

Continuous Uncertainty in Trio

Parag Agrawal, Jennifer Widom
{paraga,widom}@cs.stanford.edu

Stanford University

Abstract. We present extensions to Trio for incorporating *continuous uncertainty* into the system. Data items with uncertain possible values drawn from a continuous domain are represented through a generic set of functions. Our approach enables precise and efficient representation of arbitrary probability distribution functions, along with standard distributions such as Gaussians. We also describe how queries are processed efficiently over this representation, without knowledge of specific distributions. For queries that cannot be answered exactly, we can provide approximate answers using sampling or histogram approximations, offering the user a cost-precision trade-off. Our approach exploits Trio’s *lineage* and *confidence* features, with smooth integration into the overall data model and system.

1 Introduction

There has been a great deal of interest recently in developing database management systems to handle data that is *uncertain* [1, 3, 10, 13]. Many of these efforts, including our own work in *Trio* [13], use a data model based on a set of *alternative values* for tuples, and/or *probabilities* (or *confidences*) assigned to a tuple’s existence. However, some applications must manage data that cannot be represented with such forms of *discrete* uncertainty. For example, sensor measurements may be described by a Gaussian distribution around the reported reading (based on sensor calibration) [5]; astronomical data may contain locations described by two-dimensional Gaussians; a database that records time-of-day may use intervals bounding the possible time; predictive models stored in a database may include continuous probability distributions.

There have been a few proposals for managing continuous uncertainty in a DBMS. Reference [5] suggests using an abstract data type for continuous uncertain values, focusing specifically on Gaussian distributions. The *Orion* system [11] generalizes to other distributions using discrete histogram approximations, and defines a semantics for interpreting SQL over this data. In this paper we describe how to incorporate continuous uncertainty into Trio. Our proposal is more general than the previous work in both data representation and querying, but admittedly it is not implemented yet. More detailed discussion of related work is provided in Section 6.

Next we describe in brief the key components of our proposal: semantics of the data model and query language, data representation in the system, query

processing over this representation, interfaces to the uncertain data, and integration into other aspects of Trio, including Trio’s model for discrete uncertainty and its important *lineage* feature.

Data Model Semantics: Models for uncertain data usually are based on *possible worlds*: a set of possible instances for the database. With discrete uncertainty, an uncertain database always represents a finite set of possible-instances; with continuous uncertainty, we may represent an infinite set. Specifically, the value of an uncertain attribute may be an arbitrary *probability distribution function (pdf)* over a continuous domain, describing the possible values for the attribute. For example, an uncertain temperature may be described as a Gaussian distribution around a mean temperature of 50 with variance 12; a time may be described as a uniform distribution between 9:20 and 9:25. Gaussian and uniform distributions are special cases of pdfs, as are the discrete alternatives on which Trio was based originally. In our model, a pdf may also be over a high-dimensional domain, representing multiple correlated uncertain attributes. For example, predicted object locations may be comprised of uncertain correlated *latitude-longitude* pairs, i.e., two-dimensional pdfs.

Query Semantics: With possible-world semantics, the semantics of a standard query is defined naturally: The result of a query Q over an uncertain database U must represent the result of applying Q to each possible-instance of U . There is no problem with these semantics even when the set of possible-instances is infinite, although some query results may be surprising. For example, a query filtering by equality on values that are continuous pdfs will always return an empty result: the probability of a value from a pdf being equal to any specific value is zero.

Representation: Previous work [5, 11] has used symbolic representations for a set of known distribution types, and histogram approximations for the rest. For example, Gaussian distributions can be represented by mean and variance; uniform distributions by their endpoints. We have taken a different approach to handling a wide class of distributions: we represent a pdf by a set of functions. For example, a pdf ρ can be represented by a pair of functions:

- The function $weight(low,high)$ returns the total probability of a value in ρ lying between low and $high$. For example, $weight(3,4)$ for the uniform distribution over $[2, 6]$ returns 0.25.
- The function $sample(low,high)$ returns a random value between low and $high$ according to ρ . For example, $sample(3,4)$ for a Gaussian distribution with mean 4 is more likely to return a value close to 4 than close to 3.

