Optimizing Probabilistic Query Processing on Continuous Uncertain Data

Liping Peng, Yanlei Diao, Anna Liu†
Department of Computer Science †Department of Mathematics and Statistics
University of Massachusetts, Amherst

ABSTRACT

Uncertain data management is becoming increasingly important in many applications, in particular, in scientific databases and data stream systems. Uncertain data in these new environments is naturally modeled by continuous random variables. An important class of queries uses complex selection and join predicates and requires query answers to be returned if their existence probabilities pass a threshold. In this work, we optimize threshold query processing for continuous uncertain data by (*i*) expediting joins using new indexes on uncertain data, (*ii*) expediting selections by reducing dimensionality of integration and using faster filters, and (*iii*) optimizing a query plan using a dynamic, per-tuple based approach. Evaluation results using real-world data and benchmark queries show the accuracy and efficiency of our techniques and significant performance gains over a state-of-the-art threshold query optimizer.

1. INTRODUCTION

Uncertain data management is becoming increasingly important in a wide range of applications. Much research in the literature has been motivated by traditional applications such as data integration, information extraction, and sensor networks. Recent studies have shown that uncertain data management also plays a key role in large-scale scientific applications including severe weather monitoring [20] and computational astrophysics [17, 18]. The recent initiative to build professional data management and analytics software for scientific research [9] has further confirmed that almost all scientific data that results from real-world measurements is uncertain, and hence capturing uncertainty from data input to query output is a key component of the scientific data management system.

A Motivating Application. Let us consider computational astrophysics for a concrete example. The Sloan Digital Sky Survey (SDSS) benchmark [18] and the ArrayDB benchmark show the following characteristics of the uncertain data management problem:

Continuous uncertain data. Most attributes that resulted from scientific measurements or their data cooking processes are uncertain. These attributes are naturally modeled by continuous random variables. Gaussian distributions are the most commonly used distributions [17, 18] while more complex distributions such as asymmetric and bimodal distributions can be useful in special domains such as tornado detection [20]. As a concrete example, the Galaxy table in the SDSS archive has 297 attributes, out of which 151 attributes are uncertain. (The schema is partially shown in Appendix A.)

Complex selection and join predicates. An important class of queries employs a Select-From-Where SQL block using a wide variety of predicates. Queries Q1 and Q2 from the SDSS benchmark (shown in Appendix A) involve four types of predicates: (*i*) predicates on deterministic attributes, (*ii*) range predicates on a single uncertain attribute, e.g., the first selection predicate in Q1; (*iii*) multivariate linear predicates on uncertain attributes, e.g., the second and third join predicates in Q2; (*iv*) multivariate quadratic predicates on uncertain attributes, e.g., the last selection predicate in Q1 and the last join predicate in Q2. Each query can have an arbitrary mix of these types of predicates.

Efficient processing of threshold queries. Given uncertainty of input data, the user would want to retrieve query answers of high confidence, reflected by high existence probabilities of these answers. A common practice is that the user specifies a threshold so that only those tuples whose existence probabilities pass the threshold are finally returned. In many scenarios, such threshold queries need to be processed efficiently, for instance, in *near real-time* to detect dynamic features, transient events, and anomalous behaviors, or with short delay to support *interactive analysis* where scientists issue explorative queries and wait for quick answers online.

Most work on probabilistic databases models uncertain data using discrete random variables and evaluates queries based on the possible worlds semantics (e.g., [3, 10, 21]). Recent work has argued for using new techniques natural to continuous random variables [17], and showed significant performance gains of such techniques over discretization or Monte Carlo simulation for evaluating relational operators [20] and for ranking [12]. CLARO [19, 20] and ORION [14, 16] are two state-of-the-art systems that provide native support of queries on continuous uncertain data (without using discretization or sampling). CLARO, however, does not consider the threshold in query processing; a naive extension that applies the threshold filter at the end of query processing wastes a lot of computation on nonviable answers. ORION has general evaluation strategies for selection-projection-join queries, but lacks optimizations of queries with complex predicates such as those in Q1 and Q2. Furthermore, its query optimizer uses simple static query plans and results in inefficient execution, which we show later in this paper.

Contributions. In this paper, we address efficient threshold query processing on continuous uncertain data. We support selection-join-projection queries with a threshold on the existence probabilities of query answers. We propose to optimize such query processing

*This work was supported in part by the National Science Foundation under the grants IIS-0746939 and IIS-0812347.

Proceedings of the VLDB Endowment, Vol. 4, No. 11

using a suite of new techniques grounded in statistical theory and a new design of the query optimizer. Our contributions include:

Selections (§3): Selections with complex predicates on continuous uncertain data often involve high-dimensional integrals such as with Q1 and Q2. In this work, we propose optimizations to reduce dimensionality of integration. We further develop fast filters for a wide range of predicates which efficiently compute an upper bound of the probability that a tuple satisfies the predicates. Hence, these filters can be used to prune tuples quickly.

Joins (§4): Joins on continuous uncertain data have traditionally used the strategy of a cross-product followed by a selection. This strategy can be highly inefficient as it may generate a large number of intermediate tuples. A join index can potentially prune many intermediate tuples. However, the design of a join index for continuous uncertain data is challenging because the index not only stores continuous distributions, but also takes a search condition based on the distribution from the probing tuple, which is not deterministic, and returns all candidates that can potentially produce join results that pass the threshold filter. The (only) relevant type of join index [7, 8] is based on primitive statistical results and has limited filtering power. We propose several new indexes based on much stronger statistical results and support a range of join predicates.

Query optimization (§5). Query optimization for continuous uncertain data fundamentally differs from traditional query optimization because selectivity becomes a property of each tuple that carries a distribution. Depending on the attribute distribution, the optimal plan for one tuple can be bad for another tuple. This change dictates per-tuple based query planning. Furthermore, in data stream systems the selectivity of operators cannot be estimated until the tuple arrives, and the selectivity of post-join operators cannot be estimated until the join results with new joint distributions are produced. Hence, selectivity estimation and query planning need to be performed during query execution. We design a query optimizer that supports such dynamic, per-tuple based planning at a low cost.

Evaluation (§6). Using real data and benchmark queries from SDSS, we demonstrate the accuracy and efficiency of our join indexes, selection filters, and optimization technique. Our results further demonstrate remarkable performance gains over the state-of-the-art join indexes [7, 8] and optimizer for threshold queries [14].

2. BACKGROUND

In this section, we present a data model for probabilistic query processing on continuous uncertain data. This model provides a technical context for our discussion in later sections.

Probability distributions. A Gaussian Mixture Model (GMM) describes a probability distribution using a convex combination of Gaussian distributions. A multivariate Gaussian Mixture Model (multivariate GMM) naturally follows from the definition of multivariate Gaussian distributions. We include their definitions in Appendix B. GMMs offer two key benefits to uncertain data management: First, theoretical results have shown that GMMs can approximate any continuous distribution arbitrarily well [11]. Hence, they are suitable for modeling complex real-world distributions. Second, GMMs allow efficient computation of relational operators based on Gaussian properties and advanced statistical theory as shown in our prior work [20] and later sections in this paper.

Data model. We consider an input data set that follows 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, $\mathbf{A}^p$ is modeled by a vector of continuous random variables, $\mathbf{X}$, that has a joint pdf, $f_{\mathbf{A}^p}(\mathbf{x})$. According to the schema, $\mathbf{A}^p$ can be partitioned into independent groups of correlated attributes. Each group of correlated attributes can be modeled by a (multivariate) GMM denoted by $f_j(\mathbf{x}_j)$. Then the joint distribution for $\mathbf{A}^p$ can be written as $\prod_j f_j(\mathbf{x}_j)$. For simplicity, we use $\mathbf{A}$ to refer to uncertain attributes when our discussion focuses on uncertain attributes only. This model can be extended to address inter-tuple correlation by leveraging existing work [16] (for details, see Appendix B).

3. OPTIMIZING THRESHOLD SELECTION

In this section, we consider probabilistic threshold selections over a relation on a set of continuous uncertain attributes. Our goal is to support efficient evaluation of such selections, especially when they contain complex predicates.

Definition 3.1 (Probabilistic Threshold Selection) *A probabilistic threshold selection, $\sigma_{\theta,\lambda}$, over a relation T is defined as:*

$$\sigma_{\theta,\lambda}(T) = \{t \mid Pr[R_\theta(\mathbf{X})] \geq \lambda, t \in T\},$$

where θ is the selection condition on continuous uncertain attributes $\mathbf{A}$, λ is the probability threshold, and R_θ is the selection region defined as $\{\mathbf{a} | \mathbf{a} \in \mathbb{R}^{|\mathbf{A}|} \wedge \theta(\mathbf{a}) = true\}$. For each tuple t, $\mathbf{X}$ is the random vector for $t.\mathbf{A}$, and $Pr[R_\theta(\mathbf{X})]$ is the probability for $\mathbf{X}$ to satisfy the selection condition, i.e., $Pr[R_\theta(\mathbf{X})] = \int_{R_\theta} f_{\mathbf{X}}(\mathbf{x})d\mathbf{x}$.

A basic evaluation strategy for the selection follows the definition above, using an integral of the joint attribute distribution $f_{\mathbf{X}}$ for each tuple. For instance, the WHERE clause in Q2 (in Appendix A) specifies a condition θ involving ten uncertain attributes. Assume that a cross-product is first performed. Then the selection with condition θ involves a ten-dimensional integral for each tuple.

An improvement is to factorize this integral into lower dimension integrals based on independence [16]. Suppose that the schema indicates that the uncertain attribute set $\mathbf{A}$ can be partitioned into attribute groups that are independent of each other. Denote this partitioning using a set system $S = \{\mathbf{A}_1, \mathbf{A}_2, \ldots \mathbf{A}_G\}$. Then consider each predicate in the condition θ: if the predicate involves attributes from different groups, merge these groups into one. After doing so for all predicates, we obtain a new set system $S' = \{\mathbf{A}'_1, \mathbf{A}'_2, \ldots \mathbf{A}'_{G'}\}$. Now we can rewrite the big integral as the product of the integrals for the attribute groups in S'. Revisit Q2. The SDSS schema shows that $S = \{\{G_1.u\}, \{G_1.g\}, \{G_1.r\}, \{G_1.rowc, G_1.colc\}, \{G_2.u\}, \{G_2.g\}, \{G_2.r\}, \{G_2.rowc, G_2.colc\}\}$. The predicates in WHERE yields $S' = \{\{G_1.u, G_1.g, G_1.r, G_2.u, G_2.g, G_2.r\}, \{G_1.rowc, G_1.colc, G_2.rowc, G_2.colc\}\}$. Hence for each tuple, we will perform a 6-dimensional integral plus a 4-dimensional integral.

As can be seen, the basic evaluation approach can be expensive or even intractable when the integral has a high dimensionality and a complex shape of the selection region. In this section, we propose two classes of optimization techniques grounded in statistical theory: the first class reduces the dimensionality of integration, while the second class efficiently filters (most of) tuples whose probabilities fall below the threshold without using integrals.

3.1 Reducing Dimensionality of Integration

We first propose to reduce the dimensionality of integration by leveraging the following result [15]:

Linear Transformation: Let $\mathbf{X} \sim N_k(\boldsymbol{\mu}, \Sigma)$. For a given $l \times k$ matrix $\mathbf{B}$ of constants and a l-dimensional vector $\mathbf{b}$ of constants, $\mathbf{Y} = \mathbf{BX} + \mathbf{b} \sim N_l(\mathbf{B}\boldsymbol{\mu} + \mathbf{b}, \mathbf{B}\Sigma\mathbf{B}^T)$.

The result states that a linear transformation of multivariate normal random vector still has a multivariate normal distribution. It is natural to extend linear transformation to a GMM: we simply perform a linear transformation of each mixture component separately.

