5-2013

High-Performance Processing of Continuous Uncertain Data

Thanh Thi Lac Tran
University of Massachusetts Amherst, ttran@cs.umass.edu

HIGH-PERFORMANCE PROCESSING OF CONTINUOUS UNCERTAIN DATA

A Dissertation Presented

by

THANH T. L. TRAN

Submitted to the Graduate School of the
University of Massachusetts Amherst in partial fulfillment
of the requirements for the degree of

DOCTOR OF PHILOSOPHY

May 2013

Computer Science

HIGH-PERFORMANCE PROCESSING OF CONTINUOUS UNCERTAIN DATA

A Dissertation Presented

by

THANH T. L. TRAN

Approved as to style and content by:

Yanlei Diao, Chair

Jim Kurose, Member

Anna Liu, Member

Andrew McGregor, Member

Charles Sutton, Member

Lori A. Clarke, Department Chair
Computer Science

ACKNOWLEDGMENTS

This thesis would not have been possible without the guidance and support of my advisor Prof. Yanlei Diao. She introduced me to the world of database research and taught me much about research skills over the years. I am grateful for her close mentorship, her countless hours spent to help define and shape this work. I would also like to thank her for teaching me to always strive for clarity and precision in writing papers and giving presentations, which is undoubtedly valuable to me in many years to come.

I am grateful for having Profs. Jim Kurose, Anna Liu, Andrew McGregor, and Dr. Charles Sutton on my thesis committee. I would like to thank Prof. Jim Kurose for his input on the CASA case study and valuable comments on improving the presentation of this thesis. I am grateful to Prof. Andrew McGregor for his help and many insights on the approximation algorithms, and for his sense of humor that brought more fun to the work. I am thankful to Dr. Charles Sutton for collaborating in the work on RFID data inference and user-defined functions, and answering my questions about machine learning in general. I thank Prof. Anna Liu for her invaluable input on the statistical techniques for various parts of this work.

During my study, I had an opportunity to do a summer internship at AT&T Labs. I would like to thank Drs. Graham Cormode, Magda Procopiuc, and Divesh Srivastava for their great mentorship, the experience of working on a different interesting research problem, and also an enjoyable summer.

I have benefited from the teaching of many professors at UMass, Amherst. I especially thank Prof. Andrew Barto for his kindness and guidance during my early

time at UMass. I thank Prof. Prashant Shenoy for his helpful comments on my RFID work. I am grateful to Profs. Gerome Miklau and Alexandra Meliou for input and discussions about many aspects of database research.

I would like to thank all members of the Database Lab, past and present. Many thanks to Liping Peng for her close collaboration on the CLARO project and also her friendship. I particularly thank Boduo Li for doing the hard work of performing the experiment in the CASA case study. I thank Wentian Lu, Chao Li, Michael Hay, Rick Cocci, Haopeng Zhang, Ed Mazur, Vani Gupta, Abhishek Roy, Kevin Conor, Yue Wang, Ravali Pochampally for discussing ideas, offering technical comments and help, and making the lab a nice place to work.

I thank Rachel Lavery and Leeanne Leclerc for their help with paperwork and administrative issues during my time at UMass.

I am thankful to the Vietnam Education Fellowship for the financial support for the first two years of my program. My work was also supported by the National Science Foundation under the grants CNS-0626873, IIS-0746939, and IIS-0812347.

I am grateful to my Viet friends in Amherst. Thanks to all for the support, the fun occasions and many laughters, which have kept me sane and healthy. I enjoy the good food and the outdoor activities we have shared together.

Finally, I am greatly indebted to my parents and my sisters. I thank my sisters for always caring and being a joy to talk with. I thank my parents for supporting me through many ups and downs, being patient with me and my many years away from home. I cannot thank them enough for their love and sacrifice in bringing me and my sisters up. I dedicate this thesis to them.

ABSTRACT

HIGH-PERFORMANCE PROCESSING OF CONTINUOUS UNCERTAIN DATA

MAY 2013

THANH T. L. TRAN

B.E., UNIVERSITY OF MELBOURNE

M.S., UNIVERSITY OF MASSACHUSETTS AMHERST

Ph.D., UNIVERSITY OF MASSACHUSETTS AMHERST

Directed by: Professor Yanlei Diao

Uncertain data has arisen in a growing number of applications such as sensor networks, RFID systems, weather radar networks, and digital sky surveys. The fact that the raw data in these applications is often incomplete, imprecise and even misleading has two implications: (*i*) the raw data is not suitable for direct querying, (*ii*) feeding the uncertain data into existing systems produces results of unknown quality.

This thesis presents a system for uncertain data processing that has two key functionalities, (*i*) capturing and transforming raw noisy data to rich queriable tuples that carry attributes needed for query processing with quantified uncertainty, and (*ii*) performing query processing on such tuples, which captures changes of uncertainty as data goes through various query operators. The proposed system considers data naturally captured by continuous distributions, which is prevalent in sensing and scientific applications.

The first part of the thesis addresses data capture and transformation by proposing a probabilistic modeling and inference approach. Since this task is application-specific and requires domain knowledge, this approach is demonstrated for RFID data from mobile readers. More specifically, the proposed solution involves an inference and cleaning substrate to transform raw RFID data streams to object location tuple streams where locations are inferred from raw noisy data and their uncertain values are captured by probability distributions.