We also generalize both functions to multidimensional pdfs. This approach subsumes the symbolic representation for specific distributions, while at the same time enabling higher efficiency and accuracy than histogram approximations for the rest.

Query Processing: In Trio, query results include *lineage* identifying the data from which each result value was derived [13]. Lineage is needed to properly represent uncertainty, and to compute result confidence values lazily. (Orion [11] uses *history* in a similar way.) In the presence of pdfs, we make all processing over pdf values lazy, deferring potentially expensive computations until they are needed. At query time, we simply generate lineage, and for those results involving pdfs, we extend the lineage to include relevant predicates and mappings. The information contained in the lineage is sufficient to be able to compute the functions that encode the result lazily. For example, a query asking for temperatures less than 60 would generate lineage annotated with the “< 60” predicate. Now if we have a result pdf ρ' whose lineage points to a pdf ρ , the function $weight_{\rho'}(low,high)$ on ρ' is translated to $weight_{\rho}(low,\max(high,60))/weight_{\rho}(-\infty,60)$ on ρ .

The “translation” approach motivated above can be used to answer many queries efficiently. However, for expensive queries we support approximate answers, either using the *sample* function, or a histogram based on the *weight* function; both options offer the user a cost-precision trade-off. Query processing is modular and generic: our approach of mapping functions on result data to functions on data in the result’s lineage is independent of specific distributions—all functions can be treated as “black boxes.” However, for specific distributions we do introduce specialized processing that is more efficient.

Interfaces: It is difficult to know how to present a pdf in a database system API or user interface. Our approach is to offer an extensible suite of relevant properties of a pdf; examples are *mean*, *variance*, *weight in a range*, or *histogram approximation*. All of these properties are computed easily using calls to the functions that represent the pdf (like *weight* and *sample*), again possibly using an approximation. Users may add additional properties by providing translations to the functions.

Integration into Trio: In addition to lineage and discrete uncertainty, Trio’s data model includes *confidence* values associated with tuples, denoting the probability of the tuple existing. In our new model and query language for continuous uncertainty, operations tend to modify the probability of a tuple’s existence. Thus, the confidence feature is particularly helpful for integrating pdfs into Trio. Consider for example the predicate “ ≤ 4 ” applied to a uniform distribution over [3, 7]. The result can be represented by a uniform distribution over [3, 4] that exists with probability 0.25, or more generally one fourth of the original probability.

Not only do Trio’s previous features (lineage and confidences) help integrate continuous uncertainty into the system, but the ability to represent and query pdfs enhances some pre-existing Trio functionality. For example, aggregate query results [8] may naturally be represented as continuous pdfs, and pdfs over discrete domains can sometimes yield more efficient representations than tuple alternatives, for certain types of data.

2 Data Model

2.1 Original ULDB Data Model

We present a brief overview of the the basic ULDB model; for details see [2]. Our extensions for continuous uncertainty will be presented in Section 2.2. We use a contrived example for illustrative purposes. Table `Reading(sensor-id,temp)` maintains temperature observations from sensors, and the table `Location(sensor-id,zone)` maintains locations of sensors.

Alternatives: ULDB relations are comprised of *x-tuples*. Each x-tuple consists of one or more *alternatives*, where each alternative is a regular tuple over the schema of the relation. For example, if sensor 1 is located in zone *A* or *B*, then in table `Location` we have:

<code>Location(sensor-id,zone)</code>
(1,A) (1,B)

This x-tuple yields two *possible instances* or *possible worlds* for table `Location`. In general, the possible instances of a database *D* correspond to all combinations of alternatives for x-tuples in *D*. In cases like the example tuple above in which only some attributes are uncertain, attribute-level uncertainty may be used:

sensor-id	zone
1	A B

Confidences: Numerical *confidence* values may be attached to alternatives of an x-tuple. Suppose sensor 1 is in zone *A* with confidence 0.4 and *B* with confidence 0.6, while sensor 2 is operating at all with confidence 0.7. Then we have:

<code>Location(sensor-id,zone)</code>
(1,A):0.4 (1,B):0.6
(2,C):0.7

The sum σ of confidence values of all alternatives in an x-tuple must be at most 1. If σ for an x-tuple is less than 1, with probability $1 - \sigma$ the tuple is not present. X-tuples are assumed to be independent

Lineage: Lineage in ULDBs is recorded at the granularity of tuple alternatives: lineage connects an x-tuple alternative to the other x-tuple alternatives from which it was derived. Consider joining tables `Location` and `Reading` to create a table `Temp(zone, temp)`. Let column ID serve as a unique identifier for each x-tuple, and let (i, j) denote the j^{th} alternative of the x-tuple with ID i . See the join result below. For example, tuple $(B, 77)$ in `Temp` (identified by $(31, 2)$) is derived from $(1, B)$ in `Location` and $(1, 77)$ in `Reading`.