To apply the above result, given a selection condition θ, we define a transformed selection region $R'_\theta = \{\mathbf{y} | \mathbf{y}{=}\mathbf{B}\mathbf{x}{+}\mathbf{b} \wedge \mathbf{x}{\in}R_\theta\}$. If there exists a transformation matrix $\mathbf{B}_{l\times k}$ ($l<k$) such that $Pr[R_\theta(\mathbf{X})] = Pr[R'_\theta(\mathbf{Y})]$, we can reduce the dimensionality of integration, i.e.,

$$\int_{R_\theta} f_{\mathbf{X}}(\mathbf{x})\,\mathrm{d}\mathbf{x} = Pr[R_\theta(\mathbf{X})] = Pr[R'_\theta(\mathbf{Y})] = \int_{R'_\theta} f_{\mathbf{Y}}(\mathbf{y})\,\mathrm{d}\mathbf{y}. \quad (1)$$

Given a condition θ on a set of continuous uncertain attributes, we can construct $\mathbf{B}$ and $\mathbf{b}$ by taking the following steps: (1) Partition attributes into independent groups based on the schema and θ, as described at the beginning of the section. (2) For each group of attributes, define a random vector $\mathbf{X}$. Find maximum linear subexpressions relevant to $\mathbf{X}$ from θ, and denote them using a new vector $\mathbf{Y}$. Rewrite each variable in $\mathbf{Y}$ as the product of a row vector and $\mathbf{X}$, plus a constant. Let $\mathbf{B}$ be the matrix that contains all the row vectors and $\mathbf{b}$ be the column vector that contains all the constants. (3) If $\mathbf{B}$ does not have full row rank, remove rows from $\mathbf{B}$, one at a time, until it has full row rank. Remove elements from $\mathbf{b}$ accordingly. The correctness of the procedure is shown in Appendix C.

Denote the matrix returned as $\mathbf{B}_{l\times k}$. Since it has full row rank, $l \le k$. If $l < k$, we can apply Eq. (1) to transform the integration from the space for $\mathbf{X}$ to that for $\mathbf{Y}$; If $l = k$, linear transformation does not help to reduce the dimensionality of integration.

Example 3.1 *For Q2, we obtain two independent groups of attributes after step* (1)*, which is the factorization described at the beginning of the section. In step* (2)*, let us consider the first group:* $S'_1 = \{G_1.u,\ G_1.g,\ G_1.r, G_2.u,\ G_2.g, G_2.r\}$. *Let* $\mathbf{X}$ *be the random vector for* S'_1. *There are two maximum linear subexpressions in* θ *for* S'_1. *Let* $y_1 = (G1.u - G1.g) - (G2.u - G2.g)$, $y_2 = (G1.g - G1.r) - (G2.g - G2.r)$. *Then, we have*:

$$\mathbf{Y} = \begin{pmatrix} y_1 \\ y_2 \end{pmatrix} = \begin{pmatrix} 1 & -1 & 0 & -1 & 1 & 0 \\ 0 & 1 & -1 & 0 & -1 & 1 \end{pmatrix} \mathbf{X} + \begin{pmatrix} 0 \\ 0 \end{pmatrix} = \mathbf{B}\mathbf{X} + \mathbf{b}$$

Since $\mathbf{B}$ *has full row rank, step* (3) *is omitted. We can get the pdf of* $\mathbf{Y}$ *using linear transformation from* $\mathbf{X}$. *Finally based on Eq* (1)*, the integral dimensionality is reduced from 6 to 2:*

$$\int_{R_\theta} \cdots \int \prod_{i=1}^{6} (f_{X_i}(x_i)\,\mathrm{d}x_i) = \int_{-0.05}^{0.05} \int_{-0.05}^{0.05} f_{\mathbf{Y}}(y_1, y_2)\,\mathrm{d}y_1\,\mathrm{d}y_2$$

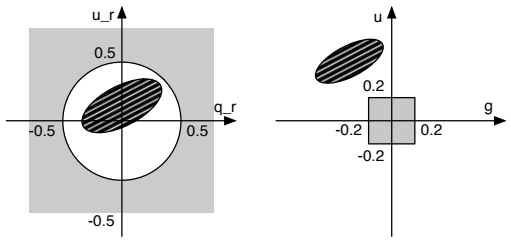
3.2 A Filtering Framework without Integrals

Although linear transformation can improve performance by decreasing the dimensionality of integration, it still requires the use of one integral for each tuple. In this section, we propose a filter operator, $\tilde{\sigma}$, that computes an upper bound ($\tilde{p}$) of the true probability (p) that a tuple satisfies the selection condition without using integrals. When such upper bounds are tight enough, most tuples that fail to pass the threshold selection can be removed by the filter $\tilde{p} < \lambda$. In (rare) cases that $p < \lambda \le \tilde{p}$, the original selection operator (σ) with exact integration is needed to compute the true probability. Hence, a key issue in designing the filter is how to derive a tight upper bound at a cost much lower than the integration cost.

3.2.1 A General Filtering Technique

We first propose a general filtering technique that leverages the *multidimensional Chebyshev's inequality*, and explores its relationship with a selection region in a high-dimensional space. Let $\mathbf{X}$ be a k-dimensional random vector with expectation $\boldsymbol{\mu}$ and covariance matrix Σ. If Σ is an invertible matrix, then for any real number $a > 0$, multidimensional Chebyshev's inequality states that:

$$Pr[(\mathbf{X}-\boldsymbol{\mu})^T\Sigma^{-1}(\mathbf{X}-\boldsymbol{\mu}) > a^2] \le \frac{k}{a^2}. \quad (2)$$



(a) $q\_r^2+u\_r^2>0.25$ (b) $|u|<0.2$ and $|g|<0.2$

Figure 1: Illustration of the predicate region (shaded in gray) and a tuple's chebyshev region (shaded in stripes).

To leverage the above result, we transform threshold selection evaluation into a geometric problem. Besides the predicate region $R_\theta \subseteq \mathbb{R}^k$ from Definition 3.1, we also define a geometric region specific to each given random vector $\mathbf{X}$ of size k and a threshold λ:

Definition 3.2 (Chebyshev region) *A Chebyshev region* $R_\lambda(\mathbf{X})$ *for a given* λ *and a random vector* $\mathbf{X}$ *with mean* $\boldsymbol{\mu}$ *and variance* Σ *is:*

$$R_\lambda(\mathbf{X}) = \{\mathbf{x} \,|\, (\mathbf{x}-\boldsymbol{\mu})^T\Sigma^{-1}(\mathbf{x}-\boldsymbol{\mu}) < \frac{k}{\lambda}\}. \quad (3)$$

Geometrically, the Chebyshev region is an ellipse (in $\mathbb{R}^2$) or ellipsoid (in $\mathbb{R}^k$ where $k \ge 3$) centered at $\boldsymbol{\mu}$. According to Eq. (2), we can see that $Pr[R_\lambda(\mathbf{X})] > 1-\lambda$. That is, for the random vector $\mathbf{X}$, the Chebyshev region covers the probability mass of more than $1-\lambda$. Therefore, when the Chebyshev region $R_\lambda(\mathbf{X})$ for a given tuple does not overlap with the predicate region R_θ, it is easy to bound the probability mass of this tuple in the predicate region: $Pr[R_\theta(\mathbf{X})] \le 1 - Pr[R_\lambda(\mathbf{X})] < \lambda$. We can then safely filter the tuple. As such, the threshold selection problem is transformed into the geometric problem of judging whether the predicate region and a tuple's Chebyshev region are disjoint in a k-dimensional space.

Example 3.2 *For a bivariate random vector* $\mathbf{X}$ *with mean* $\boldsymbol{\mu}$ *and covariance* Σ*, the Chebyshev region is a region bounded by an ellipse centering at* $\boldsymbol{\mu}$*, shown as the areas shaded in stripes in Fig. 1(a) and Fig. 1(b). The predicate "*$q\_r^2+u\_r^2>0.25$*" in Q1 is marked by the grey area outside the circle with center* $(0,0)$ *and radius* 0.5 *in Fig. 1(a). The predicate region of "*$|u| < 0.2$ *and* $|g|<0.2$*" is a square shown by the grey area in Fig. 1(b).*

Detecting disjoint regions. The R_θ and $R_\lambda(\mathbf{X})$ regions are disjoint in the space $\mathbb{R}^k$ if they satisfy two conditions: (1) the center of $R_\lambda(\mathbf{X})$, which is $\boldsymbol{\mu}$, falls outside of R_θ; (2) the boundary of R_θ is outside $R_\lambda(\mathbf{X})$. When R_θ has a simple shape, e.g., a rectangle as shown in Fig. 1(b), condition (2) is satisfied if none of the edges intersects with the boundary of $R_\lambda(\mathbf{X})$ and the center of R_θ lies outside $R_\lambda(\mathbf{X})$. Generally, to test condition (2), we can minimize $(\mathbf{X}-\boldsymbol{\mu})^T\Sigma^{-1}(\mathbf{X}-\boldsymbol{\mu})$ on R_θ. If the minimum is larger than k/λ, condition (2) is satisfied. While constrained optimization in general can be a difficult problem, in many common cases it can be solved efficiently. For example, when the region R_θ is an intersection of ellipsoids, the constrained optimization becomes the so called Quadratically Constrained Quadratic Program (QCQP), which can be solved as easily as the linear programs [5]. When the boundary of R_θ can be readily defined by equalities, the minimization can be done on the boundary and solved using the *Lagrange multiplier*. For example, when R_θ is an ellipsoid, the Lagrange multiplier leads to a system of linear equations.

In summary, for those common predicates whose regions are of simple shapes or whose boundary can be defined by linear or quadratic equalities, our general technique based on the multidimensional Chebyshev's inequality can provide efficient filtering.

3.2.2 Fast Filters for Common Predicates

For several common types of predicates, we can devise fast filters for threshold selection evaluation by exploiting known statistical results. Consider the following predicates on the attribute set $\mathbf{A}$:

(1) *One-dimensional*: $|\mathbf{A}|=1$, $R_\theta=\bigcup_{i=1}^{n}(a_i,b_i)$, where $a_i>b_{i-1}$. An example is "$ra^2>1$", whose selection region can be written as $(-\infty,-1)\cup(1,+\infty)$.

(2) *Multi-dimensional quadratic forms*: $|\mathbf{A}|>1$, $R_\theta=\{\mathbf{a}|\mathbf{a}^T\Lambda\mathbf{a}\ op\ \delta\}$, where Λ is an $|\mathbf{A}|$-dimensional symmetric matrix, op is "$>$" or "$<$". An example is "$q\_r^2+u\_r^2>0.25$" in Q1, where Λ is the identity matrix.

(3) Predicates that can be reduced to category (1) or (2) above by applying linear transformation. Consider "$|(G_1.u-G_1.g)-(G_2.u-G_2.g)|<0.05$" in Q2. By letting $z=(G_1.u-G_1.g)-(G_2.u-G_2.g)$, we have "$|z|<0.05$", which belongs to category (1). Next consider "$(G_1.rowc-G_2.rowc)^2+(G_1.colc-G_2.colc)^2<4E6$" in Q2. With $z_1=G_1.rowc-G_2.rowc$ and $z_2=G_1.colc-G_2.colc$, we have "$z_1^2+z_2^2<4E6$", which belongs to category (2).

One-dimensional predicates. We exploit the following statistical results to devise fast filters.

Markov's inequality states that for a random variable X, $Pr[|X|\geq a]\leq E|X|/a$ for any real number $a>0$. It can be applied to predicates $R_\theta=\bigcup_{i=1}^{n}(a_i,b_i)$, if the point 0 does not lie in the predicate region (otherwise we will get a trivial upper bound of value 1).

Chebyshev's inequality and Cantelli's inequality: Chebyshev's inequality states that for a random variable X with expected value μ and standard deviation σ, $Pr[|X-\mu|\geq a\sigma]\leq 1/a^2$ for any real number $a>0$. Cantelli's inequality, known as the one-sided version of Chebyshev's inequality, provides a tighter bound on each side of the distribution: $Pr[X\geq\mu+a\sigma]\leq 1/(1+a^2)$, $Pr[X\leq\mu-a\sigma]\leq 1/(1+a^2)$. Both inequalities can be applied to predicates $\bigcup_{i=1}^{n}(a_i,b_i)$ if μ does not reside in the predicate region.