The second, also the main part, of this thesis examines query processing for uncertain data modeled by continuous random variables. The proposed system includes new data models and algorithms for relational processing, with a focus on aggregation and conditioning operations. For operations of high complexity, optimizations including approximations with guaranteed error bounds are considered. Then complex queries involving a mix of operations are addressed by query planning, which given a query, finds an efficient plan that meets user-defined accuracy requirements.

Besides relational processing, this thesis also provides the support for user-defined functions (UDFs) on uncertain data, which aims to compute the output distribution given uncertain input and a black-box UDF. The proposed solution employs a learning-based approach using Gaussian processes to compute approximate output with error bounds, and a suite of optimizations for high performance in online settings such as data stream processing and interactive data analysis.

The techniques proposed in this thesis are thoroughly evaluated using both synthetic data with controlled properties and various real-world datasets from the domains of severe weather monitoring, object tracking using RFID readers, and computational astrophysics. The experimental results show that these techniques can yield high accuracy, meet stream speeds, and outperform existing techniques such as Monte Carlo sampling for many important workloads.

TABLE OF CONTENTS

LIST OF TABLES

Table **Page**

LIST OF FIGURES

Figure **Page**

CHAPTER 1

INTRODUCTION

Recent advances of sensing technology have enabled many scientific and monitoring applications such as sensor networks [27, 28, 57], radio frequency identification (RFID) networks [21, 50, 78, 100], GPS systems [52], severe weather monitoring [30, 58], and computational astrophysics [92, 91]. While these applications have been shown to be important in many domains, they raise new challenges for data management. A big challenge is that data resulting from real-world measurements is inherently noisy, incomplete, and even misleading, hence referred to as *uncertain data*. Capturing uncertainty from raw input data to query processing results then becomes a key component of data management systems (DBMSs). However, existing DBMSs either are not ready to process raw uncertain data or cannot quantify the uncertainty of query results, hence are of limited use for these applications. This chapter presents the motivating applications and then sets the objectives for this thesis.

1.1 Motivations

We now consider three specific applications that motivate this thesis work, including object tracking and monitoring using RFID technology, computational astrophysics using digital sky surveys, and severe weather monitoring using radar networks.

A. RFID tracking and monitoring. The first motivating application is object tracking and monitoring using RFID technology, in particular, wide-range mobile readers that enable cost-effective deployments in areas such as retail management healthcare, pharmaceuticals [36], and library management [32, 80]. For example, a

mobile RFID reader, as attached to a robot or a handheld device, can be deployed to repeatedly scan a storage area. The collected RFID readings contain the tag ids of observed objects, and optionally the reader locations of the mobile reader. These RFID readings have two important characteristics. First, the observed data is incomplete and noisy, since the read rate of RFID readers is far less than 100% due to environmental factors such as occluding metal objects, interference, and contention among tags [33]. Second, while the monitoring application wants precise *object locations* for further processing, the observed data simply contains observed *tag ids*—this is a fundamental limitation of the identification technology. As a result, the *raw data* is not directly queriable for those queries that require object locations in processing.

Despite these data quality issues, the monitoring application needs accurate object locations to derive high-level information. We illustrate such needs using a fire monitoring application. Assume that raw RFID readings can be transformed into a stream of tuples each containing `(time, tag_id of` O_i`,` $(x, y, z)^p$`)`, where $(x, y, z)^p$ denotes the uncertain (x, y, z) location of the object O_i. The following query Q1 detects potential violations of a fire code, which states that display of solid merchandise shall not exceed 200 pounds per square foot of shelf area. This is a group-by aggregation query, that considers tuples in each 5 second window, groups them based on the square foot area, computes the total weight of the objects in each group. It then reports the area and the total weight for each group whose weight exceeds 200 pounds. The query is written as follows as if the object location were precise.

```
Q1:  Select   Rstream(R2.area, sum(R2.weight))
     From     ( Select Rstream(*, area(R.(x,y,z)) As area,
                                  weight(R.tag_id) As weight)
                From   RFIDStream R [Now])
                R2 [Range 5 seconds]
     Group By R2.area
     Having   sum(R2.weight) > 200 pounds
```

Due to the nature of this application, it is important to capture the quality of the detection results. For example, in the above query, the user may want to know for each result tuple, how likely `sum(weight) > 200` evaluates to true, or what the distribution of `sum(weight)` looks like.

B. Computational astrophysics. There have been several recent initiatives to apply relational techniques to computational astrophysics such as the Sloan digital sky survey (SDSS) [89], and the SciDB project [85]. As detailed in the recent work [91], massive astrophysical surveys will soon generate observations of 10^8 stars and galaxies at nightly data rates of 0.5TB to 20TB. The observations are inherently noisy as the objects can be too dim to be recognized in a single image. Repeated observations (up to a thousand times) allow scientists to model the location, brightness, and color of objects using appropriate distributions, for example, represented as (id, $time$, $(x, y)^p$, $luminosity^p$, $color^p$). This data cooking process has transformed the raw data into attributes needed for query processing. However, query processing of the resulting uncertain data remains underaddressed in the literature. More specifically, queries can be issued to detect dynamic features, transient events, and anomalous behaviors. Query Q2 below detects regions of the sky that have high luminosity from the observations in the past hour. It groups the objects into the predefined regions and for the regions with the maximum luminosity above a threshold, it reports the maximum luminosity.

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

The fact that *luminosity* is an uncertain attribute characterized by continuous distributions complicates the computation of the query. Moreover, this computation needs to be performed in real time as tuples arrive, posing additional challenges in processing the uncertain data.