ID	<code>Location(sensor-id,zone)</code>
11	(1,A):0.4 (1,B):0.6
12	(2,C):0.7

ID	Reading(sensor-id,temp)
21	(1,77)
22	(2,81)

ID	Temp(zone,temp)
31	(A,77):0.4 (B,77):0.6
32	(C,81):0.7

$\text{lineage}(31,1) = \{(11,1), (21,1)\}$

$\text{lineage}(31,2) = \{(11,2), (21,1)\}$

$\text{lineage}(32,1) = \{(12,1), (22,1)\}$

Possible Worlds Semantics: Query results in Trio follow possible worlds semantics. Consider a ULDB D whose possible instances are D_1, \dots, D_n . The result of a query Q on D must have as its possible instances $Q(D_1), \dots, Q(D_n)$. For example, the join of **Location** and **Reading** above should logically perform the join in all four possible instances of the ULDB. Of course a query processing algorithm does not expand possible instances; it computes the ULDB representation of the query result directly from D .

Lineage imposes restrictions on the possible instances of a ULDB, effectively coordinating the uncertainty in derived data with the uncertainty in the data from which it was derived. Roughly speaking, a tuple alternative is present in a possible instance if and only if all the alternatives in its lineage are present, although the actual constraints are somewhat more complex [2]. For the example above, $(B, 77)$ is present in exactly those possible instances that also contain the tuples in its lineage, i.e., $(1, B)$ and $(1, 77)$ are present.

Lazy Confidence Evaluation: The confidence values associated with alternatives in a query result must be consistent with possible worlds semantics: the confidence of an alternative is always the sum of probabilities of all possible worlds that contain that alternative. It is $\#P$ -hard to compute these confidence values in general. The use of lineage allows us to compute result confidences either during query processing or on-demand [13].

2.2 Continuous Uncertainty Extensions

The following extensions to the data and lineage components of the ULDB model enable it to manage continuous uncertainty. Note that here we are describing the model only. Specific representation and query processing details are discussed in Sections 3 and 4 respectively.

Pdf Attributes: We now allow *pdf attributes* in x-tuple alternatives. The value for a pdf attribute is given by a probability distribution function over a given domain. The domain may be discrete or continuous. For example, $P(A) = 0.4; P(B) = 0.6$ is a pdf over a discrete domain $\{A, B\}$; a pdf of a Gaussian distribution $G(77, 20)$ with mean 77 and variance 20 is a pdf over the continuous

domain of all real numbers \mathcal{R} . Suppose the sensors used to record temperature values in the **Reading** table have errors described by Gaussian distributions: say sensors 1 and 2 have variance of 20 and 30 respectively. The uncertainty in a sensor’s temperature report can now be captured in the database by making **temp** a pdf attribute over \mathcal{R} . Using $G(\mu, \sigma^2)$ to represent a Gaussian pdf:

Reading(sensor-id,temp)	
(1,	$G(77,20)$)
(2,	$G(81,30)$)

The **Reading** table now has an infinite number of possible worlds, since it contains a pdf attribute over a continuous domain. Any table with two tuples of the form $(1, x)$ and $(2, y)$ is a possible instance of **Reading**. As with other uncertain data in Trio, the values of pdf attributes are independent. Hence, the probability of a possible instance $(1, x), (2, y)$ is obtained by multiplying the value $G(77, 20)$ assumes at x with the value $G(81, 30)$ assumes at y . The probability of an individual possible world is infinitesimally small, but instances with values of x and y close to the respective means 77 and 81 are more likely.