We derive upper bounds using the above three inequalities for predicates in category (1), as listed in Appendix C.

Multi-dimensional quadratic forms. If $\mathbf{X}$ follows a GMM, its quadratic form $\mathbf{X}^T\Lambda\mathbf{X}$ yields a new random variable for which we can compute the mean and variance. This allows us to apply Chebyshev's inequality and Cantelli's inequality similarly as above. See Appendix C for the details of the upper bounds.

4. OPTIMIZING THRESHOLD JOIN

In this section, we consider probabilistic threshold joins of relation R and relation S on a set of continuous uncertain join attributes.

Definition 4.1 (Probabilistic Threshold Join) *A probabilistic threshold join of R and S on continuous uncertain attributes $\mathbf{A}$ is:*

$$R\bowtie_{\theta,\lambda}S=\{(r,s)\,|\,Pr[R_\theta(\mathbf{X}_r,\mathbf{X}_s)]\geq\lambda, r\in R, s\in S\},$$

where θ is the join predicate, λ is the probability threshold, $\mathbf{X}_r$ and $\mathbf{X}_s$ are the random vectors for $r.\mathbf{A}$ and $s.\mathbf{A}$, R_θ is the predicate region in $\mathbb{R}^{2|\mathbf{A}|}$, and $Pr[R_\theta(\mathbf{X}_r,\mathbf{X}_s)]$ is the probability for $\mathbf{X}_r$ and $\mathbf{X}_s$ to satisfy the join condition.

When the input relations R and S are independent, $Pr[R_\theta(\mathbf{X}_r,\mathbf{X}_s)]=\iint_{R_\theta}f_{\mathbf{X}_r}(\mathbf{x}_r)f_{\mathbf{X}_s}(\mathbf{x}_s)\mathrm{d}\mathbf{x}_r\mathrm{d}\mathbf{x}_s$. If R and S tuples are correlated, we can compute the joint distribution using history [16].

A default evaluation strategy for the threshold join $R\bowtie_{\theta,\lambda}S$ is to perform a cross-product $R\times S$ followed by a threshold selection with the condition $\theta(R.\mathbf{A},S.\mathbf{A})$ and the threshold λ. The cross-product can create a large number of intermediate tuples, hence highly inefficient. In this section, we propose new join indexes to implement a *filtered cross-product*, denoted by $R\times_{\theta,\lambda}S$, which returns a superset of true join results but a subset of the cross-product results. Then $R\bowtie_{\theta,\lambda}S=\sigma_{\theta,\lambda}(R\times_{\theta,\lambda}S)$, that is, the true join results are produced by further applying a threshold selection.

Designing a join index for continuous random variables is much more difficult than its counterpart for deterministic values. Consider $R\bowtie_{R.A-S.A<\delta}S$, and we want to build an index on S. First, the design of the join index needs to answer two questions: (1) what is the search key of the index? (2) given a R tuple, how do we form a query region over the index? In a traditional database, $S.A$ has a deterministic value and naturally forms the search key of the index. Given a R tuple, the join predicate is instantiated with $R.A=v$, which naturally yields a query region, $S.A>v-\delta$, on the index. Now consider the join where $R.A$ and $S.A$ are random variables and each follows a distribution. To build an index on $S.A$, it is not clear which aspects of the distribution of $S.A$ can be used as the index key, and given an R tuple, how we use the distribution of $R.A$ to form the query region over the index.

Second, the index for probabilistic threshold join needs to take into account the probability threshold λ. For each tuple r in R, probing the index should return all those tuples s in S that can possibly satisfy $Pr[R_\theta(\mathbf{X}_r,\mathbf{X}_s)]\geq\lambda$, called *candidate tuples*. In other words, we want to ignore other tuples $\tilde{s}$ for which we know for sure $Pr[R_\theta(\mathbf{X}_r,\mathbf{X}_{\tilde{s}})]<\lambda$, hence improving performance.

Our main idea is that if we can find a necessary condition for $Pr[R_\theta(\mathbf{X}_r,\mathbf{X}_s)]\geq\lambda$, then the negation of the necessary condition identifies all those tuples $\tilde{s}$ that can be ignored in the index lookup. To improve the index's filtering power, we seek necessary and sufficient conditions if possible, or necessary conditions that are "tight" enough. Furthermore, to have real utility for index design, the necessary condition has to meet two requirements: (1) In the necessary condition, the quantities concerning S can be used to form the search key of a common index structure such as an R-tree [2]; (2) Given an R tuple, after the necessary condition is instantiated with all the quantities concerning R, it should yield a query region that can be easily tested for overlap with the index entries.

Existing join indexes for continuous uncertain attributes [7, 8] make simplifying assumptions about attribute distributions, and use a "loose" necessary condition in index design, resulting in poor performance as we will show in §6. Below we derive tighter necessary conditions for common join predicates, including a necessary and sufficient condition, and develop new indexes based on them.

4.1 Band Join of General Distributions

We start with the simple case of a single join attribute. A band join uses the predicate "$a<R.A-S.A<b$". Given a tuple r from relation R, we use X_r to denote the random variable of its join attribute, which follows a univariate GMM. Similarly, X_s denotes the random variable for the join attribute in an s tuple, again following a GMM. We denote the mean, variance, and *pdf* of X_t using μ_t, σ_t^2, $f_{\mu_t,\sigma_t^2}(x_t)$, respectively $(t=r,s)$.

A necessary condition. As shown in §3.1, the linear transformation $Z=X_r-X_s$ can transform the original band-shaped integral region into a single interval:

$$Pr[a<Z=(X_r-X_s)<b]=\int_a^b f_{\mu_r-\mu_s,\sigma_r^2+\sigma_s^2}(z)\,\mathrm{d}z.$$

For a single variable Z, the following theorem provides a necessary condition for $Pr[a<Z<b]\geq\lambda$. Our proof is based on Cantelli's inequality as shown in Appendix D.1.

Theorem 1 *Given a range $[a,b]$, if a random variable Z with mean μ and variance σ^2 satisfies the condition $Pr[a<Z<b]\geq\lambda$, then $\mu+\sigma\sqrt{(1-\lambda)/\lambda}\geq a$ and $\mu-\sigma\sqrt{(1-\lambda)/\lambda}\leq b$.*

Since $Z = X_r - X_s \sim N(\mu, \sigma^2)$, plug $\mu = \mu_r - \mu_s$, $\sigma^2 = \sigma_r^2 + \sigma_s^2$ back to the inequalities in the theorem. Then we obtain a necessary condition for $Pr[a < X_r - X_s < b] \geq \lambda$:

$$\mu_r - \mu_s + \sqrt{\frac{1-\lambda}{\lambda}(\sigma_r^2 + \sigma_s^2)} \geq a, \tag{4a}$$

$$\mu_r - \mu_s - \sqrt{\frac{1-\lambda}{\lambda}(\sigma_r^2 + \sigma_s^2)} \leq b. \tag{4b}$$

Index construction and retrieval. We now design an index on the S relation to provide efficient support for the filtered cross product $R \times_{a<R.A-S.A<b,\,\lambda} S$. In Eq. (4a) and (4b), μ_s and σ_s^2 are the quantities from relation S. We use them to form the *search key* of an index: for each tuple s, we insert the pair (μ_s, σ_s^2) together with the tuple id into an R-tree index [2]. This index essentially indexes points in a two-dimensional space in the leaf nodes and groups them into minimum bounding rectangles in non-leaf nodes. All existing R-tree construction methods can be used.

For each probing tuple r, the *query region* is naturally formed by instantiating μ_r and σ_r^2 in Eq. (4a) and (4b). However, this query region has a nonstandard shape, so we re-implement the *overlap* method in the R-tree, which returns True when a minimum bounding rectangle in a tree node, denoted by R_I, overlaps with the query region, denoted by R_Q. Let (x, y) denote the search key of the index, that is, $x = \mu_s$ and $y = \sigma_s^2$. Then R_I is a rectangle $[x_1, x_2; y_1, y_2]$. The query region R_Q has two conditions. By setting $x = \mu_s$ and $y = \sigma_s^2$ in Eq. (4a), we can rewrite the first condition as:

$$R_{Q1}\text{: } (1) x \leq \mu_r - a, \text{ or}$$
$$(2) x > \mu_r - a \text{ and } y \geq \lambda(x - \mu_r + a)^2/(1-\lambda) - \sigma_r^2.$$

It is not hard to see that R_I overlaps with the union of region (1) and region (2) in R_{Q1} if its upper left vertex (x_1, y_2) lies in either region. We can rewrite the second condition from Eq. (4b) and develop the test condition in a similar way.

4.2 Band Join of Gaussian Distributions

We next consider the most common distributions, Gaussian distributions, for continuous random variables. The known Gaussian properties allow us to find **a sufficient and necessary condition** and hence design an index with better filtering power.

Theorem 2 *Given a range $[a, b]$, a normally distributed random variable $Z \sim N(\mu, \sigma^2)$ satisfies the condition $Pr[a < Z < b] \geq \lambda$ **iff** there exists an $\alpha \in (0, 1-\lambda)$ such that $a - \Phi^{-1}(\alpha)\sigma \leq \mu \leq b - \Phi^{-1}(\lambda + \alpha)\sigma$, where Φ^{-1} is the inverse of the standard normal cdf (also called the quantile function).*

The proof is given in Appendix D.1. Given a range $[a, b]$ and a threshold λ, Theorem 2 essentially identifies all normally distributed random variables, i.e., all the (μ, σ) pairs, that satisfy $Pr[a < Z < b] \geq \lambda$. Let Ω denote this collection of (μ, σ). Formally,

$$\Omega(a, b, \lambda) = \bigcup_{0 \leq \alpha \leq 1-\lambda} \left\{ (\mu, \sigma) \mid a - \Phi^{-1}(\alpha)\sigma \leq \mu \leq b - \Phi^{-1}(\lambda + \alpha)\sigma \right\} \tag{5}$$

The Ω region will play a key role in the index design, in particular, representing the query region. Fig. 4 in the appendix shows the shape of Ω when $\lambda = 0.7$, where the x and y axes denote μ and σ, respectively. In general, λ controls the shape of Ω, and the a and b values determine the stretch along both dimensions.

Index construction and retrieval. We next present a new index that exploits the above sufficient and necessary condition and thus returns only the true matches for each probing tuple. Recall that the join predicate is "$a < R.A - S.A < b$", and X_r and X_s denote the join attribute of a r tuple and an s tuple. As in Section 4.1, we build an R-tree index on $S.A$: we take the mean μ_s and variance σ_s^2 of the variable X_s for each tuple and insert them as a pair to the R-tree.

Given each probing tuple r, we next design the query region over the R-tree. As before, consider two variables X_r and X_s, and let $Z = X_r - X_s$. Eq. (5) has defined all possible distributions of Z that would satisfy $Pr[a < Z < b] \geq \lambda$. Now plug $\mu = \mu_r - \mu_s$ and $\sigma = \sigma_r^2 + \sigma_s^2$ into Eq. (5). Since for a particular probing tuple r, μ_r and σ_r are simply constants, Eq. (5) naturally yields a query region over all the distributions (μ_s, σ_s^2) in the R-tree.

