to the existence probability of the result tuple; hence, the KSM of the tuple does not change. This also holds for the case where multiple derived attributes are projected out, since one attribute can be projected out at a time. A special, but trivial, case is when an approximate derived attribute having an exact existence probability is projected out, the KSM error of the result tuple is reduced by the KSM error of that attribute, as indicated by Proposition 2.

Query Planning: A Top-down Approach. In query planning, we start from base tuples, assign a variable indicating the error incurred by each operation, and combine these variables into a formula using the results from the above bottom-up analysis. Then given a target error bound ϵ for the entire query (and error formula), we traverse the query plan top-down, assign an error bound to each variable to satisfy the target error bound. We next consider query planning for a set of queries that covers all cases that CLARO supports.

Computing a single aggregate. Revisit query Q2, whose query plan is illustrated in Fig. 11a. The query plan first performs the group-by operation, which com-

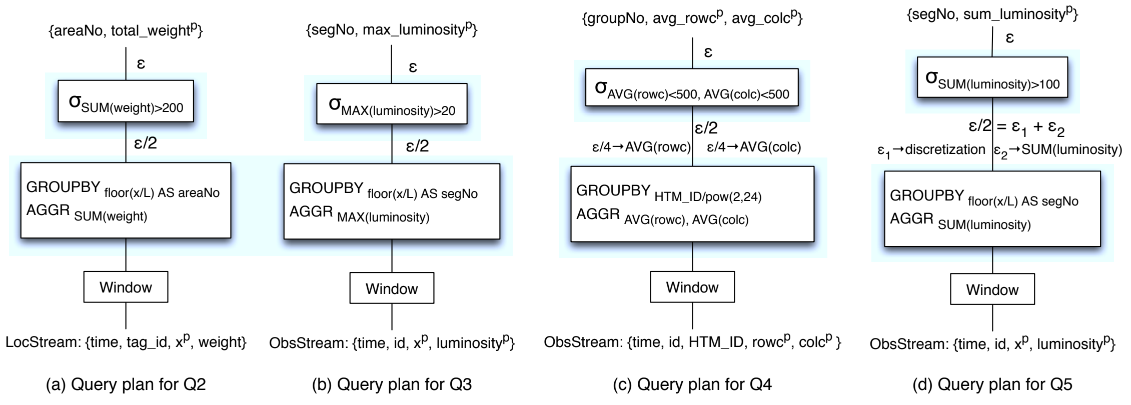

{areaNo, total_weight^p}
ε
σ SUM(weight)>200
ε/2
GROUPBY floor(x/L) AS areaNo
AGGR SUM(weight)
Window
LocStream: {time, tag_id, x^p, weight}
(a) Query plan for Q2
{segNo, max_luminosity^p}
ε
σ MAX(luminosity)>20
ε/2
GROUPBY floor(x/L) AS segNo
AGGR MAX(luminosity)
Window
ObsStream: {time, id, x^p, luminosity^p}
(b) Query plan for Q3
{groupNo, avg_rowc^p, avg_colc^p}
ε
σ AVG(rowc)<500, AVG(colc)<500
ε/2
ε/4→AVG(rowc)
ε/4→AVG(colc)
GROUPBY HTM_ID/pow(2,24)
AGGR AVG(rowc), AVG(colc)
Window
ObsStream: {time, id, HTM_ID, rowc^p, colc^p}
(c) Query plan for Q4
{segNo, sum_luminosity^p}
ε
σ SUM(luminosity)>100
ε/2 = ε1 + ε2
ε1→discretization
ε2→SUM(luminosity)
GROUPBY floor(x/L) AS segNo
AGGR SUM(luminosity)
Window
ObsStream: {time, id, x^p, luminosity^p}
(d) Query plan for Q5

Fig. 11 Query planning for Q2-Q5

putes the tuple existence probability of an object in each group, and then computes `sum` of weight for each group using the SUM approximation algorithm, with error ϵ_1. After that, the selection, $\texttt{sum}(weight) > 200$, is applied to each group. Proposition 3 bounds the error of the selection by $2\epsilon_1$. Therefore, given the target error bound, ϵ, the approximate `sum` should have an error bound $\epsilon_1 = \epsilon/2$.

Query planning for Q3, except for computing `max`, is similar, as shown in Fig. 11b. These two queries are examples of case (1) we support.

Computing multiple independent aggregates. Query Q4 below is a modified query taken from the Sloan digital sky survey (SDSS) example queries. Q4 groups object into HTM buckets, a deterministic attribute, and computes two independent averages of *rowc* and *colc* and returns the groups when these averages are in a certain range. This corresponds to case (2) above.

```
Q4: Select   HTM_ID/power(2,24), AVG(rowc), AVG(colc)
    From     Galaxy
    Group by HTM_ID/power(2,24)
    Having   AVG(rowc) ≤ 500 and AVG(colc) ≤ 500
```

If the target accuracy requirements is ϵ, we can assign an error bound of $\epsilon/2$ to each average according to Proposition 2. Then due to the effect of selection, the error bound assigned to each average before selection is set to $\epsilon/4$. Note that error provisioning remains the same if we use other aggregates than `avg`.