Note that Trio’s attribute-level uncertainty (Section 2.1) is captured by pdf attributes over finite discrete domains. Also note that the presence of alternatives along with pdf attributes allows us to represent certain cases that pdf attributes alone can’t represent. For example, this x-tuple represents a temperature 78 with probability 0.3 or a Gaussian around 77 with variance 20:

Temperature(zone,temp)	
(A,78):0.4	(A, $G(77,20)$):0.7

Predicates applied to pdf attributes can affect confidence values of result alternatives. For example, consider a tuple that exists with confidence 0.8 and has just one pdf attribute whose value is uniformly distributed between 2 and 6 (denoted by $U(2, 6) : 0.8$). The result of applying a predicate $A \leq 5$ to this tuple is $U(2, 5) : 0.6$.

We also support correlated pdf attributes. Correlations may exist in the base data, for example uncertain location data may be represented by correlated x and y coordinates. A joint pdf (over possibly heterogeneous domains) is used to represent multiple correlated attributes. Correlated pdf attributes, and processing queries over them, are described in more detail in Section 5.1. Note that independent attributes in base data can yield correlated attributes in the result. For example, a selection query with predicate $A < B$ over pdfs A and B can make attributes corresponding to A and B be correlated in result. Lineage is used to correctly handle these correlations, as described in Section 4.2 .

Lineage Extensions: An important part of our ULDB extension is to include *predicates* and *mappings* as part of lineage, but at the table granularity. Predicates in the **where** clause of a query that reference at least one pdf attribute are

not evaluated at query time. Instead, they are stored as part of the new table-level *pdf-lineage*. In addition, references to pdf attributes in the `select` clause are stored in the pdf-lineage as mappings from result attributes to expressions over input attributes. Pdf-lineage allows us to evaluate the values for the pdf attributes in a result alternative lazily, along with evaluation of confidence values. The generation of pdf-lineage is described in Section 4.1, and its utilization for on-demand uncertainty processing is described in Section 4.2.

3 Representation and Interfaces

We now describe how pdf attributes can be represented in the system and exposed to the user or application. For now, we describe our techniques for pdfs over the simple domain of real numbers. We will discuss higher-dimensional domains for joint pdfs in Section 5.1.

A pdf is represented by an extensible set of functions. Consider a pdf value ρ with a density function f . This pdf can be represented by a function $weight(l, h)$ that returns the weight of the pdf ρ between l and h .

$$weight(l, h) = \int_l^h f(x)dx$$

For example, for Gaussians with mean μ and variance σ^2 :

$$weight(l, h) = erf\left(\frac{h - \mu}{\sigma\sqrt{2}}\right) - erf\left(\frac{l - \mu}{\sigma\sqrt{2}}\right)$$

where erf is the error function. The *weight* function can completely describe any pdf, but we may often use additional functions in the representation for increased efficiency and convenience. One such function is *sample(l,h)*, which returns values between any given l and h drawn at random according to the pdf.

Notice that a call to *sample* can be computed by using calls to *weight* and a uniformly-distributed random number generator. Conversely, an approximation to *weight* can be computed using calls to *sample* with Monte-Carlo simulations. In addition to *weight* and *sample*, we might use *inverse-cdf*, *fourier*, or *describe* for a pdf ρ representing a random variable X :

- The function *inverse-cdf(v)* returns a value r such that $v = weight(-\infty, r)$. Intuitively, *inverse-cdf(v)* returns the $100v$ percentile value for the pdf. For example, the *inverse-cdf* of a Gaussian with mean μ and variance σ^2 is:

$$inverse-cdf(v) = \mu + \sqrt{2}\sigma erf^{-1}(2v - 1)$$

If the *inverse-cdf* function is not instantiated, it can be computed approximately using calls to *weight* and binary search, or using *sample*. Functions *weight* and *sample* can also be computed using *inverse-cdf*.

- The function *fourier*(x) returns the value of the Fourier transform of the pdf of the random variable X at x . The Fourier transform may be known as an expression for certain distributions. Efficient numerical computation of the Fourier transform and inverse Fourier transform has been studied extensively. Function *fourier* may be computed using *weight* and vice-versa.
- The function *describe*() returns a parametrized form of the pdf. For example, *describe* might return [Type:"Gaussian",Parameters: μ, σ^2]. For a known type of pdf, the system can compute *describe* using *weight*, *sample*, or *inverse-cdf*, and vice-versa.