Besides relational processing as illustrated in the above query, this application makes intensive use of *user-defined functions (UDFs)*, which process and analyze the data using complex, domain-specific algorithms. In practice, UDFs can be provided in any form of external code, e.g., C programs, and hence treated mainly as *black boxes* in traditional databases. The following query Q3 shows a simple example of the use of UDFs in astrophysics. Q3 computes the age of each galaxy given its redshift using the UDF $GalAge$. Since $redshift$ is uncertain, the output $GalAge(redshift)$ is also uncertain, characterized by a distribution.

```
Q3:  Select G.objD, GalAge(G.redshift)
     From Galaxy G
```

These UDFs are often expensive to compute due to their complexity of processing. Unfortunately, the support for UDFs on uncertain data is largely lacking in today's data management systems.

C. Severe weather monitoring. Our third application is severe weather monitoring. The Engineering Research Center for Collaborative Adaptive Sensing of the Atmosphere (CASA) [15] is leading an effort to create distributed radar sensor networks with the goal to detect and monitor hazardous weather events like storms and tornados [58]. A fundamental problem that emerges in this system is the possibility of detection errors caused by the uncertainty in the data generated by the radars and transformed in various processing stages. Data uncertainty can arise from environmental noise, device noise, and inaccuracies of various radar components. The raw radar data is generated at a high volume of 205 Mb per second. The current system deals with this high-volume noisy data by means of taking average over the data, which may result in loss of precision.

Given the raw data, the first task in the CASA data processing workflow is to derive attributes needed for query processing and model those attributes using probability distributions to capture the data uncertainty. For example, this may result in a

data stream with the format ($time$, $azimuth$, $distance$, $velocity^p$, $reflectivity^p$). The current system performs initial data cooking to compute the needed attributes, but does not quantify their uncertainty. After sufficient data cooking, the transformed data then needs to be processed through subsequent operators, mainly aggregations, and eventually fed into the tornado detection algorithm. Data uncertainty can propagate through the entire system, making tornado detection results error-prone. Given the potential social impact of such a system, it is absolutely vital that the system capture the data quality at various processing stages and the uncertainty of its detection results.

1.2 Thesis Statement

The three applications above present a number of challenges to existing database management systems. In these applications, the raw data resulted from sensing processes is significantly different from the traditional data. More specifically, the differences include the following:

(i) Observed data is inherently incomplete and noisy due to the limitations of the sensing technology and many environmental factors, and the noise varies with time and location.

(ii) Observed data is different from data needed for further processing. For example, in RFID tracking and monitoring, the observed data contains object tag ids while the data of interest to monitoring applications concerns object locations. In computational astrophysics, the observed data is the image of the sky generated by telescopes while the data of interest is the properties of the stars and galaxies such as luminosity and color. And in weather monitoring, the observed data is raw signal data whereas data needed for further processing is a numeric description of each unit area of space in terms of reflectivity, wind speed, etc. Hence, given the raw data, the

supporting database system needs to handle both the mismatch between observed data and data of interest, and the noise in observed data.

(*iii*) After raw data is transformed into a suitable format, it needs to undergo sophisticated query processing to derive final query answers. The challenge is to capture uncertainty as data propagates through query operators until the final results.

(*iv*) The nature of the sensing applications adds performance requirements. Since data often arrives as data streams and requires online analysis, data processing needs to keep up with stream speed. Besides, as can be seen in the above applications, the raw data is particularly of higher volume than in traditional sensing applications, e.g., 205Mb/sec from a single radar node, which requires the processing of raw data to keep up with such high data rates.

This thesis presents *the design of a system, named* CLARO*, that provides an end-to-end solution from raw data collection to query processing to final result generation.* To support uncertainty as a first-class citizen, CLARO models uncertain data using *continuous random variables*, which are natural to most types of sensing and scientific data. More specifically, the CLARO system offers two main functionalities: (*i*) capturing and transforming raw, noisy data to rich, queriable tuples with quantified uncertainty, and (*ii*) performing complex query processing on the resulting tuples and capturing the uncertainty as it propagates through processing operations. CLARO aims to compute the distributions of final processing results, either exact or approximate with bounded errors. It also aims to meet user-specified accuracy requirements, and at the same time be efficient for data streams or interactive analysis, and scale to high-volume data.

1.3 Thesis Contributions

This section summarizes the contributions achieved in the design and development of the CLARO system for uncertain data processing. These contributions address the two main functionalities of CLARO.

1. Transforming raw noisy data into queriable tuples with quantified uncertainty. Specifically, this contribution involves deriving tuple attributes needed for query processing and characterizing uncertainty in these attributes using continuous probability distributions. This thesis proposes a general approach using probabilistic modeling and inference to recover the data of interest from the raw noisy data. Since this task is application-specific and requires domain knowledge, this proposed approach is demonstrated for the application of object tracking and monitoring using mobile RFID readers. The contribution is a complete solution for efficient, scalable cleaning and transformation of mobile RFID data streams while offering high precision results. More specifically, this involves (1) *modeling precisely how mobile RFID data is generated from the true state of the physical world, e.g., true object locations, through the sensing process,* and (2) *inferring likely estimates of the true state as noisy, raw data streams arrive.* These two tasks are described in more detail as follows.