Discretization of continuous distributions. Recall that Q3 computes `max` of *luminosity*, a continuous attribute. Due to the partitioning scheme of the MAX algorithm, we do not need to discretize the distribution of the input tuples in advance. Now consider Q5, a slightly different version of Q3, that computes $\texttt{sum}(luminosity)$. This query is to detect regions with high cumulative luminosity. Due to the effect of selection after `sum`, given a target error bound ϵ, the approximation of `sum` can have an error bound $\epsilon_0 = \epsilon/2$. Since `sum` is computed for continuous random variables, we need to use discretization as discussed in §4.6.3.

The error for `sum` is the sum of the discretization error and the error given to the SUM algorithm; therefore, we can assign error bounds ϵ_1 and ϵ_2 for them respectively, where $\epsilon' = \epsilon_1 + \epsilon_2$. Next we need to allocate the error bound ϵ_1 to individual tuples, given the fact that the error accumulates across tuples. If n is the number of tuples in a given group, then each tuple can be uniformly assigned an error bound of ϵ_1/n. The allocation of ϵ_2 to each tuple is performed in the SUM algorithm. (See §4.6.3 for the discussion on how to choose the error bounds ϵ_1 and ϵ_2.) We have discussed discretization to compute one aggregate, i.e., case (1). The discretization for case (2), when multiple aggregates are computed, is similar, hence omitted.

6 Performance Evaluation

In this section, we evaluate the performance of our techniques for relational processing under the mixed-type model as presented in Section 4 and compare them with state-of-the-art techniques. We then evaluate our approach to query planning as discussed in Section 5. We have also conducted a case study of tornado detection to demonstrate the effectiveness of our data model and processing algorithms; the details are available in [29].

We use both synthetic streams with controlled properties, and real datasets and queries [25, 26] for evaluation. The experiments were run on a server with an Intel Xeon 3GHz CPU and 2GB memory running Java HotSpot 64-Bit server VM 1.6.

6.1 Joins and Aggregations under GMMs (TEP=1)

We first evaluate our techniques for joins and aggregates under Gaussian mixture models, and compare them to sampling and discretization methods.

6.1.1 Evaluation of Joins

We evaluate the techniques for joins using probabilistic views in §4.3.1 in the context of Query Q1. We use two streams R=(A^p) and S=(A, B), where B is deterministic and then summarize similar results observed when B is probabilistic. We then compare our techniques with state-of-the-art solutions including a sampling technique that constructs histograms to represent join result distributions directly from samples (which is adopted from [11]) and a discretization technique.

Input data. The R stream is an object location stream from an RFID inference system [26] where each tuple has a Gaussian distribution (using a mixture distribution will not incur more cost given our closed-form solution). The S stream, produced by our temperature simulator, contains tuples according to the underlying function between temperature and location, with added noise. The query-specified *update window size* (UW) on S is 1, i.e., containing the most recent temperature reading from each sensor. R and S tuples arrive at the same rate. Throughput measures the number of R tuples pipelined through the join.

Histogram-based sampling. For each R tuple, this method takes samples from the distribution of $R.A^p$. It then attempts to extend each sample a for $R.A^p$ with a sample b for $S.B$. To do so, it searches all S tuples in the update window for the two S tuples whose $S.A$ values are closest to the given sample a. It then applies linear interpolation to the $S.B$ values in these two tuples, with added random noise to facilitate later histogram construction, to obtain a sample b. Finally it uses all the samples (a, b) to construct an equi-depth 2-dimensional histogram as an approximate distribution for each join result (A^p, B^p). The histogram setting $\mathrm{H}(k, \mu)$ depends on the number of buckets per dimension, k, and the number of samples per bucket, μ.

Discretization. For each R tuple, we discretize its continuous distribution $R.A^p$ into equally spaced discrete points (as in [23]). For each discrete point a, we extend it with a value b for $S.B$ via linear interpolation as above, without adding noise. The join result is a discrete distribution containing pairs (a, b). A parameter of this technique is the number of discrete points used.