Since the functions above can be computed using calls to other functions, not all of them need to be instantiated. In fact, all the above functions can be computed (approximately) if any one of them were instantiated. Of course when more functions are instantiated, it may be possible to obtain more precise answers more efficiently. The functions above are our starting point. We will provide hooks to the user to create additional functions.

Pdf data represented as functions is presented to the application or user through *interfaces*. Interfaces, like functions, return properties of the pdf, but they are not used to represent the pdf and hence cannot be used to compute functions or other interfaces. Here are two examples of interfaces:

- A *discrete histogram approximation* might be used by a human to visualize a pdf. This interface is computed easily using *weight*: the height of a histogram bar of width $2 \cdot \epsilon$ around a is the result of *weight*($a - \epsilon, a + \epsilon$).
- Some applications may ask for the *median* of a pdf attribute. The property of the median m is that *weight*($-\infty, m$) = *weight*($m, -\infty$) = 0.5. Median m can be found approximately by calling *weight* repeatedly with binary search. The precision of the result improves as more calls are made.

Although *weight* is suitable for computing the discrete histogram interface, it isn't ideal for median, and possibly other interfaces. On the other hand, median can be computed easily using *inverse-cdf* by calling it with argument 0.5. If *inverse-cdf* is instantiated, it should be used for a median interface. If *inverse-cdf* is not instantiated, calling it from the median interface will degenerate to the same binary search over calls to *weight*. Some other interesting interfaces might be are mean, variance, percentile, inverse-percentile, and human-readable form of *describe*. All can be implemented using the functions described earlier, some more efficiently than others.

Our extensible functional representation, along with the notion of interfaces, sets up a framework permitting easy incorporation of new techniques for pdf attributes. We will discuss how the set-of-functions approach enables efficient query processing, and introduces a new optimization problem over a space of "plans" that can be compared in terms of precision and efficiency.

4 Query Processing

We consider select-project-join queries over x-relations with pdf attributes. For this paper, we consider only conjunctive **where** clauses. Generalizing the query

language is a topic of future work. Users and applications may pose queries, use *interfaces* (as described in Section 3) to examine the result data, and invoke confidence computation in the usual Trio style [13].

In Trio, query results include lineage identifying the data from which each result value was derived, as described in Section 2.1. Along with tracing the origins of result data, lineage is also used to properly represent possible worlds semantics, and to compute result confidence values lazily. For pdf attributes we also use lineage to handle uncertainty in a lazy manner, deferring potentially expensive computations until they are needed. More precisely, we generate lineage at query-time, then use it to compute the functions corresponding to the pdf attributes in the result data.

4.1 Query-Time Processing

Query-time processing involves generating both data and lineage. We describe each of these in turn, focusing on new requirements for managing pdf attributes.

Data Processing: In Trio, data processing at query time uses a standard relational processor. We use the algorithm described in [13] with the following modifications:

- Any predicate in the **where** clause referencing one or more pdf attributes is not evaluated. The query is processed using only the remaining predicates in the **where** clause. (Recall we assume conjunctive **where** clauses).
- A *placeholder* is created for each element in the *select* clause that is of type pdf.

For example, consider a Trio query over relation $R(A, B, C)$, where A and B are pdf attributes:

```
select A, B+5, C, D
from R
where A ≤ 5 and B ≤ A and C = 11
```

We instead execute the Trio query:

```
select X1, X2, C, D
from R
where C = 11
```

where $X1$ and $X2$ are placeholders.

Lineage Generation: For each alternative in the result, lineage is generated as in [13]. In addition, we generate table-level *pdf-lineage* as introduced in Section 2.2. Specifically, pdf-lineage is composed of two parts: *predicates* P , recording all unevaluated predicates from the **where** clause, and *mappings* M , recording

mappings from `select` clause placeholders to expressions over input attributes. Intuitively, pdf-lineage captures portions of the query that we wish to evaluate later.

For the example query above, P is $A \leq 5 \wedge B \leq A$, and M is $X1 \rightarrow R.A, X2 \rightarrow R.B+5$. Pdf-lineage, along with the standard Trio lineage is sufficient for confidence computation and evaluation of functions for the pdf attributes in the result. We discuss how lineage is used for these two tasks in the next section.