Given the query region, the next task is to design the "overlap" routine that directs the search in the R-tree by comparing the query region (R_Q) with the minimum bounding rectangles (R_I) in each non-leaf node of the tree. However, the above query region has a complex shape and hence it is slow to test the overlap between R_Q and R_I. The first technique we use is to transform both R_Q and R_I to a different domain through a mapping. Letting $(x, y) = (\mu_s, \sigma_s^2)$ be the search key of the index, the mapping $\mathcal{F}$ has:

$$x' = \mu_r - x \text{ and } y' = \sqrt{\sigma_r^2 + y}$$

Each rectangle in the index $R_I = [x_1, x_2; y_1, y_2]$ is transformed to:

$$R_I' = \left[u_r - x_2, u_r - x_1; \sqrt{\sigma_r^2 + y_1}, \sqrt{\sigma_r^2 + y_2} \right]$$

Finally, the query region becomes:

$$R_Q' = \bigcup_{0 \leq \alpha \leq 1-\lambda} \left\{ (x', y') \mid a - \Phi^{-1}(\alpha) y' \leq x' \leq b - \Phi^{-1}(\lambda + \alpha) y' \right\},$$

which is exactly Ω. It is also known R_I and R_Q overlap if and only if R_I' and R_Q' overlap because $\mathcal{F}$ is a one-to-one mapping.

Now our task becomes testing the overlap between R_I' and $R_Q' = \Omega$. The overlap test involves several steps based on results of a detailed mathematical analysis. Due to space constraints, this analysis is described in Appendix D.2.

Other types of join indexes. We also provide join indexes for (1) band joins of multivariate GMMs, (2) other joins using linear predicates, and (3) proximity joins using Euclidean distance. Due to space constraints, we leave the details to Appendix D.3.

5. PER-TUPLE BASED PLANNING

We next discuss threshold query processing that takes a selection-join-projection query and returns tuples that satisfy the query with a probability over the threshold λ (i.e., their tuple existence probabilities $> \lambda$). A naive approach would be to perform probabilistic query processing as in earlier work and then apply the threshold filter at the end of the processing, wasting a lot of computation on nonviable answers. To prune nonviable answers early, we push the threshold λ earlier to each relational operator in the query plan, and apply our techniques from the previous sections as follows:

$$R \bowtie_{\theta,\lambda} S = \sigma_{\theta,\lambda}(R \times_{\theta,\lambda} S), \quad \sigma_{\theta,\lambda}(T) = \sigma_{\theta,\lambda}(\tilde{\sigma}_{\theta,\lambda}(T)),$$

where $R \times_{\theta,\lambda} S$ is the filtered cross product using a join index (§4), $\tilde{\sigma}_{\theta,\lambda}(T)$ is the fast filter that prunes tuples with a relaxed condition (§3.2), and $\sigma_{\theta,\lambda}(T)$ is the exact selection that evaluates the condition using integrals but possibly with reduced dimensionality (§3.1). For continuous uncertain attributes, projections do not involve duplicate elimination because there does not exist an *finite* set of values to project onto. Hence, projections do not change tuple existence probabilities and are not further discussed in this work.

Besides our techniques for joins and selections separately, there remains a query optimization issue: What is the most efficient way to arrange filtered cross products, fast filters for selections, and exact selections in a query plan? We consider both the cost and

Tuple				Selectivity	
id	r	q_r	u_r	r < 24	$q\_r^2 + u\_r^2 > 0.25$
1	$N(27.0, 2.2)$	$N(1.2, 2.2)$	$N(0.1, 1.1)$	0.08	0.95
2	$N(21.6, 0.1)$	$N(0.1, 0.1)$	$N(-0.1, 0.1)$	1	1.74×10^{-4}

Table 1: Illustration of per-tuple based selectivity with Q1 and two tuples: each tuple has three normally distributed attributes, r, q_r, and u_r; for each predicate in Q1, these tuples have different selectivities.

selectivity of operators as in a traditional query optimizer. However, several key differences exist in the new context: (1) Due to the use of integrals, exact selections can have high costs and should be treated as "expensive predicates". (2) The selectivity of an operator captures its filtering power on the input data. Under attribute uncertainty, *selectivity needs to be defined on a per-tuple basis*. (3) The above property further implies that the optimal order of evaluating operators also varies on a per-tuple basis.

Example 5.1 *Consider Q1 and two tuples t_1, t_2 in Table 1. For predicate θ_1 : "$r < 24$", let X_r^1 and X_r^2 be the random variables for $t_1.r$ and $t_2.r$ correspondingly. $X_r^1 \sim N(27, 2.2)$, so $Pr[X_r^1 < 24] = 0.08$; $X_r^2 \sim N(21.6, 0.1)$ and $Pr[X_r^2 < 24] \approx 1$. So t_1 has a much lower probability of satisfying θ_1, hence more likely to be filtered. To the contrary, for predicate θ_2 : "$q\_r^2 + u\_r^2 > 0.25$", t_2 has a much lower probability to pass θ_2 than t_1. Thus, the optimal evaluation order is θ_1 followed by θ_2 for t_1, and the reverse for t_2.*

Due to the above reasons, we advocate a *per-tuple, dynamic* query optimization approach with the following features: (1) A query plan is determined for each tuple rather than a whole set. (2) The query plan arranges all operators based on both cost and selectivity. (3) Such planning is performed at a low cost for each tuple. Traditional query optimizers consider a static query plan for a set of tuples [6], hence not suitable for our problem. Data stream systems [1, 4] can adapt query plans dynamically but only estimate selectivity for a set of tuples and lack support of uncertain attributes. Recent work on probabilistic threshold query optimization establishes algebraic equivalence for query optimization, but still uses static query plans and further ignores operator costs in query planning [14].

In the rest of this section, we detail our new query optimization approach. We focus on the data stream setting: like in existing systems [1], some streams can have indexes built on while other streams are used to probe these indexes. We assume that the decision of which indexes to build has been made separately and focus on query optimization only. Our approach can be applied to stored data by viewing the result of a file scan as a data stream.

To begin with, we define the selectivity, denoted by Γ, of a selection on each tuple t and the selectivity of a filtered cross product between a probing tuple t and a set S:

$$\Gamma^{\sigma}_{\theta,\lambda}(t) = Pr[R_\theta(\mathbf{X^t})], \quad \Gamma^{\times}_{\theta,\lambda}(t, S) = \frac{\text{num. true matches from } S}{|S|}$$

Query optimization requires the knowledge of both cost and selectivity of each operator. Our approach combines offline measurements of unit operation costs, which depend only on the types of predicates, and online selectivity estimation, which depends on the attribute distribution in each tuple. These techniques are fairly straightforward and hence left to Appedix E.

5.1 Online Query Planning and Execution

In our approach, online query planning and execution for each tuple interleaves selectivity estimation and ordering of operators in iterations. This is because while we can estimate the selectivity of predicates on a base tuple *r*, we cannot estimate the selectivity of the join predicate on *r* and *s* until the tuple including *r* and *s* is produced with the new joint attribute distribution. Moreover, we cannot afford

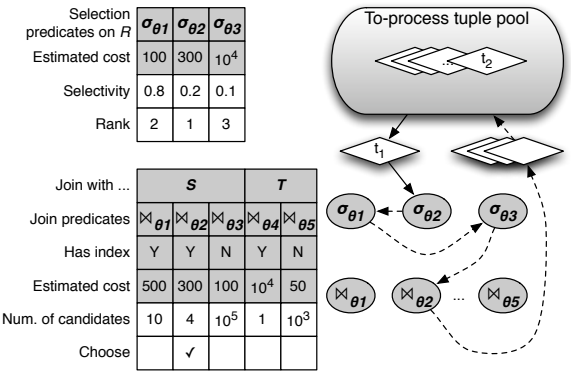


Figure 2: Illustration of tuple-based query planning.

to perform an exhaustive search of the global optimal plan as in earlier work [6] due to per-tuple based planning. Therefore, we break a query into several blocks that each involve at most one join. For each query block, we repeat the following steps:

Step 1: Estimate selectivities of selections. We take all predicates specified on an input tuple t, and group these predicates into independent groups as described at the beginning of §3. For each independent group of predicates θ_i, we then allocate a selection operator $\sigma_i(t)$. We estimate the selectivity of each selection by taking the average of its lower and upper bounds.

Step 2: Rank and execute filters and selections. We expand each selection with all possible filters: $\sigma_i(t) = \sigma_i(\tilde{\sigma}_{ij}(\cdots\tilde{\sigma}_{i1}(t)))$. If there exist fast filters based on known statistical inequalities, we apply all of them as they have negligible costs; otherwise, we apply the filter using constraint optimization. We rank filters and selections in ascending order of selectivity over cost. Filters are ranked before the corresponding selection if they have a lower cost; otherwise, they are unnecessary and should be removed from the plan.

Then we execute the filters and selections in order. The tuple starts with the existence probability $E_p = 1$ and a query threshold λ_q. A selection with the predicate θ reduces the tuple existence probability to $E'_p = E_p \cdot Pr[R_\theta(X^t)]$. A filter estimates an upper bound $\tilde{E}'_p$. The tuple is dropped whenever $E'_p < \lambda_q$ or $\tilde{E}'_p < \lambda_q$.

Step 3: Choose a relation to join with. For all relations that have not been joined with t, we probe all available indexes and count the number of matches of t from each index. We then multiply the number of matches with the cost of an index lookup, and finally choose the join index that yields the smallest value of the product. If we have exhausted join indexes, we simply choose the relation with the smallest size and use a full scan as the access method.

Step 4: Execute the (filtered) cross product. Once we have chosen to join the tuple t with an relation S, we execute the filtered cross product using the index on S if existent, or a cross product using a file scan on S. Once a new tuple t' is emitted, we mark all join predicates relevant to t' as selection predicates, and repeat the above four steps for the next query block.

Example 5.2 *Figure 2 shows the planning for tuple t_1 from relation R. t_1 has to pass three selection predicates, a join with relation S with three join predicates, and a join with relation T with two join predicates. These predicates and their estimated costs are shown in the shaded rows of the tables. In step 1, the selectivities of three selections are entered into the top table. In step 2, the selections are ranked with θ_2 first, then θ_1, and finally θ_3 (the filters are not shown in this example). In step 3, we choose the join with S using the second predicate because there is a join index and the expected cost of retrieving matches is the lowest. In step 4, tuple t_1 is paired with three matches from the join index. The three new tuples are sent back to the to-process pool for further processing. For these tuples,*

δ_1	δ_2	static order	static time (ms)	dynamic time (ms)	performance gain	optimal time (ms)
20	0.2	[1 2]	0.6	0.181	70%	0.177
20	0.5	[1 2]	0.6	0.068	89%	0.067
20	1	[2 1]	9.6	0.050	99%	0.048
22	0.2	[2 1]	18.2	7.216	60%	7.007
22	0.5	[2 1]	13.9	1.515	89%	1.482
22	1	[2 1]	9.6	0.351	96%	0.348
24	0.2	[2 1]	18.2	15.613	14%	15.287
24	0.5	[2 1]	14.4	6.390	56%	6.334
24	1	[2 1]	9.6	2.264	76%	2.236

Table 2: Static planning vs Dynamic planning for Q1.

δ_3	δ_4	static order	static time (s)	dynamic time (s)	performance gain
0.5	400	[3 4]	28.0	4.25	85%
0.5	800	[3 4]	80.1	12.1	85%
0.5	1600	[3 4]	142	22.9	84%
1	400	[4 3]	149	16.7	89%
1	800	[3 4]	105	49.3	53%
1	1600	[3 4]	187	97.2	48%
2	400	[4 3]	160	72.1	55%
2	800	[4 3]	486	217	55%
2	1600	[3 4]	487	432	11%

Table 3: Static planning vs Dynamic planning for Q2.

the join predicates associated with S become selection predicates, while those associated with T remain as join predicates.

6. EXPERIMENTAL EVALUATION

In this section, we evaluate our threshold query processing and optimization techniques. To demonstrate our performance benefits, we compare to the state-of-the-art techniques for indexing continuous uncertain data [7, 8] and for optimizing threshold queries [14]. Our evaluation uses real data and queries from the Sloan Digital Sky Survey (SDSS) [18]. See Appendix A for details about the dataset.

6.1 Techniques for Optimizing Selections

Expt 1: We first evaluate our general filtering technique described in Section 3.2.1. We consider a selection "$100 < rowc < 100 + \delta$ and $100 < colc < 100 + \delta$", which selects stars located in a square region anchored at the lower left vertex $(100, 100)$ and with side length δ. We can directly test the overlap between the selection region and each tuple's Chebyshev region due to their regular shapes.