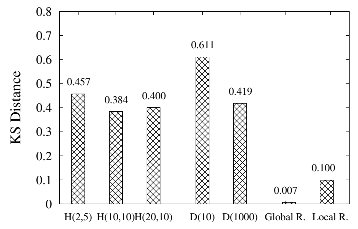
Expt 1: Sampling and discretization versus regression. We compare histogram-based sampling and discretization with our join technique using global or local regression. We first use a linear function to generate the temperature stream with added noise. The local regression region (LRR) is set to be 20. As seen in Fig. 12(a), histogram-based sampling and discretization fail to approximate true result distributions (KS ≥ 0.38) while our regression techniques are much more accurate (KS ≤ 0.10). This is because sampling and discretization consider only two S points in interpolation when matching a R point, thus sensitive to errors in temperature readings (in S). In contrast, our regression-based view uses more points to better estimate the underlying function and the random noise. The accuracy of sampling improves as more samples are used to construct the histogram, e.g., from H(2,5) to H(10,10). However, after a certain point, increasing k does not help, e.g., from H(10,10) to H(20,10), because when k is large, each bucket is very small and the samples in each bucket mostly fit the noise added during interpolation.

Fig. 12(b) shows that even when we tolerate a high KS values (≤ 0.4), sampling is already very slow: the throughput of H(10,10) is 33 tuples/sec and that of H(20,10) is 4. For discretization, as the number of discrete points increases, the accuracy improves but the throughput decreases. However, the accuracy is still low even for a large number of points. Our global regression gains a throughput of 1544 and local regression gains 52609, outperforming sampling and discretization methods by 2 to 4 orders of magnitude.

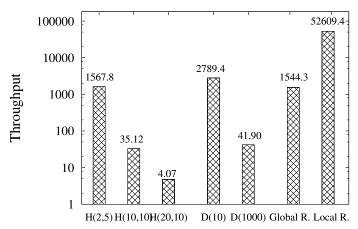
Expt 2: Global versus local regression. We next use a quadratic function to generate the temperature stream and compare global versus local regression. The sampling and discretization techniques perform poorly again, so we omit its results here. Since local regression is sensitive to the number of data points available, we vary its local regression region LRR in a wide range. As Fig. 12(c) shows, global regression has poor accuracy since its global linearity assumption is not true any more. The KS of local regression (UW=1) first improves as an increased region gives more points for regression, but then it degrades because the region is too large to meet the local linearity assumption—local regression becomes more like global regression.

A further optimization for local regression is to enlarge the update window UW, e.g., using the most recent 5 readings from each sensor. The rationale behind this is that the underlying function usually changes slowly, hence using some old tuples from the past few seconds will not add much stale information. We observe improved KS with UW=5 and 10 in Fig. 12(c), but reduced throughput since regression uses more points in Fig. 12(d). Despite that, local regression outperforms global regression by a wide margin. If we choose a reasonable setting, e.g., LRR=6 and UW=5, local regression can gain both high accuracy and efficiency.

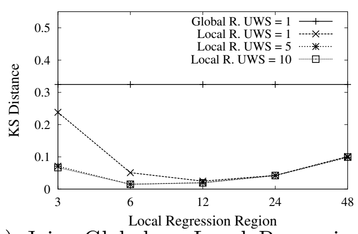
In another experiment, we generate the stream S=(A, B^p) where the attribute B is probabilistic. As stated in §4.3, we sample each S tuple on the attribute B, collect all the samples into a new input S', proceed as if S' has a deterministic attribute B. We have similar



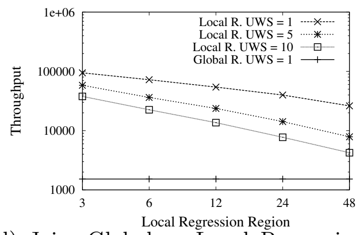
(a) Join: Sampling, Discretization vs Regression, linear function (Accuracy)



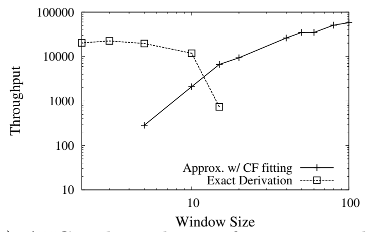
(b) Join: Sampling, Discretization vs Regression, linear funct. (Throughput)



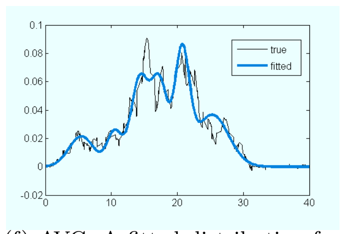
(c) Join: Global vs Local Regression, quadratic function (Accuracy)



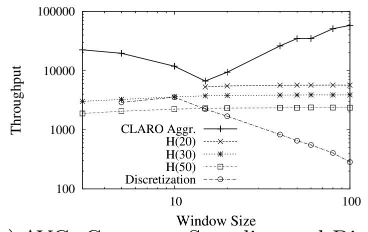
(d) Join: Global vs Local Regression, quadratic function (Throughput)



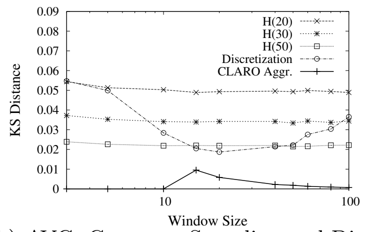
(e) AVG: Throughput of approx. and exact algorithms