4.2 On-Demand Processing

We now describe how we compute pdf attributes and confidence values in the result data. This processing occurs on-demand after the data processing and lineage generation is complete. Since pdf attributes are represented as functions, the requirement is that we are able to answer any calls to functions representing result pdfs. For any result alternative, we can trace its lineage all the way to the base alternatives from which it is derived, as in [2]. We can similarly build transitive versions of P and M , say P_t and M_t , using a recursive procedure. For example, consider a base tuple t_1 with pdfs A, B , and a result tuple t_2 with pdf-lineage $P = A \leq 5 \wedge B \leq A$ and $M = X1 \rightarrow A, X2 \rightarrow B + 5$, as above. Suppose another query is executed on this result, producing a tuple t_3 with pdf-lineage $P = X2 \geq 10$ and $M = X3 \rightarrow X1 + 5, X4 \rightarrow X2$. The transitive pdf-lineage for the tuple t_3 is $P_t = A \leq 5 \wedge B \leq A \wedge B + 5 \geq 10$ and $M_t = X3 \rightarrow A + 5, X4 \rightarrow B + 5$.

The transitive lineage of a result tuple only points to base alternatives, and all predicates and mappings in P_t and M_t only refer to attributes in the base alternatives along with result placeholders. As in regular Trio, unfolding to the base data is required since the input relations being processed in any query may not be independent, while all base data is independent.

We now discuss four complimentary techniques for on-demand processing: *translation*, *discrete approximation*, *fourier*, and *sampling*.

Translation: Some simple queries permit a very efficient technique that gives precise answers. Consider for example an alternative with $P_t = A \leq 5 \wedge B \geq 10$, and $M_t = X1 \rightarrow A, X2 \rightarrow B$, where A and B are base pdfs and $X1$ and $X2$ are pdfs in the result alternative. The confidence of the result alternative is first computed as described in [2], but may need to be further scaled down because of unevaluated predicates over pdf attributes. For this example, the scaling factor s is computed as:

$$s = \text{weight}_A(-\infty, 5) * \text{weight}_B(10, \infty)$$

The result alternative exists only in possible worlds with values for A and B that satisfy the predicate, and the product is taken because base pdfs are assumed to be independent. Similarly, the *weight* functions for $X1$ and $X2$ are:

$$\text{weight}_{X1}(l, h) = \frac{\text{weight}_A(\min(l, 5), \min(h, 5))}{\text{weight}_A(-\infty, 5)}$$

$$weight_{X_2}(l, h) = \frac{weight_B(\max(l, 10), \max(h, 10))}{weight_B(10, \infty)}$$

This simple technique yields precise and efficient answers. Unfortunately, for somewhat more involved queries, this approach does not work. For example, if P_t has any predicate that involves more than one pdf, such as $A \leq B$, where A and B , we cannot compute confidence values and weight functions by simple translation.

Discrete Approximation: Discrete approximations to pdfs can be used to get approximate answers using a “numerical integration” approach. This technique, though less precise and efficient than the translation technique above, is applicable for a larger class of P_t and M_t . Consider for example an alternative with $P_t = A \leq B$ and $M_t = X1 \rightarrow A$. Let us assume that $weight_A(0, 100)$ is almost 1. A can be approximated using 100 intervals between points $0, \dots, 100$ for this computation. This approximation can be represented using a vector $v = [weight(0, 1), \dots, weight(99, 100)]$. The confidence scaling factor s can be approximated using:

$$s = \sum_{0 \leq i < 100} weight_A(i, i + 1) \cdot weight_B(i, \infty)$$

An approximation to the weight function for the pdf $X1$ in the result can be computed approximately as:

$$weight_{X1}(j, k) = \frac{1}{s} \cdot \sum_{j \leq i \leq k} weight_A(i, i + 1) \cdot weight_B(i, \infty)$$

This example uses a discrete approximation for the *weight* function. It is easy to see that *inverse-cdf* can also be used. There is a trade-off between cost and precision when this method is used: more points in the approximation increases both precision and cost. The predicate $A \leq B$ in the example above is equivalent to the comparison of an arithmetic expression with a constant: $A - B \leq 0$. The discrete approximation technique scales poorly as the number of pdf attributes in the arithmetic expression increases, in terms of both cost and precision. For example, an expression like $\sum_i A_i \leq 0$ with n pdfs can be very inefficient: $O(d^{n-1})$ where d points are used to approximate each pdf. However, it should be noted that the discrete approximation technique is applicable for all queries.