We first set the threshold $\lambda = 0.7$ and varied δ from 200 to 2000, which approximately covers selectivities from 0% to 100%. We report the time cost per tuple for evaluating the selection with and without filters (baseline) in Fig. 3(a). The baseline has a constant high cost because it computes a 2-dimensional integral for each tuple, no matter what δ value is given. In contrast, using our filter the per tuple cost is very low for small δ values because most tuples can be filtered without computing integrals. As δ grows, more tuples pass the filter and invoke integrals for exact evaluation. The two curves meet when $\delta = 2000$ and 98% tuples satisfy the predicate.

We also varied λ in [0.7, 0.9]. λ mainly affects the number of tuples that pass the selection. A larger λ value results in fewer true matches, so our filter can prune more tuples and hence improve the performance shown in Fig. 3(a). More results are shown in [13].

Expt 2: We also evaluate the effectiveness of the fast filters from §3.2.2. Our results can be summarized as follows: For predicates on a single attribute, the Cantelli filter provides tight upper bounds. For multivariate quadratic predicates, both the Cantelli filter and general filter work well. In the interest of space, the details are left to §F.2.

6.2 Techniques for Optimizing Joins

Expt3: Band Join of Gaussians. We first study the filtering power and efficiency of our index for Gaussians, called GJ for short, and the state-of-the-art indexing method called xbound (detailed in Appendix F.1). We consider a join "$|R.u - S.u| < \delta$" with the threshold $\lambda = 0.7$ and varied δ.

We first evaluate the join in the stream setting: A tumbling window of size W is applied to both R and S inputs; each window contains a set of tuples. An index is built in-memory on the current window of S. Each tuple in the current R window probes the index after it is constructed. The retrieved (r, s) pairs are finally validated for true matches by computing an integral of the joint distribution. Hence, there are three cost components in this windowed join: *index construction*, *index lookup*, and *validation using integrals*. We observe consistently that the validation step is the dominating cost as integration is indeed very expensive. Below we report results using W =500 (other W sizes reveal similar trends).

Fig. 3(b) shows the number of candidates returned by our GJ index and the xbound index as well as the number of true matches. We can see that GJ returns exactly the true match set because it uses a sufficient and necessary condition for the join predicate. In contrast, the xbound index returns much more candidates. The difference becomes smaller as δ increases, because more tuples become true matches; when $\delta = 100$, almost all tuples in the indexed relation are true matches. Fig. 3(c) shows the efficiency of the two indexes. GJ significantly outperforms xbound because there is no need to validate the candidates returned from the index.

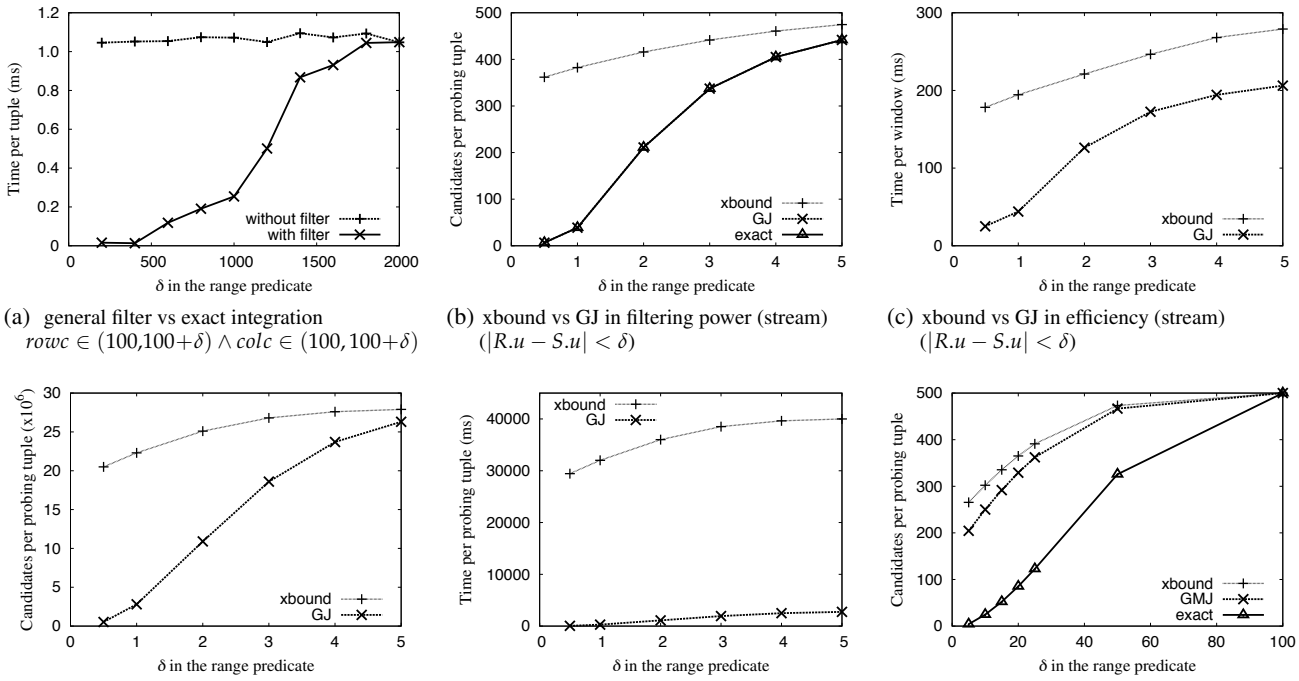
We then evaluate the join in a disk setting, where indexes are precomputed and stored on disk. Due to the limited size of the real data, we replicated it to 500MB with 28 million tuples. The R-tree took 1.3GB while the memory size was set to 1GB in our Java system. Since indexes are pre-constructed, their construction costs are not reported. Fig 3(d) shows the number of candidates for each probing tuple. While the trend appears similar to that in the stream setting, the absolute number for the y-axis is much larger. This determines the drastic difference between GJ and xbound in time cost shown in Fig 3(e). This difference comes from both the validation cost and the I/O cost as xbound returns many false positives.

Expt 4: Band Join of GMMs We then evaluate our join index for general distributions modeled as GMMs (called GMJ for short) and compare to xbound. As the SDSS uses Gaussians only, we generated a synthetic trace of GMMs for this experiment. The attribute u in each tuple has two Gaussian components; the coefficient, mean and variance of each component are uniformly drawn from (0,1), (0, 100), and (0,10) respectively. We report the results using the stream setting with $W = 500$. As Fig. 3(f) shows, both indexes return more candidates than the true matches because they are both based on necessary conditions for the join predicate. But GMJ is always better than xbound until they meet at $\delta = 100$, where the selectivity is near 100%, because xbound uses a "looser" condition as discussed in Appendix F.1. Since validation is the dominating cost, the time cost follows the same trend as the number of candidates in Fig. 3(f).

6.3 Per-tuple Based Planning and Execution

We finally evaluate our dynamic per-tuple planning technique. We compare it with static query planning [14], where a fixed plan is chosen for each query based on the selectivities of predicates over the entire data set—we give such full knowledge to the static query optimizer, hence showing its best performance. We design two query templates based on Q1 and Q2 from SDSS. We vary the parameters δ_1 and δ_2 to control selectivities of predicates for Q1 and δ_3 and δ_4 for Q2. The details of our setup are given in Appendix F.3.

Expt 5: We first consider Q1 and vary δ_1 and δ_2. Table 2 shows the time cost per tuple for static, dynamic and optimal planning. The optimal planning loads the optimal plan for each tuple (generated offline) into memory before it runs. The plan space for dynamic



(a) general filter vs exact integration
$rowc \in (100,100+\delta) \wedge colc \in (100,100+\delta)$

(b) xbound vs GJ in filtering power (stream)
$(|R.u - S.u| < \delta)$

(c) xbound vs GJ in efficiency (stream)
$(|R.u - S.u| < \delta)$

(d) xbound vs GJ in filtering power (disk)
$(|R.u - S.u| < \delta)$

(e) xbound vs GJ in efficiency (disk)
$(|R.u - S.u| < \delta)$

(f) xbound vs GMJ in filtering power (stream)
$(|R.u - S.u| < \delta)$

Figure 3: Experimental results for selections and joins.

planning is shown in Appendix F.3. Our dynamic query planning outperforms the static one in all cases, with over 50% gains in most cases and is very close to the optimal planning. The reasons are three-fold: (1) Each tuple is routed based on its distribution and resulting selectivities of predicates. A tuple may be sent to a predicate that is overall not selective but has a larger chance to filter this tuple. (2) The predicate cost is taken into consideration. It is possible for a tuple to be routed to a predicate with only a modest chance to filter the tuple but has a very low cost. (3) Our fast filters can drop tuples earlier at a lower cost than using the exact integration to evaluate predicates.

Expt 6: We next consider Q2 and vary δ_3 and δ_4. For this query, a join index is constructed for G_2 on the "rowc" and "colc" attributes for each window of size W. Table 3 shows the time cost of joining W tuples from the input G_1 with W tuples from G_2. We make similar observations as before: The dynamic planning is better than the static one in all cases. As we increase δ_3 and δ_4, more tuples satisfy both predicates. So the difference between the two schemes decreases and is mainly due to the benefit of using fast filters. Due to space constraints, the plan space for dynamic planning and the detailed analysis are given in Appendix F.3.

7. CONCLUSIONS

We presented techniques to optimize threshold query processing on continuous uncertain data by (*i*) expediting joins using new indexes, (*ii*) expediting selections by reducing dimensionality of integration and using faster filters, and (*iii*) using dynamic, per-tuple based plannig. Results using the SDSS benchmark show significant performance gains over a state-of-the-art indexing technique and its threshold query optimizer. In future work, we will extend threshold query optimization to a larger class of queries including group by aggregation, support user-defined functions, and evaluate our techniques in broader applications.

8. REFERENCES

[1] R. Avnur and J. M. Hellerstein. Eddies: Continuously adaptive query processing. In *SIGMOD*, 261–272, 2000.

[2] N. Beckmann, H.-P. Kriegel, et al. The R*-tree: An efficient and robust access method for points and rectangles. In *SIGMOD*, 322–331, 1990.

[3] O. Benjelloun, A. D. Sarma, et al. Uldbs: Databases with uncertainty and lineage. In *VLDB*, 953–964, 2006.

[4] P. Bizarro, S. Babu, et al. Content-based routing: different plans for different data. In *VLDB*, 757–768, 2005.

[5] S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004.

[6] S. Chaudhuri and K. Shim. Optimization of queries with user-defined predicates. In *ACM TODS*, 24(2):177–228, 1999.

[7] R. Cheng, S. Singh, et al. Efficient join processing over uncertain data. In *CIKM*, 738–747, 2006.

[8] R. Cheng, Y. Xia, et al. Efficient indexing methods for probabilistic threshold queries over uncertain data. In *VLDB*, 876–887, 2004.

[9] P. Cudré-Mauroux, H. Kimura, et al. A demonstration of SciDB: A science-oriented dbms. *PVLDB*, 2(2):1534–1537, 2009.

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

[11] M. Geoffrey and P. David. *Finite Mixture Models*. Wiley, 2000.

[12] J. Li and A. Deshpande. Ranking continuous probabilistic datasets. *PVLDB*, 3(1):638–649, 2010.

[13] L. Peng, Y. Diao, et al. Optimizing Probabilistic Query Processing. UMass Tech Report. http://claro.cs.umass.edu/pubs/tr11.pdf.

[14] Y. Qi, R. Jain, S. Singh, et al. Threshold query optimization for uncertain data. In *SIGMOD*, 315–326, 2010.

[15] N. Ravishanker and D.K. Dey. *A first course in linear model theory*. Chapman & Hall/CRC, 2002.