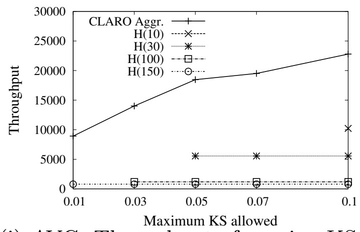
(f) AVG: A fitted distribution for the result of 5 tuples



(g) AVG: Claro vs Sampling and Discretization (Throughput)



(h) AVG: Claro vs Sampling and Discretization (Accuracy)



(i) AVG: Throughput of varying KS

Fig. 12 Experimental results for join and aggregation using our algorithms, histogram-based sampling and discretization (TEP=1)

observations about global and local regression, and their advantages over sampling (see [27]).

If there is more than one relation, S_i, that view-depends on a given relation, R, the cost of our evaluation technique is strictly linear in the number of relations. This is because we can perform the join between a tuple in R and each view S_i one at a time, and this join produces exactly one tuple for each R tuple.

6.1.2 Evaluation of Aggregations

Input data. We generate a synthetic tuple stream with one continuous uncertain attribute. Each tuple is modeled by a mixture of two Gaussian components. The means of the two components are uniformly sampled from [0, 5] and [5, 50] respectively to model complex real-world distributions from asymmetric to bimodal. The standard deviations are in [0.5, 1] and the coefficients are uniform from [0, 1]. We evaluate `avg` over this stream by using tumbling windows containing N tuples. The default KS requirement is KS ≤ 0.05.

Expt 4: Compare our algorithms. We first compare two algorithms, exact derivation and approximation using CF fitting, which constitute our hybrid solution. We vary the window size N, since it directly affects the result distribution and the computation needed.

Fig. 12(e) shows the throughput results in the number of tuples processed per second. As expected, the throughput of exact derivation is high when N is small, e.g., up to 10, but deteriorates quickly afterwards because the exact result formulas generated grow exponentially in N. In contrast, CF fitting works well for large numbers of N, e.g., after 10. This is due to the smoother result distributions in this range, hence easier to fit, and the one-time fitting cost being amortized over more tuples. We observe that both algorithms satisfy the requirement of KS ≤ 0.05. (The accuracy graphs are omitted due to space constraints.) The hardest range is 5 to 10 tuples, where the result distributions are complex and require a mixture of many components to fit, hence low throughput. An example of the true and fitted distributions for 5 tuples is shown in Fig. 12(f). From 15 tuples onwards, the result distributions become smoother with fewer peaks.

We also run experiments using the VD metric and other workloads, and observe the same trends in accuracy, throughput, and similar crossing points between the two algorithms. (The details are shown in [27].)

The above results suggest the configuration for the hybrid solution. When the number of tuples N is 10 or below, we use exact derivation. After that, we switch to CF fitting. In addition, when N is large enough (e.g. > 30), the result distributions are mostly a smooth Gaussian and can be computed directly using the Central Limit Theorem (CLT). In Expt 6 below, we will use this as an optimization when $N \geq 30$.

Expt 5: Compare to histogram-based sampling and discretization. We now compare our hybrid solution with the histogram-based sampling algorithm [11] and the discretization approach. Similar to the algorithm for joins, the sampling algorithm (1) generates $k \cdot \mu$ samples for each tuple, (2) performs aggregation over them to get $k \cdot \mu$ result samples, and (3) sorts the result samples and builds a histogram with k buckets and μ samples for each bucket. Since we find the accuracy of this algorithm to be more sensitive to k, we vary k among 20, 30, and 50 while fixing μ to 50. For discretization, we approximate continuous distributions using discrete points as in joins, and then use the algorithm in [16] to compute the distribution of `avg`.

Figs. 12(g) and 12(h) show the results of the three algorithms. Our hybrid algorithm outperforms all settings of histograms in both throughput and accuracy. For accuracy, only histograms with $k \geq 30$ ensures KS ≤ 0.05; $k = 20$ violates this in the "hard" range of 5 to 15 tuples (hence their throughput is omitted). The discretization approach offers no accuracy guarantee like the histogram method. So we manually varied the number of points and chose the best setting that met our accuracy requirement (see [27] for details). The throughput of this approach is shown to be even lower than that of histograms, especially when N is large. These results confirm the advantages of our algorithm over sampling and discretization since we can adapt to a given accuracy requirement while optimizing for throughput.

Expt 6: Vary the KS requirement. To further study our adaptivity to accuracy requirements, we vary KS from 0.01 to 0.1. The window size N is chosen randomly from the range $[2, 50]$, so that we can examine different ranges of the hybrid solution. Fig. 12(i) shows the throughput (where the KS requirement is met). Our algorithm outperforms the histogram algorithm for all values of KS by at least three times. Moreover, we can adapt to given accuracy requirements while it is unknown if a setting of the histogram algorithm can satisfy these requirements in advance.