Fourier: Addition and subtraction over pdf attributes is computed using a convolution operation on pdfs. A convolution is a kind of integration problem that can be solved using Fourier transforms, since the convolution operation translates to multiplication in Fourier space. Consider for example a result tuple with $M_t = X1 \rightarrow A + 2B - C$ and an empty P_t , where A, B, C are pdf attributes.

$$fourier_{X1}(x) = fourier_A(x) \cdot 2fourier_B(x) \cdot -fourier_C(x)$$

For base data, these functions may be symbolic making them efficient and precise; or they may use the *weight* function to numerically approximate the Fourier transform yielding a less precise, more expensive, execution. Also, the *weight* function for the result can be computed numerically from the *fourier* function. This technique can be more efficient and precise than discrete approximations, particularly for expressions involving a large number of base pdfs. The limitation of the technique is that it is useful only for addition and subtraction operations over pdf attributes. It needs to be combined with other approaches when the query involves predicates.

Sampling: Random sampling over possible worlds can be used to compute confidences and to answer calls to functions representing the result. We use the *sample* function for this technique. For example, consider a result tuple with pdf-lineage: $P_t = A + B \leq 4$ and $M_t = X1 \rightarrow A + B$. The confidence scaling factor s can be approximated as the fraction of times the following expression is true using a large number of calls:

$$sample_A(-\infty, \infty) + sample_B(-\infty, \infty) \leq 4$$

A call to *sample* on $X1$ can be answered by repeatedly calling *sample* on A and B until the above expression is satisfied, then returning the value of the left-hand side. Sampling, like discrete approximations, is a very general technique that presents a cost precision trade-off: more precise answers are obtained by using more samples. In general, sampling scales better than discrete approximations as the number pdf attributes in expressions increase, in terms of both efficiency and precision.

Specialized Processing: Our approach for pdf attributes makes it easy to use specialized processing for known distributions. For example, consider a result tuple with $M_t = X1 \rightarrow A + B$ and an empty P_t . If both A and B are Gaussians, *describe* on $X1$ can be computed as `[Type: "Gaussian", Parameters: $\mu_A + \mu_B, \sigma_A^2 + \sigma_B^2$]` where $\mu_A, \mu_B, \sigma_A^2, \sigma_B^2$ are obtained using *describe* calls on A, B . Other functions including *weight*, *sample*, *inverse-cdf*, and even *fourier* can be computed efficiently from *describe* for known distributions. The framework allows for adding knowledge about specific distributions to the system in computing the *describe* function. This processing may be limited in scope with regards to complicated queries, but can yield precise answers efficiently when applicable.

Combining: For any interface, function, or confidence computation call, the answer may be produced by combining more than one of the techniques presented above. Combining the techniques results in multiple “query plans”. Choosing a good plan is important, since each evaluation is potentially expensive, and wish to optimize for a combination of precision and efficiency.

5 Discussion

5.1 Correlated Attributes

Correlated attributes in base data are represented through shared functions. If X and Y are correlated, then they share all of their functions. A *weight* call now specifies a l and h on each attribute, i.e., $weight(l_X, h_X, l_Y, h_Y)$. Similarly a call to *sample* requires four arguments, and returns a pair x, y . This representation using shared functions is equivalent to allowing higher dimensional domains for a single attribute, while making all dimensions accessible in queries. Recall we assume that all functions on base data are independent, thus dependent attributes need to share functions.

Correlation across attributes in query results may occur as a side-effect of a query (or a series of queries) as discussed earlier in Section 2.2. The query processor handles these correlations by tracing through to the base data for all aspects of the query that process uncertainty as described in Section 4.

5.2 Interfaces and Functions in Queries

Interfaces and functions may be called directly, and they may be used in queries. For example, a query might refer to $median(A)$ for a pdf attribute A in the **where** or the **select** clause. This call triggers the computation of the interface (or function) which then produces a certain value used by the remainder of query processing.