- ***Modeling the data generation process.*** CLARO presents a probabilistic model that captures the underlying data generation process, including the key components such as reader motion, object dynamics, and noisy sensing of these objects by the reader. In particular, the proposed model employs a flexible parametric RFID sensor model that can be automatically and accurately configured for a variety of environments using a standard learning technique. In contrast, existing work resorts to manual calibration of the sensor model for each RFID deployment environment [32, 44, 50], precisely because they lack such a flexible parametric sensor model.

- ***Efficient, scalable inference.*** To generate clean location event streams from noisy, raw RFID data streams, CLARO applies a sampling-based inference technique, called particle filtering, to the model developed above. The basic application of this technique requires a prohibitively large number of samples to cope with the number of objects typical in our target environment, and hence is inadequate for stream processing. The second contribution made in this work is to enhance particle filtering to *scale to large numbers of objects* and *keep up with high-volume streams* while offering high precision inference results. To do so, the CLARO system presents three advanced techniques, namely, particle factorization, spatial indexing, and belief compression, which together lead to a solution that uses only a small number of samples at any instant by focusing on a subset of the objects, while maintaining high inference accuracy.

Besides RFID data, CLARO employs some alternative techniques for data capture and transformation for extremely high-volume raw data such as radar network data. These techniques involve statistical models for time series data and their approximations for improved efficiency.

2. Relational query processing of continuous uncertain tuples. After data capture and transformation, tuples carrying continuous probability distributions propagate through various query processing operations. While the type of sensor data may vary in our applications, query processing can be supported by a unified framework, because data processing in these applications involves a common set of relational operators such as selection, aggregation, join, group-by. The CLARO system aims to capture uncertainty of both intermediate and final results. More specifically, CLARO characterizes the full probability distributions of the output of each processing operator, either exact or approximate with bounded errors. From these distributions, the confidence regions and error bounds can be generated when needed.

CLARO is designed to be a probabilistic data management system that supports query processing of continuous-valued uncertain data, in either stored databases or data streams. It provides data models, formal semantics and processing techniques of relational operators, and query planning for complex queries. The main contributions are as follows.

- ***Data model***. The foundation of CLARO is a unique data model, named *mixed-type model*. In this model, continuous uncertain attributes follow *Gaussian mixture distributions*, which can model complex real-world distributions [39]. They also allow us to develop efficient solutions for many relational operators. Besides the attribute-level uncertainty captured by such distributions, the mixed-type model can also capture tuple-level uncertainty regarding the existence of a tuple.
- ***Formal semantics***. In like manner that the possible worlds semantics (PWS) [23] laid the foundation for query processing on discrete uncertain data, CLARO defines formal semantics for relational processing under its chosen model for continuous uncertain data. This formal semantics, based on measure theory, is shown to be equivalent to PWS when used in the discrete case.
- ***Aggregates of Gaussian mixture distributions***. The chosen data model enables the design of efficient techniques for aggregates such as `sum` and `avg`. Specifically, when the tuple existence is certain, there are *exact* result distributions of aggregates, which eliminates the use of integrals. In workloads when the exact solution is slow, CLARO derives *approximate* distributions with bounded errors for improved efficiency. These techniques, when used as a hybrid solution, can meet arbitrary accuracy requirements while achieving high throughput.
- ***Aggregates under the mixed-type model.*** Given uncertain attributes, conditioning operations, e.g., selection and group-by, can introduce uncertainty regarding tuple existence, which complicates the computation for aggregates. the CLARO system proposes an *approximate* evaluation framework for the mixed-type model

that includes tuple existence probabilities. Within this framework, CLARO supports deterministic and randomized approximation algorithms with error bounds for common aggregates like `max`, `min`, `sum`, and `count`.

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

3. Supporting user-defined functions on uncertain data. Besides queries expressed with relational operations, we observe that user-defined functions (UDFs) are prevalent in many scientific applications. These UDFs can be provided in any form of external code, hence treated mainly as black boxes. Given a UDF and uncertain input, the CLARO system aims to compute the distribution of each output tuple.

To this end, CLARO explores a learning-based approach by modeling UDFs using a technique called *Gaussian processes* (GPs). The key idea is that over time, one can use past function evaluations to build an approximate model of the black-box function, and use the model to avoid most expensive function evaluations in the future. Within this framework, CLARO innovates by using novel techniques to compute output distributions of a UDF modeled as a GP, when given uncertain input, and providing new theoretical results to bound errors of output distributions.

Further, CLARO proposes an efficient online algorithm to compute approximate output distributions that satisfy application accuracy requirements. This algorithm employs a suite of novel optimizations for the GP learning and inference modules, namely *local inference*, *online tuning*, and *online training* to improve performance and accuracy.

Finally, the CLARO system adopts Monte Carlo sampling as an alternative to compute UDFs on uncertain data, especially for fast functions, and suggest a hybrid solution of using direct Monte Carlo sampling and Gaussian process modeling. Specifically, this solution aims to choose the more efficient approach depending on the characteristics of UDFs such as their evaluation time and complexity.

4. System prototyping and performance evaluation. Lastly, this thesis work involves implementing a prototype system, performing evaluation of the proposed techniques, and comparing them with the state-of-the-art solutions. The evaluation uses both synthetic data with controlled properties and a variety of real-world workloads and datasets from a RFID object tracking lab deployment, CASA real radar traces, and a dataset from the SDSS project. The experimental results show that the proposed techniques outperform the state of the art such as Monte Carlo sampling for most important workloads. For the CASA case study, the proposed techniques can enable the tornado detection system to produce detection results at stream speed with improved quality. For the real data and queries from the applications of object tracking and computational astrophysics, CLARO can meet high accuracy requirements while achieving throughput of thousands of tuples per second or higher for most workloads tested.