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

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

[18] A. S. Szalay, et al. Designing and mining multi-terabyte astronomy archives: The sloan digital sky survey. In *SIGMOD*, 451–462, 2000.

[19] T. T. L. Tran, et al. Conditioning and aggregating uncertain data streams: Going beyond expectations. *PVLDB*, 3(1):1302–1313, 2010.

[20] T. T. L. Tran, L. Peng, et al. Pods: a new model and processing algorithms for uncertain data streams. In *SIGMOD*, 159–170, 2010.

[21] D. Z. Wang, E. Michelakis, et al. Bayesstore: Managing large, uncertain data repositories with probabilistic graphical models. In *VLDB*, 340–351, 2008.

APPENDIX

A. DETAILS OF THE SDSS DATABASE

We illustrate the schema of the SDSS data set in Table 4.

name	type	description
OBJ_ID	bigint	SDSS identifier with [run, ..., field, obj]
...	...	
(*rowc*, *rowc_err*)	real	(row center position, error term)
(*colc*, *colc_err*)	real	(column center position, error term)
(*q_u*, *qErr_u*)	real	(stokes Q parameter, error term)
(*u_u*, *uErr_u*)	real	(stokes U parameter, error term)
(*ra*, *dec*, *ra_err*, *dec_err*, *ra_dec_corr*)	real	(right ascension, declination, error in ra, error in dec, ra/dec correlation)
...	...	

Table 4: Schema of the Galaxy table in the Sloan Digital Sky Survey (SDSS). Attributes in italics are uncertain.

We use two complex queries from the SDSS benchmark [18], where the attributes in the lower case are uncertain attributes:

```
Q1:  SELECT *
     FROM Galaxy G
     WHERE G.r < 22
     AND G.q_r^2+G.u_r^2 > 0.25;
Q2:  SELECT *
     FROM Galaxy AS G1, Galaxy AS G2
     WHERE G1.OBJ_ID < G2.OBJ_ID
     AND |(G1.u-G1.g)-(G2.u-G2.g)| < 0.05
     AND |(G1.g-G1.r)-(G2.g-G2.r)| < 0.05
     AND (G1.rowc-G2.rowc)^2+(G1.colc-G2.colc)^2<1E4;
```

The released data archive takes about 1GB. For experiments on selections and joins, we used the *Star* table, which has 57328 tuples and each column of uncertain attribute is about 1MB. We consider Q1 and Q2 for experiments on query planning, both of which involve the *Galaxy* table with 91249 tuples.

B. DETAILS OF THE DATA MODEL

A Gaussian Mixture Model (or distribution) is defined as follows.

Definition B.1 *A Gaussian Mixture Model (GMM) 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:*

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

where $0 \le p_i \le 1$, $\sum_{i=1}^{m} p_i = 1$, and each mixture component is a Gaussian distribution with mean μ_i and variance σ_i^2.

Definition B.2 *A multivariate Gaussian Mixture Model (multivariate GMM) for a random vector $\mathbf{X}$ naturally follows from the definition of multivariate Gaussian distributions:*

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

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

Correlation. To address inter-tuple correlation in our data model, we adopt the use of history to capture dependencies among attribute sets as a result of prior database operations [16]. The history, $\mathbf{H}$, of an attribute set is defined as follows: (1) For a newly inserted tuple t, $\mathbf{H}(t.\mathbf{A}) = t.\mathbf{A}$. (2) If a new set of attributes, $\bar{t}.\bar{\mathbf{A}}$, is derived from multiple attribute sets, $\{t_i.\mathbf{A_i} | i = 1, 2, \ldots\}$, via a database operation, then $\mathbf{H}(\bar{t}.\bar{\mathbf{A}}) = \cup\mathbf{H}(t_i.\mathbf{A_i})$. That is, the history of the new attribute set includes the base pdf's that can be used to derive the joint pdf of this set of attributes. Finally, if two attribute sets intersect, they become correlated. Then a joint distribution of the two sets can be computed from their histories to capture correlation.

C. DETAILS OF OPTIMIZING SELECTIONS

Correctness of finding the transformation matrix

PROOF. Given a selection region R_θ, it is known that for any nonsingular transformation matrix $\mathbf{D}_{k\times k}$, $Pr[R_\theta(\mathbf{X})] = Pr[R_\theta^D(\mathbf{Y}^D)]$, where R_θ^D is the transformed region by matrix $\mathbf{D}$. Below we show that given $\mathbf{B}_{l\times k}$ and $\mathbf{b}_{l\times 1}$ obtained from the procedure in §3.1, we can find a nonsingular matrix $\mathbf{D}_{k\times k}$ such that $Pr[R_\theta^D(\mathbf{Y}^D)] = Pr[R_\theta^B(\mathbf{Y}^B)]$, i.e., $\mathbf{B}$ gives the correct integration result.

Since $\mathbf{B}$ has full row rank, we can apply elementary column operations and transform it into $\left(\mathbf{B}'_{l\times l}, \mathbf{0}_{l\times(k-l)}\right)$, where $\mathbf{B}'$ is nonsingular and $\mathbf{0}$ is the zero matrix. Define a new matrix and a new vector

$$\mathbf{D}'_{k\times k} = \begin{pmatrix} \mathbf{B}'_{l\times l} & \mathbf{0}_{l\times(k-l)} \\ \mathbf{0}_{(k-l)\times l} & \mathbf{I}_{(k-l)\times(k-l)} \end{pmatrix}, \quad \mathbf{d} = \begin{pmatrix} \mathbf{b}_{l\times 1} \\ \mathbf{0}_{(k-l)\times 1} \end{pmatrix},$$

where $\mathbf{I}$ is the identity matrix. It can be proved that $\mathbf{D}'$ is nonsingular. We can then apply elementary column operations to $\mathbf{D}'$ such that the two upper blocks are transformed back to $\mathbf{B}$. Denote the new matrix as $\mathbf{D}$ and obviously $\mathbf{D}$ is also nonsingular. Then we have:

$$\mathbf{Y}^D = \mathbf{D}\mathbf{X} + \mathbf{d} = \begin{pmatrix} \mathbf{B}_{l\times k} \\ \mathbf{I}'_{(k-l)\times k} \end{pmatrix}\mathbf{X} + \begin{pmatrix} \mathbf{b}_{l\times 1} \\ \mathbf{0}_{(k-l)\times 1} \end{pmatrix} = \begin{pmatrix} \mathbf{B}\mathbf{X}+\mathbf{b} \\ \mathbf{I}'\mathbf{X} \end{pmatrix} = \begin{pmatrix} \mathbf{Y}^B \\ \mathbf{Y}' \end{pmatrix}.$$

Since there is no predicate on $\mathbf{Y}'$ in the selection condition θ, the integration interval on each dimension of $\mathbf{Y}'$ is $(-\infty, +\infty)$ and that on each dimension of $\mathbf{Y}^B$ does not vary with $\mathbf{Y}'$. So the integral of the joint *pdf* of $\mathbf{Y}^D$ over region R_θ^D equals that of its marginal distribution on $\mathbf{Y}^B$ over region R_θ^B, i.e., $Pr[R_\theta^D(\mathbf{Y}^D)] = Pr[R_\theta^B(\mathbf{Y}^B)]$. Then $Pr[R_\theta(\mathbf{X})] = Pr[R_\theta^B(\mathbf{Y}^B)]$. □

One-dimensional predicates:

Fast filters using Markov's inequality. We consider three cases that apply Markov's inequality based on the different relationships between the point 0 and the predicate region:

- $0 < a_1$: $Pr[R_\theta(X)] < Pr[(a_1, +\infty)] \le E|X|/a_1$
- $b_n < 0$: $Pr[R_\theta(X)] < Pr[(-\infty, b_n)] \le -E|X|/b_n$
- $b_{i-1} < 0 < a_i$: $Pr[R_\theta(X)] < Pr[(-\infty, b_{i-1})] + Pr[(a_i, +\infty)] \le E|X|/\min\{-b_{i-1}, a_i\}$

The distribution of $|X|$ can be computed as follows: When X follows a GMM with m components, each identified by parameters (p_i, μ_i, σ_i^2), we have $E|X| = \sum_{i=1}^{m} p_i E|X_i|$, where $E|X_i| = \sigma_i\sqrt{2/\pi}\exp(-\mu_i^2/(2\sigma_i^2)) + \mu_i(1 - 2\Phi(-\mu_i/\sigma_i))$ and Φ is the *cdf* of a standard normal distribution.

Fast filters using Chebyshev's inequality and Cantelli's inequality. Both inequalities can be applied to predicates $\bigcup_{i=1}^{n}(a_i, b_i)$ if μ does not reside in the predicate region. Again, we consider three cases below. In the first two cases, Cantelli's inequality gives a tighter upper bound. In the third last case, we need to compute upper bounds using both inequalities and choose the smaller one.

- $\mu < a_1$: $Pr[R_\theta(X)] < Pr[(a_1, +\infty)] \le \dfrac{\sigma^2}{\sigma^2 + (a_1 - \mu)^2}$
- $b_n < \mu$: $Pr[R_\theta(X)] < Pr[(-\infty, b_n)] \le \dfrac{\sigma^2}{\sigma^2 + (\mu - b_n)^2}$
- $b_{i-1} < \mu < a_i$: $Pr[R_\theta(X)] < Pr[(-\infty, b_{i-1})] + Pr[(a_i, +\infty)] \le \min\left\{\dfrac{\sigma^2}{(\min\{\mu - b_{i-1}, a_i - \mu\})^2}, \dfrac{\sigma^2}{\sigma^2 + (\mu - b_{i-1})^2} + \dfrac{\sigma^2}{\sigma^2 + (a_i - \mu)^2}\right\}$

Multi-dimensional quadratic predicates:

If $\mathbf{X}$ follows a GMM, its quadratic form $\mathbf{X}^T\Lambda\mathbf{X}$ yields a new random variable. We first derive the new distribution as follows: For $\mathbf{X} \sim N(\boldsymbol{\mu}, \Sigma)$, we have

$$E[\mathbf{X}^T\Lambda\mathbf{X}] = tr[\Lambda\Sigma] + \boldsymbol{\mu}^T\Lambda\boldsymbol{\mu}$$

$$Var[\mathbf{X}^T\Lambda\mathbf{X}] = 2tr[\Lambda\Sigma\Lambda\Sigma] + 4\boldsymbol{\mu}^T\Lambda\Sigma\Lambda\boldsymbol{\mu},$$

where $tr[\cdot]$ denotes the trace of a matrix.