6.2 Aggregations under Mixed-type Models (TEP ≤ 1)

In this section, we evaluate our approximation algorithms for aggregates of mixed-type models. We compare the performance of the deterministic algorithms and the randomized algorithm based on Monte Carlo simulation, which are presented in §4.6. They compute (ϵ, δ) approximation, where the former has $\delta = 0$ while for the latter, we set δ to 0.1, 0.05, and 0.01 (corresponding to three guarantees, 90%, 95% and 99%). Note that for both `max` and `sum`, the result tuple existence probability can be computed exactly, hence the KS error is used to quantify the approximate distributions only.

The default setting for the experiments is as follows. Each tuple has a tuple existence probability p uniformly sampled from [0, 0.5]. A tuple, if exists, has two possible real values uniformly sampled from [0, 20]. This way, each tuple corresponds to a mixed type distribution with an existence probability and two possible values, or $\lambda = 3$ in our setting. (This data model was also used for aggregation of uncertain data streams in [14].)

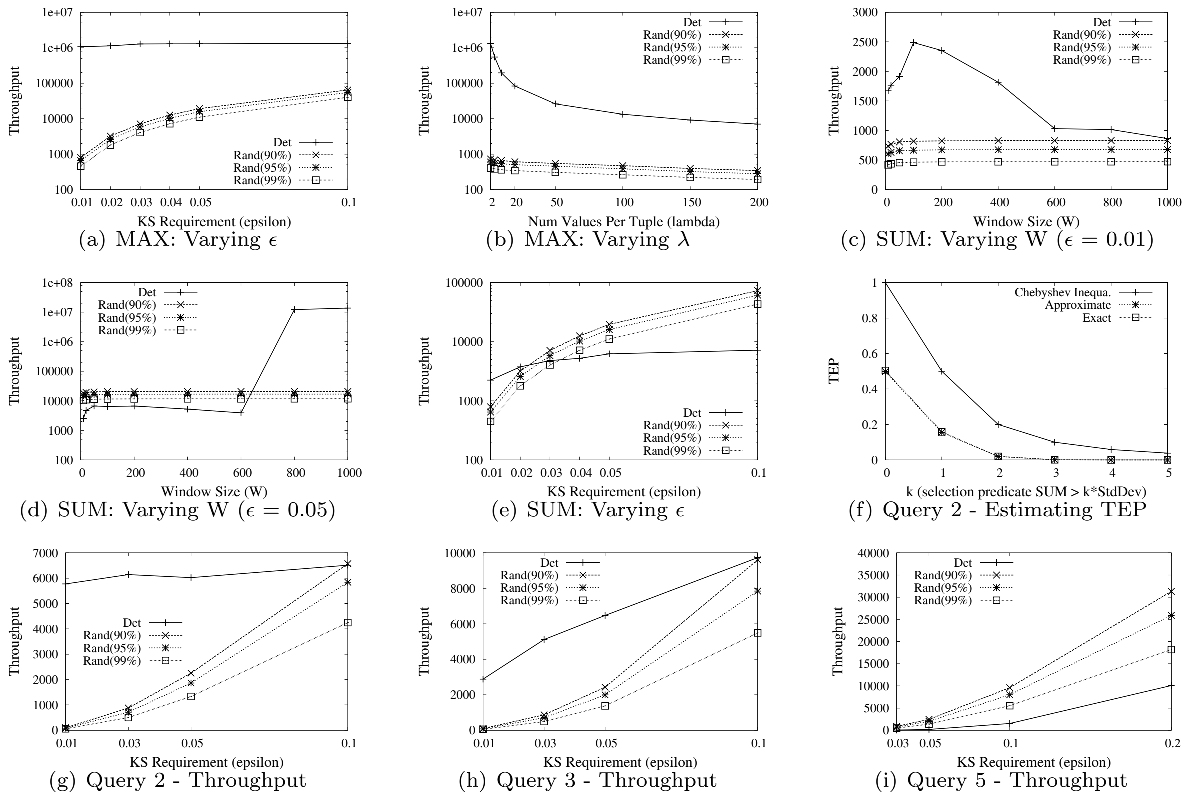
6.2.1 Evaluation of MAX

We first evaluate the performance of both the deterministic algorithm for `max`, D_{max}, and the randomized algorithm, Rand.

Expt 7: Vary the KS requirement. We vary the KS bound ϵ in a common range [0.01, 0.1]. The window size W is uniformly sampled from [10, 1000]. Fig. 13(a) shows the throughput of the algorithms. The deterministic algorithm, D_{max}, is 10 to 1000 times faster than the randomized algorithm, Rand, for all ϵ values tested. This is because 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 D_{max} is more accurate than Rand, because it sets its parameter ϵ', to meet the worst case scenario (i.e., reaching the maximum generation $\log \mathbb{U}$).