1.4 Thesis Layout

This chapter has presented the overview of the thesis, including the motivating applications, the objectives, and the technical contributions. The rest of this thesis is outlined as follows. Chapter 2 describes the architecture of the CLARO system. Chapter 3 examines the techniques for data capture and transformation to derive rich, queriable tuples with quantified uncertainty from raw, noisy data. Chapter 4 proposes the data models and the formal semantics of relational operations under the chosen models. Then, the techniques for relational query processing are discussed in

Chapter 5. Chapter 6 considers complex operations presented in the form of user-defined functions (UDFs) and proposes efficient techniques to compute their output on uncertain input. Chapter 7 summarizes this thesis work and states possible directions for future work.

CHAPTER 2

SYSTEM ARCHITECTURE

This chapter presents the architectural design of the CLARO system for uncertain data processing. At a high level, CLARO adopts a *two-layer architecture*. The first layer involves cleaning and transformation of raw data, while the second performs query processing for relational operations and more complex operations such as user-defined functions.

This two-layer architecture is adopted due to the following reasons. First, raw data cleaning and transformation is application-specific and often requires extensive domain knowledge. Hence, this task may need to be addressed by different techniques for different applications. As a result, in many scientific applications, this is often done by the scientists directly. For example, in the Sloan digital sky surveys (SDSS) project [89], the scientists have their own data cooking procedure. Specifically, they repeatedly take raw images of the sky using telescopes and *cook* this data to obtain various properties of the stars and galaxies. The resulting data capture the properties such as location, redshift, color, using Gaussian distributions. Second, query processing often involves common operations across applications, since a large number of analytical queries can be expressed using relational operations or mathematical user-defined functions. Therefore, the techniques proposed in this thesis can be used in any applications as long as the queries are written using some known structures (as will be discussed in Section 5.7). Overall, compared to any one-layer approach, which does the above two tasks at once, the proposed architecture is more modular,

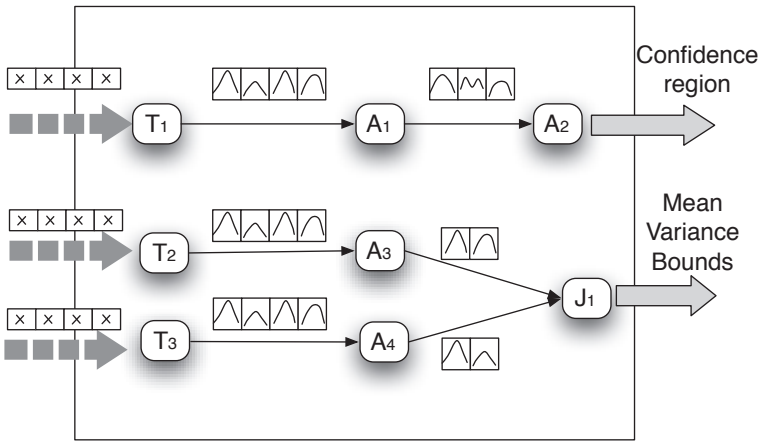


Figure 2.1. Architecture of an uncertainty-aware data management system.

hence facilitating application deployment and software usage. It also allows for new operations to be incorporated easily.

2.1 Operations in Two-Layer Architecture

This section discusses the system architecture in more detail by considering each layer separately. The CLARO system employs the general box-arrow paradigm [14] for query processing. In this paradigm, a box represents an operator and boxes are connected using arrows that represent the data flow from one operator to another. This box-arrow diagram can be either compiled from a query (e.g., Q1, Q2 and Q3 in Section 1.1) or obtained from a scientific workflow (e.g., the workflow in the CASA radar system). Figure 2.1 illustrates the operations in the CLARO system. CLARO extends the box-arrow architecture in the two following aspects.

A. Data capture and transformation (T) operators. The task of capturing uncertainty of raw data is encapsulated in a new "data capture and transformation" (T) operator. Allocated for each sensor device, a T operator can serve as an ingress operator for the stream processing network. It offers two functions:

First, it transforms raw data into a format suitable for further processing, e.g., a tuple stream with each tuple carrying an object location in the RFID application, or each tuple carrying velocity for each voxel in the weather monitoring application.

Second, it includes a *probability density function (pdf)* in each output tuple. It is important to note that in the sensing applications presented in Chapter 1, it is impossible to know the ground truth of the data of interest; rather, one has only the observations generated through the sensing process. The distributions of the output tuples then capture the prediction of the data of interest given the observed data. It is evident, as experimentally shown later in Section 5.8, that to analyze uncertainty of further processing results, the pdf of each tuple is needed—merely having mean, variance cannot fully capture uncertainty of subsequent query processing results.

In Figure 2.1, these capture and transformation operators are denoted as the T operators. Note that if the tuples produced by a T operator are independent, the pdf in each tuple completely characterizes its distribution. The pdf can also be used to capture the correlated attributes in a tuple. For example, the x and y locations in the RFID application are likely to be correlated and can be characterized by a bivariate distribution. There are also scenarios where the produced tuples are correlated, in particular, temporally correlated. The temporally correlated tuples, X_1, X_2, ..., X_n, each carry a conditional distribution $p(X_n|X_{n-1}, ..., X_{n-k})$ where $k \geq 1$. This way, a subsequent operator can construct their joint distribution, when needed, by multiplying these conditional distributions. This case is however not considered further in the scope of this thesis, and is left to future work.