For $\mathbf{X}$ follows a GMM with m components, each identified by $(p_i, \boldsymbol{\mu}_i, \Sigma_i)$ $(i = 1 \cdots m)$,

$$E[\mathbf{X}^T\Lambda\mathbf{X}] = \sum_{i=1}^{m} p_i E[\mathbf{X}_i^T\Lambda\mathbf{X}_i]$$

$$\begin{aligned} Var[\mathbf{X}^T\Lambda\mathbf{X}] &= E\left[\left(\mathbf{X}^T\Lambda\mathbf{X}\right)^2\right] - \left(E[\mathbf{X}^T\Lambda\mathbf{X}]\right)^2 \\ &= \sum_{i=1}^{m} p_i E\left[\left(\mathbf{X}_i^T\Lambda\mathbf{X}_i\right)^2\right] - \left(E[\mathbf{X}^T\Lambda\mathbf{X}]\right)^2 \\ &= \sum_{i=1}^{m} p_i\left(Var[\mathbf{X}_i^T\Lambda\mathbf{X}_i] + \left(E[\mathbf{X}_i^T\Lambda\mathbf{X}_i]\right)^2\right) - \left(E[\mathbf{X}^T\Lambda\mathbf{X}]\right)^2 \end{aligned}$$

Now suppose that the quadratic form $\mathbf{X}^T\Lambda\mathbf{X}$ yields a new random variable with mean μ_0 and variance σ_0^2. This allows us to apply Chebyshev's inequality and Cantelli's inequality as before. However, in the case of comparing a quadratic form with a constant, the predicate contains only one interval: either $(-\infty, \delta)$ or $(\delta, +\infty)$. Hence, when u_0 lies outside the predicate region, Cantelli's inequality always gives a tighter bound. Formally,

- If the predicate region is $\mathbf{X}^T\Lambda\mathbf{X} < \delta$, when $\delta < \mu_0$: $Pr[R_\theta(\mathbf{X})] \le \sigma_0^2 / \left(\sigma_0^2 + (\mu_0 - \delta)^2\right)$
- If the predicate region is $\mathbf{X}^T\Lambda\mathbf{X} > \delta$, when $\delta > \mu_0$: $Pr[R_\theta(\mathbf{X})] \le \sigma_0^2 / \left(\sigma_0^2 + (\delta - \mu_0)^2\right)$

D. DETAILS OF JOIN INDEXES

D.1 Proofs of Theorems

Proof for Theorem 1

PROOF. Suppose $\mu + \sigma\sqrt{(1-\lambda)/\lambda} < a$ or $\mu - \sigma\sqrt{(1-\lambda)/\lambda} > b$. Then interval $(\mu - \sigma\sqrt{(1-\lambda)/\lambda}, \mu + \sigma\sqrt{(1-\lambda)/\lambda})$ does not overlap with (a, b). According to Cantelli's inequality,

$$Pr[a < Z < b] < \frac{1}{1 + (1-\lambda)/\lambda} = \lambda,$$

which is a contradiction. This completes the proof. □

Proof for Theorem 2

PROOF. If there exists an $\alpha \in (0, 1-\lambda)$ such that $a - \Phi^{-1}(\alpha)\sigma \le \mu \le b - \Phi^{-1}(\lambda + \alpha)\sigma$, then we have

$$\frac{a-\mu}{\sigma} \le \Phi^{-1}(\alpha), \quad \frac{b-\mu}{\sigma} \ge \Phi^{-1}(\lambda + \alpha).$$

Therefore

$$\begin{aligned} Pr[a < Z < b] &= Pr[\frac{a-\mu}{\sigma} \le \frac{Z-\mu}{\sigma} \le \frac{b-\mu}{\sigma}] \\ &\ge Pr[\Phi^{-1}(\alpha) \le \frac{Z-\mu}{\sigma} \le \Phi^{-1}(\lambda + \alpha)] = \lambda. \end{aligned}$$

On the other hand, when $Pr[a < Z < b] \ge \lambda$, define $\alpha = \Phi(\frac{a-\mu}{\sigma})$. Apparantly $\alpha \in (0, 1-\lambda)$ and $\Phi(\frac{b-\mu}{\sigma}) \ge \lambda + \alpha$. Therefore, we found an α such that $a - \Phi^{-1}(\alpha)\sigma \le \mu \le b - \Phi^{-1}(\lambda + \alpha)\sigma$. □

D.2 Overlap Test in the Gaussian Join Index

Recall that when search the R-tree for the Gaussian index, our task is to test the overlap between $R_I' : [\mu_1, \mu_2; \sigma_1, \sigma_2]$ (an index entry) and $R_Q' = \Omega$ (the query region), where

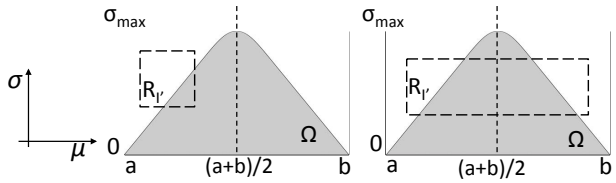


Figure 4: Two cases when R_I' and Ω overlap.

$$\Omega = \bigcup_{0 \le \alpha \le 1-\lambda} \left\{(\mu, \sigma) \,|\, a - \Phi^{-1}(\alpha)\sigma \le \mu \le b - \Phi^{-1}(\lambda + \alpha)\sigma\right\}.$$

First, we show that $\Omega \not\subset R_I'$: Since σ is the standard deviation of a random variable Z, $\sigma \ge 0$. In the extreme case when $\sigma = 0$, Z is reduced to a constant, so $\sigma_{min} = 0$, where σ_{min} is the minimum value of σ in Ω. For a valid search key, it is impossible that $\sigma_1 < 0$, so $\Omega \not\subset R_I'$.

Given that $\Omega \not\subset R_I'$, testing whether R_I' and Ω overlaps is the same as to test whether there exists a point (μ_0, σ_0) on the edges of R_I', such that $(\mu_0, \sigma_0) \in \Omega$.

Define a function $g(\mu, \sigma) = \Phi\left(\frac{b-\mu}{\sigma}\right) - \Phi\left(\frac{a-\mu}{\sigma}\right)$, where Φ is the *cdf* of the standard normal distribution. It is straightforward to see that $(\mu, \sigma) \in \Omega$ *iff* $g(\mu, \sigma) \ge \lambda$. Then the problem is again transformed to checking whether the maximum value of $g(\mu, \sigma)$ on edges of R_I', denoted as g_{max}, is no less than λ.

Without loss of generality, let us consider one edge of R_I', defined as $E_{\sigma_1} = \{(\mu, \sigma) \,|\, \mu_1 \le \mu \le \mu_2, \sigma = \sigma_1\}$. The goal is to find g_{max} for all points on E_{σ_1}. Let $g'(\mu) = g(\mu, \sigma_1) = \Phi\left(\frac{b-\mu}{\sigma_1}\right) - \Phi\left(\frac{a-\mu}{\sigma_1}\right)$. By taking the derivative w.r.t. μ and setting it to be 0, we get $\phi\left(\frac{b-\mu}{\sigma_1}\right) = \phi\left(\frac{a-\mu}{\sigma_1}\right)$, where ϕ is the *pdf* of the standard normal distribution. According to the symmetry of the standard normal distribution, $g'(\mu)$ has an extreme value at $\mu = (a+b)/2$. Then g_{max} can be gained as follows:

$$g_{max} = \begin{cases} \max\left\{g'(\mu_1), g'(\mu_2), g'\left(\frac{a+b}{2}\right)\right\} & \text{if } \mu_1 < \frac{a+b}{2} < \mu_2 \\ \max\left\{g'(\mu_1), g'(\mu_2)\right\} & \text{otherwise} \end{cases}$$

Similar analysis can be made for other edges of R_I'.

An optimization According to the above discussion, for each index entry R_I', we find the maximum value of $g(\mu, \sigma)$ on its edges and check whether it is no less than λ. A further optimization would be to find the circumscribed rectangle (minimum bounding box) of Ω, denoted as $MBR_\Omega = [\mu_{min}, \mu_{max}; \sigma_{min}, \sigma_{max}]$: if R_I' does not overlap with MBR_Ω, it is guaranteed that it does not overlap with Ω either. Below we show how to find μ_{min}, μ_{max}, σ_{min} and σ_{max}.

We have already showed that $\sigma_{min} = 0$. When $\lambda \ge 0.5$[1], $\forall \alpha \in [0, 1-\lambda]$, $\Phi^{-1}(\alpha) \le 0$ and $\Phi^{-1}(\lambda+\alpha) \ge 0$. Then $a \le a - \Phi^{-1}(\alpha)\sigma \le \mu \le b - \Phi^{-1}(\lambda+\alpha)\sigma \le b$. And we get $\mu_{min} = a$ and $\mu_{max} = b$, both obtained when $\sigma = 0$. Finally in order to find σ_{max}, we write out the expression of any point (μ, σ) on the boundary of Ω as:

$$\begin{cases} \mu = \dfrac{a\Phi^{-1}(\lambda+\alpha) - b\Phi^{-1}(\alpha)}{\Phi^{-1}(\lambda+\alpha) - \Phi^{-1}(\alpha)} \\ \sigma = \dfrac{b-a}{\Phi^{-1}(\lambda+\alpha) - \Phi^{-1}(\alpha)} \end{cases}$$

Let $\partial\sigma/\partial\alpha = 0$, we have $\phi(\Phi^{-1}(\lambda+\alpha)) = \phi(\Phi^{-1}(\alpha))$, where ϕ is the *pdf* of the standard normal distribution. Then $\alpha = (1+\lambda)/2$ and finally we get

$$\sigma_{max} = \frac{b-a}{\Phi^{-1}(\frac{1+\lambda}{2}) - \Phi^{-1}(\frac{1-\lambda}{2})}.$$

[1] We focus on cases when $\lambda \ge 0.5$, as it is more desirable in real applications.

Overlap routine design

Step 1 If R'_I does not overlap with MBR_Ω, return FALSE.

Step 2 Find g_{max} for points on the edges of R'_I, if $g_{max} < \lambda$, return FALSE; otherwise, return TRUE.

D.3 Additional Join Indexes

Extension of Band Joins to multivariate GMMs. When we have multiple join attributes $\mathbf{A}$, the band join involves conjunctive predicates that each is a 1-dim range predicate. It is easy to see that $Pr[\wedge_{i=1}^{|\mathbf{A}|}(a_i < R.A_i - S.A_i < b_i)] \leq \min_i\{Pr[a_i < R.A_i - S.A_i < b_i]\}$.

A necessary condition for $Pr[\bigwedge_{i=1}^{|\mathbf{A}|}(a_i < R.A_i - S.A_i < b_i)] \geq \lambda$ is $Pr[a_i < R.A_i - S.A_i < b_i] \geq \lambda$ for all i. This transforms the join of multivariate GMMs (or Gaussians) into multiple joins of univariate GMMs (or Gaussians). So we build an index for each join attribute in S. For a probing tuple r, all join indexes need to be retrieved, and a tuple s is a candidate match if it is returned by all the indexes.

Other Joins with Linear Predicates and General Distributions. We also support join predicates that are the "opposite" of band joins, e.g., "$|R.A - S.A| > \delta$". Such predicates can be useful for detecting a sudden dramatic change of the value of an attribute, e.g., the brightness of a star. We offer a necessary condition for such joins and build a join index accordingly. We can still apply linear transformation $Z = X_r - X_s$. Then we can prove the following statement by contradiction based on Chebyshev's inequality:

Theorem 3 *For a randome variable Z with mean μ and variance σ^2, if $Pr[|Z| > \delta] \geq \lambda$, then $\delta \leq |\mu| - \frac{\sigma}{\sqrt{\lambda}}$.*

Plugging $\mu = \mu_r - \mu_s$, $\sigma^2 = \sigma_r^2 + \sigma_s^2$ back, we can get the necessary condition for $Pr[|r.A - s.A| > \delta] \geq \lambda$ as follows:

$$\delta \leq |\mu_r - \mu_s| - \sqrt{\frac{\sigma_r^2 + \sigma_s^2}{\lambda}}. \quad (6)$$

Define the search key to be $x = \mu_s$ and $y = \mu_s^2 - \frac{\sigma_s^2}{\lambda}$, then Eq. (6) can be broken into two cases:

$$\begin{cases} x < \mu_r - \delta \\ y - 2(\mu_r - \delta)x \geq -(\mu_r - \delta)^2 + \frac{\sigma_r^2}{\lambda} \end{cases} \quad (7)$$

or

$$\begin{cases} x > \mu_r + \delta \\ y - 2(\mu_r + \delta)x \geq -(\mu_r + \delta)^2 + \frac{\sigma_r^2}{\lambda} \end{cases} \quad (8)$$

The index is still an R-tree and the query region is defined by inequalities in Eq. (7) and (8).

Distance Join of General Distributions. We next consider join predicates that involve the Euclidean distance between two sets of attributes $R.\mathbf{A}$ and $S.\mathbf{A}$:

$$\mathcal{D}(R.\mathbf{A}, S.\mathbf{A}) = \sqrt{\textstyle\sum_{i=1}^{|\mathbf{A}|}(R.A_i - S.A_i)^2} < \delta$$

Then the probabilistic threshold join is $R \bowtie_{\mathcal{D}(R.\mathbf{A},S.\mathbf{A})<\delta,\,\lambda} S$. Such joins are commonly used to check proximity of objects, e.g., two stars that are within 30 arcseconds of each other in query Q2.