6 Related Work

A number of systems have been developed recently for managing uncertain data, e.g., [1, 3, 4, 10, 13]. Many of these systems, including Trio, were designed to support discrete uncertainty. The Orion system [11] incorporates continuous uncertainty by using discrete approximations to process queries over a symbolic representation for known distributions. Reference [6] describes a system that uses sampling as the fundamental technique for managing uncertain data, including continuous uncertainty. A probabilistic XML system that manages continuous uncertainty for known distributions is described in [9].

The systems described above are DBMSs targeted specifically for managing uncertain data, usually with possible worlds semantics. There has been another line of work aimed at incorporating continuous variables (sometimes pdfs) into conventional database management. For example, techniques for querying and indexing pdf attributes are discussed in [5, 7], while *FunctionDB* [12] uses a symbolic representation and algebraic query processing for continuous data that can be represented as polynomials.

7 Future Work

This paper proposes a framework for incorporating continuous uncertainty into the Trio system. In doing so, it exposes numerous challenges for managing continuous uncertainty, and uncertain data in general. We identify the following as some interesting directions for future work:

- The cost-precision trade-offs of the proposed techniques need to be investigated both analytically and experimentally.
- As discussed in Section 4.2, combining our proposed techniques yields a space of “plans”, whose selection presents new optimization challenges.
- Functions may be invoked in a query. Thus, It will be interesting to investigate if there are benefits associated with batch-processing these functions.
- The current proposal restricts the query language to have only a conjunctive **where** clause. We would like to allow a more general query language, along with some built-in language and data-type extensions specifically for querying pdfs.
- When pdf attributes are over discrete domains, they capture Trio’s precious model. We can explore how the ideas in this paper may benefit Trio with discrete uncertain data.
- Last and most important, implementation in Trio is the next critical step to determine the viability of our approach.

References

1. L. Antova, C. Koch, and D. Olteanu. MayBMS: Managing Incomplete Information with Probabilistic World-Set Decompositions. In *Proc. of ICDE*, 2007.
2. O. Benjelloun, A. D. Sarma, A. Halevy, M. Theobald, and J. Widom. Databases with uncertainty and lineage. *VLDB Journal*, 17(2), 2008.
3. J. Boulos, N. Dalvi, B. Mandhani, S. Mathur, C. Re, and D. Suciu. MYSTIQ: a system for finding more answers by using probabilities. In *Proc. of ACM SIGMOD*, 2005.
4. A. Deshpande and S. Madden. MauveDB: Supporting Model-based User Views in Database Systems. In *Proc. of ACM SIGMOD*, 2006.
5. A. Faradjian, J. Gehrke, and P. Bonnet. GADT: A Probability Space ADT for Representing and Querying the Physical World. In *Proc. of ICDE*, 2002.
6. R. Jampani, F. Xu, M. Wu, L. L. Perez, C. Jermaine, and P. J. Haas. MCDB: A Monte Carlo approach to managing uncertain data. In *Proc. of ACM SIGMOD*, 2008.
7. V. Ljosa and A. Singh. APLA: Indexing arbitrary probability distributions. In *Proc. of ICDE*, 2007.
8. R. Murthy and J. Widom. Making aggregation work in uncertain and probabilistic databases. In *Proc. of Workshop on Management of Uncertain Data*, 2007.
9. T. Scholte. Managing continuous uncertain data by a probabilistic xml database management system. Technical report, University of Twente, 2008.
10. S. Singh, C. Mayfield, S. Mittal, S. Prabhakar, S. Hambrusch, and R. Shah. Orion 2.0: Native support for uncertain data. In *Proc. of ACM SIGMOD*, 2009.

11. S. Singh, C. Mayfield, R. Shah, S. Prabhakar, S. Hambrusch, J. Neville, and R. Cheng. Database support for probabilistic attributes and tuples. In *Proc. of ICDE*, 2008.
12. A. Thiagarajan and S. Madden. Querying continuous functions in a database system. In *Proc. of ACM SIGMOD*, 2008.
13. J. Widom. Trio: A system for data, uncertainty, and lineage. *Managing and Mining Uncertain Data*, 2009.