B. Query processing operators. Tuples outputed from the T operators are fed into downstream query operators for further processing. The CLARO system considers two classes of operations: relational operations and user-defined functions. For relational operations, CLARO supports *selection*, *projection*, *join*, *aggregation*, and *group-by aggregation* because they are common in query processing and particularly

useful to the sensing and scientific applications. This thesis will particularly focus on aggregation in different scenarios since it is a complex operation that has been under-addressed in existing work. Besides their uses in the RFID application and computational astrophysics as illustrated in Section 1.1, these operators can also model various processing steps in the radar system. For instance, the averaging operation in moment data generation can be modeled using aggregation, and the merging of two radar streams is a special form of join.

In the CLARO system, a query operator takes a set of input tuples and produces a set of ouput tuples that contain one of the following items:

- If the query operator is the last operator, its output tuples can carry full distributions, or alternatively, statistics such as the confidence region (a set of values whose confidence is over a threshold), mean, variance, or error bounds, depending on the application.
- If the query operator is an intermediate one and its output tuples are independent, each output tuple then carries its own distribution. The output tuples can be computed exactly or approximately with bounded errors.

Figure 2.1 illustrates these operators, where A stands for aggregate operators and J stands for joins.

The second class of query processing operations that the CLARO system supports is user-defined functions (UDFs), which may be given as black boxes. Then the box-arrow diagram can be extended by adding the boxes representing these UDFs, where appropriate. When the inputs to a UDF are characterized by some distributions, it outputs are also characterized by distributions.

These two sets of operations for query processing are discussed in detail in Chapters 5 and 6 respectively.

CHAPTER 3

DATA CAPTURE AND TRANSFORMATION

This chapter addresses the first functionality of the CLARO system, which aims to capture and transform raw, noisy data into rich, queriable tuples with quantified uncertainty. The related work on this topic is first surveyed, then the main approach involving probabilistic modeling and inference used in CLARO is discussed. For concreteness, this proposed approach is demonstrated for one of the target applications, RFID tracking and monitoring. Besides, some alternative techniques for the task of data cleaning and transformation are considered.

3.1 Related Work

This section surveys the related work on data capture and transformation for the domains close to the target applications of the CLARO system. In general, this thesis work differs from recent work in the database community that applies statistical or machine learning techniques to sensor types such as temperature and light [28, 29], GPS readings [52], and RFID data from static readers [34, 50]. This is because the new types of sensor data in the applications mentioned in Section 1.1 require different, and often more complex, statistical models, and pose more demanding performance requirements. More specifically, the observed data is often of high volume and reveals the data of interest *indirectly*. The following discusses different lines of related work in more detail.

Sensor data management [28, 29, 37, 64, 87] has attracted much research lately. The sensor data can describe environmental phenomena such as temperature and

light. Techniques for data acquisition [64] and model-based processing [28] are geared towards queries natural to such data, e.g., aggregation. Such techniques consider data that is directly queriable, which is not the case that needs data capture and transformation as in the CLARO system. Model-based views over sensor streams [52] employ probabilistic inference but are restricted to GPS readings, which already reveal object locations, and consider small numbers of objects.

RFID stream processing. The HiFi project [34] offers a declarative framework for RFID data cleaning and processing. Its techniques focus on temporal and spatial smoothing of readings generated by a fixed set of static readers. SMURF [50] is a particular cleaning approach employed in HiFi; however, it does not have the benefits as the approach proposed for mobile readers in CLARO, as empirically demonstrated in Section 3.3.4.

RFID databases. RFID data management issues including inference are discussed in [19]. Cascadia [100] supports RFID-based pervasive computing with a language for specifying event patterns and techniques for extracting events from raw data and archiving them in a database. Application rules are also used to archive and compress RFID data into databases [99]. Inside RFID databases, advanced data compression techniques are available [42], data cleansing is integrated with query processing [75], and high-level information is recovered from incomplete, noisy data using known constraints and prior statistical knowledge [102]. However, these lines of work assume the use of static readers, and thus can only know that the objects are in a wide read range, not the fine-grained object locations as required by the applications presented in Section 1.1.

Object and person tracking [60, 71, 84] focuses on tracking moving targets when the association between observed features and object identities is uncertain. In the RFID setting, however, object identities are given as part of the readings; the challenge is to translate high-volume, noisy readings into clean, precise location events. Hence,

models from this research area are not suitable for the problem tackled in CLARO. Some probabilistic models are developed for GPS readings [60, 71], but these do not apply to the problem under study either, because unlike GPS, RFID readings do not reveal locations directly. Moreover, the work in this area is designed for small numbers of objects (of the order of 10 objects), and does not scale to a warehouse setting. Graphical models have also been used to predict locations and goals of a single user from GPS readings [60, 71]. These techniques are inadequate for the target applications because GPS readings reveal (noisy) locations (but RFID readings do not) and the techniques proposed for a single person do not scale to many objects.

Robotics. The structure of the proposed model is similar to FastSLAM [65] that also uses a factorized particle filter for inference. Unlike the sensors used in FastSLAM, however, RFID indicates only whether an object is nearby, not its location. The logistic sensor model employed in this thesis work requires the use of different factorization and indexing techniques than FastSLAM. RFID-equipped robots have been recently used to estimate locations of robots [54] or RFID-affixed objects [32, 44]. This line of work is not designed to support scalable stream query processing—it does not support online inference over RFID streams [32] or does so only for a small set of objects [44]. In contrast, this work employs a suite of techniques to scale inference to large numbers of objects at stream speeds. Second, modeling in previous work is limited—the sensor model is manually calibrated, which is problematic because reader performance depends greatly on the characteristics of the environment. The reader motion model is also omitted in [32], which loses the ability to correct reader location noise.