Proximity joins can be supported using our techniques for band joins. When $|\mathbf{A}| = 1$, the join predicate can be rewritten to "$-\delta < R.A - S.A < \delta$", and thus can be directly supported as a band join. When $|\mathbf{A}| > 1$, we seek an upper bound of $Pr[\mathcal{D}(R.\mathbf{A}, S.\mathbf{A}) < \delta]$. We know that if the Euclidean distance between $R.\mathbf{A}$ and $S.\mathbf{A}$ is less than δ, then the Manhattan distance between $R.\mathbf{A}$ and $S.\mathbf{A}$ is less than δ in each dimension. So, $Pr[\mathcal{D}(R.\mathbf{A}, S.\mathbf{A}) < \delta] < Pr[\bigwedge_{i=1}^{|\mathbf{A}|}(-\delta < r.A_i - s.A_i < \delta)]$. This allows us to use indexes for band joins of multivariate GMMs or Gaussians for distance joins.

E. DETAILS OF QUERY PLANNING

Cost and Selectivity Estimation The unit operation in an exact selection is an integral. The integration cost depends on the dimensionality of integration and the shape of the selection region. For instance, we can benchmark 1-dimensional integrals on intervals, 2-dimensional integrals on rectangles and circles, and higher-dimensional integrals on hyper-rectangles and circles. Regarding the filters for selection, if they use known inequalities, then they have negligible costs. Filters that use optimization techniques such as the Lagrange multiplier may have a non-trivial cost, which can again be measured offline based on the types of predicates. Finally, the unit operation in a filtered cross product is to retrieve a match from the index for each probing tuple. Its cost can be estimated based on the height of the tree and the cost of the overlap test at each level of the tree.

We then define the selectivity, denoted by Γ, of a selection on each tuple t and the selectivity of a filtered cross product between a probing tuple t and a set S:

$$\Gamma^{\sigma}_{\theta,\lambda}(t) = Pr[R_\theta(\mathbf{X^t})], \quad \Gamma^{\times}_{\theta,\lambda}(t, S) = \frac{\text{num. true matches from } S}{|S|}$$

The selectivity of an operator can be estimated only when a tuple arrives with its attribute distribution. For a selection, we estimate its selectivity for a tuple, $\Gamma^{\sigma}_{\theta,\lambda}(t)$, by taking the average of its upper and lower bounds. Recall that we showed many upper bounds derived from statistical inequalities in Section 3. Actually we can also obtain lower bounds using appropriate inequalities. For instance, given a one-dimensional range predicate (a, b), Markov's inequality can be applied when $0 \in (a, b)$ and yields a lower bound on the selectivity. Chebyshev's and Cantelli's inequalities can be applied similarly. The upper and lower bounds used for the three categories of predicates in §3.2.2 are shown in our technical report [13]. Finally, to estimate the selectivity of a filtered cross product for a probing tuple, our current solution is to perform the index lookup to count the matches but without retrieving the complete tuples. We will consider more advanced techniques in future work.

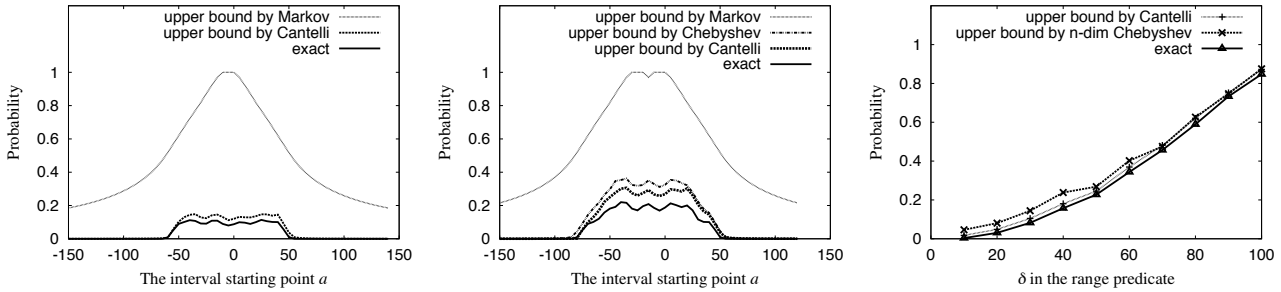
F. DETAILS OF EVALUATION

F.1 XBound-based Join Index

We implemented a state of the art join index on continuous uncertain attributes [8, 7], called xbound join index. As stated earlier, this technique makes simplifying assumptions about attribute distributions; that is, every uncertain attribute has a known interval $[l, u]$ in which a pdf is defined. This assumption does not hold for general continuous uncertain attributes which range from $[-\infty, \infty]$. Relaxing the interval of an uncertain attribute from $[l, u]$ to $[-\infty, \infty]$ will yield zero filtering power of the Xbound join index. In addition, this join index is based on a "loose" necessary condition for the join predicate to be true, hence resulting in poor performance as we demonstrate in Section 6. Details of the join index are as follows.

Definition F.1 (x-bound) *For a random variable Y with density function f and domain $[l, u]$, given $0 \leq x \leq 1$, the x-bound of a distribution consists of two values, called left-x-bound (l_x) and right-x-bound (u_x), where $\int_l^{l_x} f(y)\mathrm{d}y = x$ and $\int_{u_x}^{u} f(y)\mathrm{d}y = x$.*

Consider $R \bowtie_{\theta,\lambda} S$, a **necessary condition** for $Pr[|X_r - X_s| \leq \delta] \geq \lambda$ extracted from [7] is $u_{s,\lambda} \geq l_r - \delta$ and $l_{s,\lambda} \leq u_r + \delta$, where (l_r, u_r) is the domain of X_r, $l_{s,\lambda}$ and $u_{s,\lambda}$ are the left-λ-bound and right-λ-bound for X_s. Assume an join index is built on S. Given λ, for each tuple s, insert the **search key** $(l_{s,\lambda}, u_{s,\lambda})$ into an R-tree. When a probing tuple r arrives, the **query region** is $\{(l_{s,\lambda}, u_{s,\lambda}) \,|\, l_{s,\lambda} \leq u_r + \delta, u_{s,\lambda} \geq l_r - \delta\}$.



(a) fast filters vs exact integration $(a<x<a+10)$

(b) fast filters vs exact integration $(a<x<a+10)$ or $(a+20<x<a+30)$

(c) fast filter, general filter, and exact integration $(x^2+y^2<\delta^2)$

Figure 5: Experimental results for the filtering techniques for selections.

	δ_1			δ_2			δ_3			δ_4		
Value	20	22	24	0.2	0.5	1	0.5	1	2	400	800	1600
Selectivity(%)	8.3	61.9	95.9	44.3	14.5	6.9	40.1	81.7	99.4	45.4	83.7	100

Table 5: Selectivity for different $\delta_i (i = 1 \cdots 4)$.

For a probing tuple r with domain $(-\infty, +\infty)$, the above necessary condition has no power in guiding the search of $(l_{s,\lambda}, u_{s,\lambda})$. We modify the domain of a normally distributed random variable to be $(\mu - 5\sigma, \mu + 5\sigma)$, and modify the domain of a GMM to be $(F^{-1}(0.00001), F^{-1}(0.99999))$, where F^{-1} is the *inverse cdf* of the random variable.

F.2 More Results on Selections

Expt 2: We evaluate the effectiveness of the fast filters from §3.2.2. Since they have negligible costs, we focus on how tight their upper bounds are, i.e., their filtering power. We here use synthetic data with various controlled properties in microbenchmarks.

Consider predicates on a single attribute (Category 1 in §3.2.2): (i) $a<x<b$, (ii) $a_1<x<b_1$ or $a_2<x<b_2$. To have workloads with controlled properties, we generated synthetic data where the mean and variance of the normal distribution for each tuple are randomly chosen from (-50,50) and (0,10) respectively. For the predicate (i), the Cantelli filter always gives a tighter upper bound than the Chebyshev filter, so we only compare Markov with Cantelli and use the exact probability as the baseline. We set the interval length to be 10 and vary the starting point of the interval from -150 to 150. As shown in Fig. 5(a), Cantelli's upper bound is much tighter than Markov and close to the exact probability, which agrees with known statistical results. For predicate (ii), we compare all three filters as they may have tradeoffs. From Fig. 5(b), we observe similar trends as before. In addition, the upper bounds by the Chebyshev filter lie in between those by Markov and Cantelli on the average (though for specific tuples, the order of the filters may be otherwise).

We next consider predicates in a quadratic form: "$x^2+y^2<\delta^2$" (Category 2). We used a synthetic trace with attributes x and y whose mean and variance are uniformly distributed in (0,100) and (0,10) respectively. We compared the Cantelli filter which transforms the quadratic form to a single random variable, with the general multi-dimensional filter that uses the Lagrange multiplier for constraint optimization (for this predicate, there is a fast solver). We varied δ from 10 to 100. Fig. 5(c) shows that both the Cantelli filter and general filter capture the trend of the exact probability and offer tight bounds, with Cantelli being slightly better. They are both over 400x faster than integration, hence good choices for quadratic predicates.

F.3 More Results on Query Planning

We design two query templates based on Q1 and Q2 on the *Galaxy* view in SDSS and show them in Fig. 6. Basically all tuples will be routed to the quick filters whenever the filters can be applied and there is no specific order of applying filters, as their costs are all very low. After that, the dynamic optimizer decides the order of evaluating exact selection predicates, shown by a box in Fig. 6. We vary the parameters $\delta_1, \cdots, \delta_4$ to tune the selectivity of each predicate, which affects planning.

Query Q1 uses two predicates: $\theta_1 : r < \delta_1$; $\theta_2 : q\_r^2 + u\_r^2 > \delta_2^2$. Query Q2 uses two join predicates: $\theta_3 : |(G1.u - G1.g) - (G2.u - G2.g)| < \delta_3$ and $|(G1.g - G1.r) - (G2.g - G2.r)| < \delta_3$; $\theta_4 : (G1.rowc - G2.rowc)^2 + (G1.colc - G2.colc)^2 < \delta_4^2$.

Given various values of δ_i, the selectivity of each predicate averaged over the entire data set is shown in Table 5. The static query plan is decided by ordering predicates with lowest selectivity first.

Expt 6: We consider Q2 and vary δ_3 and δ_4. For this query, a join index is constructed for G_2 on the "rowc" and "colc" attributes for each window of size W. Table 3 shows the time cost for the join per window. To evaluate the join predicates θ_3 and θ_4, all tuples need to be routed to probe the (only) join index first. Moreover, the evaluation costs of δ_3 and δ_4 are close due to the use of 2-dimensional integrals after linear transformation. The dynamic planning is better than the static one in all cases due to the reasons mentioned previously. There are several other interesting observations: (1) When $\delta_3 = 0.5$, the static optimizer always evaluates θ_3 first; the dynamic optimizer tends to choose θ_3 for most tuples as well. This is because for each tuple, most candidates returned by our index are true matches, then the selectivity of each pair of probing tuple and its candidate w.r.t. θ_4 will be estimated to be very close to 1. Since both optimizers route tuples to θ_3 as we increase δ_4, our improvement is mainly gained by the benefit of using fast filters. (2) When $\delta_3 = 1$ and $\delta_4 = 400$, the static optimizer evaluates θ_4 first, which is not a wise choice, because as mentioned above, most candidates returned by the index are true matches, then the evaluation of θ_4 can only filter very few tuples, and most tuples will be passed on to θ_3 next. In contrast, the dynamic optimizer tends to route tuples to θ_3 as discussed above and fewer tuples will be passed on to θ_4.

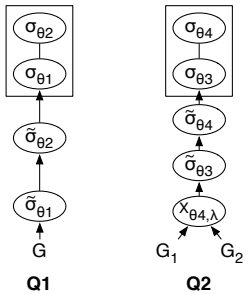


Figure 6: Plan space in dynamic planning for Q1 and Q2.