Expt 8: Varying the number of values per tuple. 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. 13(b) shows the throughput results. As expected, the cost of D_{max} increases with λ due to the costs of the first two steps of 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. Overall, the throughput of D_{max} is better than that of Rand by at least one order of magnitude.

We also perform an experiment to compute the distribution of `max` using discretization and the extensional semantics [7]. As for `avg` in Expt 5, this approach has no accuracy guarantee. Overall, we observe that its throughput is up to 10 times lower than D_{max}. The details are omitted due to space constraints, but available in [27].



(a) MAX: Varying ϵ (b) MAX: Varying λ (c) SUM: Varying W ($\epsilon = 0.01$)

(d) SUM: Varying W ($\epsilon = 0.05$) (e) SUM: Varying ϵ (f) Query 2 - Estimating TEP

(g) Query 2 - Throughput (h) Query 3 - Throughput (i) Query 5 - Throughput

Fig. 13 Experimental results of algorithms for MAX, SUM (TEP ≤ 1), and query planning

6.2.2 Evaluation of SUM

We now evaluate the performance of the deterministic algorithm for sum, D_{sum}, using the optimizations shown in §4.6.3, and the randomized algorithm, Rand.

Expt 9: Varying the window size. We vary W from 10 to 1000 for two values of ϵ, 0.01 and 0.05. Figs. 13(c) and 13(d) show the throughput. For $\epsilon = 0.01$, D_{sum} is faster than Rand in all settings because Rand uses a number of samples proportional to $1/\epsilon^2$, while D_{sum} uses about $1/\epsilon$ intervals to estimate the distribution. The throughput of D_{sum} decreases with W, since the additive error bound of D_{sum} requires provisioning error bounds to W tuples. For $\epsilon = 0.05$, 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 $\epsilon = 0.1$.

Expt 10: Varying the KS requirement. We compare the two algorithms with ϵ from 0.01 to 0.1. W is uniformly taken from $[1, 100]$, so that CLT cannot apply. Fig. 13(e) shows that the throughput of D_{sum} is comparable to higher than Rand for the high-precision range [0.01, 0.03]. This confirms that to gain high accuracy, Rand needs a very large number of samples and degrades its performance quickly. When we do not require high accuracy, Rand can be used for good throughput.

Summary: The above experiments, considered as a micro-benchmark, offer insights into our techniques and their performance compared to sampling techniques. We observed that our algorithms for joins and aggregations under GMMs consistently outperform an existing histogram-based technique. Under more complex mixed-type models, our deterministic algorithm for max is constantly faster than our randomized algorithm using Monte Carlo simulation by orders of magnitude. For sum, there is a tradeoff between the two algorithms—the deterministic algorithm is more efficient for sum of tuples with a small number of possible values (e.g., Bernoulli variables) under high accuracy requirements, while the randomized algorithm is preferable for other cases.

6.3 Approximate Query Answers

We now evaluate the performance of four queries, Q2 to Q5, whose query plans are shown in Section 5.

Expt 11: Q2. To run this query, we first obtain a stream of inferred object locations, each of which is modeled by a Gaussian distribution, by running inference [26] over a raw RFID reading stream. This query computes the sum of object weights per group and checks if it exceeds 200. Although the weight of an object is deterministic, each object belongs to a group with a probability, resulting in the `sum` of Bernoulli variables, or $\lambda = 2$ (see Fig. 2). This is a common case for aggregation on a deterministic attribute under tuple uncertainty. Given a query accuracy requirement ϵ, the predicate "`sum` > 200" requires assigning an error bound $\epsilon/2$ to the SUM algorithm.

We first compare our deterministic algorithm (with $\epsilon = 0.05$) with an alternative method that uses only the moments of the `sum` distribution to estimate the TEP when evaluating the `having` predicate "`sum` > v". This method cannot return the distribution of `sum` in the query result, so we restrict the comparison to computing TEP 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. 13(f) shows the estimates of the TEP as we vary the threshold v in the predicate. As seen, using the Chebyshev's inequality can be very inaccurate, thus confirming the need to use the `sum` distribution to compute the TEP.

We next compare the performance of the deterministic algorithm, $\mathrm{D_{sum}}$, and the randomized algorithm, Rand, to compute query result distributions. Fig. 13(g) shows that $\mathrm{D_{sum}}$ is faster than Rand. This is because smaller error bounds are provisioned to the aggregates 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. 13(e).