3.2 Modeling and Inference Approach

The foundation for building an uncertainty-aware DBMS is to capture the uncertainty while processing raw data close to the sensors that produce the data. This is a

task of the *data capture and transformation* (T) operator in the CLARO system. This is also referred to as *data cooking* in many scientific applications. Specifically, the objective of this process is to transform raw data to queriable data with quantified uncertainty. As mentioned in Chapter 2, this uncertainty captures the prediction of the transformed data, as it is impossible to observe the ground truth in the sensing applications.

This section describes a general approach to data capture and transformation by using machine learning modeling and inference techniques. We model the data in the transformed format as continuous random variables that cannot be directly observed (hidden variables X), and the data in the input format as continuous or discrete random variables that can be observed (evidence variables O). The task of data transformation and quantifying data uncertainty amounts to computing the conditional distribution $p(X|O)$. While this problem has traditionally been the purview of statistical machine learning [51], it poses a tremendous challenge to existing techniques due to the performance requirements of stream systems or interactive analysis. For instance, the experimental results in Section 3.3.4 show that to transform a raw RFID data stream to an object location stream, a standard probabilistic inference technique can process only 0.1 reading per second for 20 objects —this is neither efficient nor scalable for the data stream applications that we aim to support.

The main goal in the design of the data capture and transformation operator in CLARO is to choose appropriate statistical machine learning techniques and further optimize them so that they can be applied for high-volume data streams. Below, we describe the statistical techniques and optimizations employed in the CLARO system.

3.2.1 Modeling and Inference for Sensor Data Streams

The design of a data capture and transformation (T) operator consists of two steps: First, we model the underlying *data generation process* using a machine learning

technique, called *graphical modeling*, that captures how a sensor produces data from the true state of the world with various types of noise. Second, we employ *probabilistic inference* to transform observed data into data of interest based on the data generation model.

Modeling. The first step in the design of a T operator is to develop a probabilistic model that captures the dynamic and noisy data generation process. Formally, the model is a joint probability distribution over all hidden and evidence variables in the problem domain. For example, in the case of mobile RFID readers, hidden variables X are the object locations, and evidence variables O are the readings of objects, readings of shelf tags with known locations, and the reported reader location. Edges in the model capture dependencies among variables, e.g., the true object location and reader location jointly determining if a given object is observed at time t.

The resulting graphical model is further divided into several components that describe how data is generated from the state of the world, e.g., the RFID sensing process, and how the state of the world changes, e.g., the object locations change. Each component is described using a local probability distribution. For instance, a distribution for RFID sensing can be devised using logistic regression over factors such as the distance and angle between the reader and an object. Then, the theory of graphical modeling allows one to write the complete joint distribution using the product of these local distributions.

Inference. The second step is to, at time t, compute the distribution of hidden variables X_t given observations $O_{1..t}$ from the joint distribution for the data generation process. In the RFID application, $p(X_t|O_{1..t})$ captures the distribution of true object locations given their observations by an RFID reader. The challenge is to perform accurate inference at stream speed and for a large number of objects.

This problem is approached by adopting a sampling-based inference since exact inference is often intractable. A common method, called *particle filtering*, maintains

a list of samples, or *particles*, of the state of all hidden variables, and weights the samples based on all observations obtained thus far. Over time the weighted particles become an approximation of the target conditional distribution.

One of the objectives in this work is to employ optimizations for the inference algorithms when possible to achieve efficiency and scalability for high-volume data or streaming data. Section 3.3 describes the new optimizations, namely *particle factorization*, *spatial indexing*, and *belief compression*, for a particle filter tailored for the RFID domain. One factor that affects the inference results of sampling-based techniques is the number of samples, or particles, used. The user can choose a large number of samples to guarantee that inference results are highly accurate. When efficiency is highly desirable, a better approach is to explore the tradeoff between speed and accuracy. Section 3.3.4 discusses this tradeoff in more detail.

3.2.2 Approximating Result Distributions

An issue raised after inference is how to generate output tuples with *sufficient statistics* about uncertainty. For the RFID data, each tuple in the output stream describes the estimated location of an object. Some applications may require only a confidence region of the estimated location, e.g., with 90% confidence that the object is in a certain range. Some other applications, however, may require further processing of the location stream. To capture uncertainty of such processing, each tuple in the location stream must carry the full distribution of the object location. We call the distribution in each tuple the *tuple-level distribution*.

A direct way to generate a tuple-level distribution is to include in each tuple the weighted samples (particles) used in inference. However, an obvious problem with this approach is that every tuple now carries, e.g., tens to hundreds of samples. This will increase the stream volume by one or two orders of magnitude. In addition, it

will slow down further query processing because those samples need to be processed one at a time.