Expt 12: Q3. For the next three queries, we use a dataset from the Sloan digital sky survey (SDSS) project [25], where the uncertain attributes are modeled by Gaussian distributions. Q3 computes the maximum of luminosity per group and selects groups where `max`(*luminosity*) > 20. The main difference from Q2 is that the aggregate attribute is continuous. Hence, we set the universe size $\mathbb{U} = 40000$, assuming a high measurement precision of three decimal places.

We again consider an alternative method that estimates the TEP of result tuples using only the moments of the `max` distribution and summarize the result here. Since the state-of-the-art technique [14] can only compute the mean of `max`, we use the Markov's inequality to derive an upper bound for the TEP. We observe that using this technique can give inaccurate estimates, e.g., the error of the TEP can be as high as 0.6.

We now compare our deterministic and randomized algorithms, $\mathrm{D_{max}}$ and Rand. $\mathrm{D_{max}}$outperforms Rand under all chosen accuracy requirements ϵ, as shown in Fig. 13(h), especially for high ϵ. This confirms that $\mathrm{D_{max}}$ still performs well for large numbers of values per tuple λ by bounding the number of intervals in the distribution of `max`. Compared to Fig. 13(a), the throughput decreases for small ϵ, because this is the case when the update time is roughly proportional to $1/\epsilon$ (since λ is large), as shown in Theorem 3.

Expt 13: Q4. This query computes `avg`(*rowc*) and `avg`(*colc*) for objects grouped according to the deterministic attribute *HTM_ID*. The result TEP of an object in a group after group-by is deterministic (either 0 or 1). Since *rowc* and *colc* follow Gaussian distributions in the dataset, their `avg` are also Gaussian and can be computed exactly with high throughput of millions tuples per second. As a variant, we compute `max` instead and observe that using $\mathrm{D_{max}}$ is 2 to 10 times faster than using Rand for this query (it is similar to Fig. 13(h), except for provisioning smaller error bounds).

Expt 14: Q5. This query is similar to Q3, but computes `sum`(luminosity). Fig. 13(i) shows the throughput of two algorithms, $\mathrm{D_{sum}}$ and Rand. Since *luminosity* is continuous-valued, we use discretization before computing `sum`. Therefore, $\mathrm{D_{sum}}$ has two types of errors: errors from estimation of the input tuple, or discretization errors, and errors from approximating `sum`. Both errors accumulate with the number of tuples, having $\mathrm{D_{sum}}$ provision a small error bound per tuple. We observe that $\mathrm{D_{sum}}$ performs worse than Rand, which indicates that Rand is useful for computing the `sum` of continuous distributions or distributions with a large number of possible values.

Summary: We have applied our techniques for query planning to handle error occurrence and propagation in conditioning and aggregation queries on the real datasets. We observed that for `max`, our deterministic algorithm, even with continuous input, hence a large number of values per tuple, outperforms the randomized counterpart, whereas for `sum`, our deterministic algorithm works well for Bernoulli variables or tuples with a few values, but further discretization of continuous distributions makes it less desirable than our randomized algorithm. Overall, we can process thousands of tuples per second for most queries tested.

7 Related Work

Have discussed the existing work closely related to ours in previous sections, we now survey the broader areas.

Probabilistic databases with discrete uncertainty. Much of existing work on probabilistic databases, e.g., [2,3,7,22,30], uses discrete distributions to model tuple and attribute uncertainty. Then query evaluation is based on the possible worlds semantics. Our work considers data naturally characterized by continuous distributions, precluding the existing techniques for discrete distributions. This motivates our work to seek new data models, the formal semantics, and consider relational processing for continuous-valued data. The most relevant work on discrete uncertain data is estimating the probability of a predicate aggregate in the `having` clause using Monte Carlo simulation and returning the expectations of the uncertain attributes in query results [7,21]. In contrast, our work aims to compute full result distributions and explores a range of both deterministic and randomized algorithms.

Probabilistic databases with continuous uncertainty. An existing work [5] considers uncertainty modeled by continuous distributions and uses integration for operations such as aggregates, as we discuss in §4.4. The ORION [23] system is designed to support continuous uncertainty, but it mostly focuses on probabilistic modeling and considers a subset of relational operations, i.e., selection, projection, and join. Two recent workshop papers [1,24] also consider the extension of probabilistic databases with continuous-valued attributes. While they mainly present the motivation or initial design, they make similar arguments as in this paper for a suitable model for continuous random variables and the need to compute distributions. There are proposals to use discretization and employ possible worlds semantics for relational processing [1,23]. This is however inefficient as we show empirically in §6.

Probabilistic stream processing has gained research attention recently. Existing work [6,15,16] adopts the finite and discrete tuple model as in probabilistic databases, and is not directly applicable for continuous uncertainty. Besides, most of these techniques compute the mean or a few higher moments of result distributions [6,15]. We show in §6.3 that knowing a few moments of aggregates is not enough to answer queries accurately.

8 Conclusions and Future Work

In this paper, we presented the CLARO system for uncertain data stream processing. We proposed the mixed-type data model that captures tuple existence uncertainty and employs Gaussian mixture distributions to characterize continuous uncertain attributes. We defined the semantics for relational processing under the mixed-type model. We proposed advanced techniques for relational processing to obtain exact, closed-form solutions when possible, and fast approximation with bounded errors in this model. We also presented a technique for query planning for complex queries that meets query accuracy requirements. Our experimental results show that CLARO outperforms sampling methods in both accuracy and throughput for most workloads tested.

For future work, we plan to extend this work in new directions including query optimization, correlation among multiple aggregates and across tuples, and support for user-defined functions.

Acknowledgements. This work was supported in part by the National Science Foundation under the grants IIS-0746939, IIS-0812347, EEC-0313747, and CCF-0953754, and in part by the National Security Agency under the grant H98230-09-1-0044.

References

1. P. Agrawal and J. Widom. Continuous uncertainty in trio. In *MUD Workshop*, 2009.
2. L. Antova et al. Fast and simple relational processing of uncertain data. In *ICDE*, 983–992, 2008.
3. O. Benjelloun et al. Uldbs: Databases with uncertainty and lineage. In *VLDB*, 953–964, 2006.
4. G. Cassella et al. *Statistical Inference*. Duxbury, 2001.
5. R. Cheng et al. Evaluating probabilistic queries over imprecise data. In *SIGMOD*, 551–562, 2003.
6. G. Cormode and M. Garofalakis. Sketching probabilistic data streams. In *SIGMOD*, 281–292, 2007.
7. N. N. Dalvi and D. Suciu. Efficient query evaluation on probabilistic databases. *VLDB J.*, 16(4):523–544, 2007.
8. A. DasGupta. *Asymptotic theory of statistics and probability*. Springer Verlag, 2008.
9. A. Deshpande and S. Madden. MauveDB: supporting model-based user views in database systems. In *SIGMOD*, 2006.
10. Y. Diao et al. Capturing data uncertainty in high-volume stream processing. In *CIDR*, 2009.
11. T. Ge and S. B. Zdonik. Handling uncertain data in array database systems. In *ICDE*, 1140–1149, 2008.
12. C. Guestrin et al. Distributed regression: an efficient framework for modeling sensor network data. In *IPSN*, 2004.
13. R. Jampani et al. Mcdb: a monte carlo approach to managing uncertain data. In *SIGMOD*, 687–700, 2008.
14. T. S. Jayram et al. Efficient aggregation algorithms for probabilistic data. In *SODA*, 346–355, 2007.
15. T. S. Jayram et al. Estimating statistical aggregates on probabilistic data streams. *ACM TODS*, 33(4), 2008.
16. B. Kanagal et al. Efficient query evaluation over temporally correlated probabilistic streams. In *ICDE*, 2009.
17. R. H. Lopes et al. The two-dimensional kolmogorov-smirnov test. In *Proc of the XI Int'l Workshop on Advanced Computing and Analysis Techniques in Physics Research*, 2007.
18. G. McLachlan and D. Peel. *Finite Mixture Models*. Wiley-Interscience, 2000.
19. Y. Qi et al. Threshold query optimization for uncertain data. In *SIGMOD*, 315–326, 2010.
20. C. Ré et al. Event queries on correlated probabilistic streams. In *SIGMOD*, 715–728, 2008.
21. C. Re and D. Suciu. The trichotomy of having queries on a probabilistic database. In *VLDBJ*, 2009.
22. P. Sen et al. Exploiting shared correlations in probabilistic databases. In *VLDB*, 2008.
23. S. Singh et al. Database support for probabilistic attributes and tuples. In *ICDE*, 1053–1061, 2008.
24. D. Suciu et al. Embracing uncertainty in large-scale computational astrophysics. In *MUD Workshop*, 2009.
25. A. S. Szalay et al. Designing and mining multi-terabyte astronomy archives. In *SIGMOD*, 451–462, 2000.
26. T. Tran et al. Probabilistic inference over RFID streams in mobile environments. In *ICDE*, 2009.
27. T. T. L. Tran el al. Claro: Modeling and processing uncertain data streams. UMass Amherst, 2011. `http://www.cs.umass.edu/~ttran/pubs/claro-tr.pdf`.
28. T. T. L. Tran et al. Conditioning and aggregating uncertain data streams: Going beyond expectations. In *PVLDB*, 2010.

29. T. T. L. Tran et al. Pods: A new model and processing algorithms for uncertain data streams. In *SIGMOD*, 2010.
30. D. Z. Wang et al. Bayestore: Managing large, uncertain data repositories with probabilistic graphical models. In *VLDB*, 2008.
31. J. Widom. Trio: A system for integrated management of data, accuracy, and lineage. In *CIDR*, 262–276, 2005.