For both time and space efficiency, the CLARO system converts a sample-based tuple level distribution into an approximate *parametric* distribution such as a Gaussian distribution or more flexible distributions. CLARO does so by minimizing the KL divergence (a standard measure of "distance" between distributions) $\mathrm{KL}(\hat{p}\|q)$, where $\hat{p}$ is the sample-based tuple level distribution, and q is the target parametric distribution. Assume $\hat{p}$ to be a list of value-weight pairs, $\{(x_i, w_i)\}$. Consider a Gaussian distribution $\mathcal{N}(\mu, \sigma^2)$ for q. Then,

$$\begin{aligned}\mathrm{KL}(\hat{p}\|q) &= \sum_i w_i \cdot log\frac{w_i}{q(x_i)} \\ &= \sum_i w_i \cdot log(w_i \cdot \sigma \cdot \sqrt{2\pi}) + \sum_i w_i \cdot \frac{(x_i-\mu)^2}{2\sigma^2}\end{aligned}$$

Minimizing $\mathrm{KL}(\hat{p}\|q)$ allows us to find the optimal Gaussian parameters to represent $\hat{p}$. That is, $\mu = \sum_i w_i \cdot x_i$ and $\sigma^2 = \sum_i w_i \cdot (x_i - \mu)^2$. Hence, we can efficiently convert a sample-based distribution to the closest Gaussian using two scans of the list of samples. Similar formulas are available to convert sample-based distributions into multivariate Gaussians.

While approximating a sample-based distribution using a Gaussian distribution is efficient, there are scenarios where we have to consider more flexible, parametric distributions to reduce the error of such approximation. In the RFID application, an object may have recently moved from one location to another. The samples for this object's location can be temporarily spread over two locations, hence a mixture of Gaussians may be a better fit than a single Gaussian. Approximating these samples using a single Gaussian is obviously inaccurate. Then a mixture of two Gaussians may be more appropriate, in which one component of the mixture corresponds to the possibility that the object has not moved, while the other represents possible new locations

of the object. Selecting the number of mixture components, that is, the number of "humps" in the mixture, can be done using standard model selection techniques such as Akaike Information Criterion (AIC) and Bayesian Information Criterion (BIC) [3]. Both of these techniques attempt to choose a number of components that explain the data well while penalizing models that require many mixture components.

3.3 Modeling and Inferring RFID Data Streams

This section demonstrates the approach proposed above to the RFID tracking and monitoring application in order to transform raw RFID data streams to tuple streams containing object locations. [1]

3.3.1 Background

In Section 1.1, we have introduced the RFID object tracking application. In this section, we present the data capture and transformation problem for RFID data in a more precise way, which makes the context for our technical discussion on modeling and inference.

Given a stream of raw readings of RFID tags and a sequence of reader locations, both of which can be noisy, we wish to derive a clean, precise and queriable event stream where RFID tag observations are augmented with the locations of the corresponding objects. This high-level problem can be further described using the underlying physical world, the data streams from a mobile reader, and the desired output stream.

The Physical World. The physical world being monitored is a large storage area comprising shelves $\mathcal{S}$ and a set of objects $\mathcal{O}$. Both shelves and objects are affixed with RFID tags. Since the shelves are at fixed locations, we assume that the precise locations of their tags are known a priori. However, the object locations are

[1] The work described in this section was originally presented in [94].

unknown and must be determined as part of the cleaning and transformation process. Typically, objects stay on the same shelf but can sometimes move from one shelf to another. The facts of interest to the application are the (x, y, z) location of each object O_i at each time instant t.

A mobile RFID reader provides the only means to observe the physical world. Mobile readers come in two flavors—handheld readers that are used by humans to scan and monitor tagged objects (e.g., on shelves), and readers that be mounted on robots for automated monitoring and order processing (e.g., Kiva systems [55]). The mobile reader periodically scans the storage area. In each round, the reader produces readings that contain the tag ids of observed objects (usually a subset of $\mathcal{O}$) and tag ids of observed shelves (also a subset of $\mathcal{S}$). In addition, the (x, y, z) location of the reader itself at time t can be computed using a positioning technology such as indoor GPS or ultrasound [90].

Data Streams from Mobile Readers. Various readings from a mobile reader have the following characteristics:

No information about object locations. Since an RFID stream only consists of tag ids and observation times, the object locations are not observed directly.

Noisy object readings. Object readings are highly noisy. First, if an object is on the boundary of the sensing area, in the so-called *minor detection range*, the read rate is far less than 100%. Even if the object is close to the reader, in the so-called *major detection range*, objects can be missed due to environmental factors such as occluding metal objects and contention among tags. Objects can also be read unexpectedly due to reflection of radio waves by obstructing objects. Finally, mobile readers have lower read rates than fixed readers because they tend to read objects from arbitrary orientations, and certain orientations can result in poor read rates.

Uncertainty in reader locations. The exact reader location is usually uncertain. For example, even when handheld readers are coupled with indoor positioning systems

such as ultrasound locationing, the reported locations are imprecise (e.g., accuracy is about tens of centimeters for moving objects [90]). As another example, a robotic reader can measure its location using dead reckoning, essentially by counting the number of times that its wheels have revolved. But such location estimates may contain significant noise because the robot can drift sideways due to inertia or forward due to wheel slippage, as we observed in our lab deployment.

While the exact data format varies with the reader, in this work we assume that readings are produced in two separate streams: the RFID reading stream has readings `(time, tag_id of object` O_i `or tag_id of shelf` S_j`)` and the reader location stream has reports `(time, (x, y, z))`. In practice, these streams may be slightly out-of-sync in time. In our model, however, a time step (also called an *epoch*) is fairly coarse-grained, e.g., a second. This allows us to generate synchronized streams via simple low-level processing, such as assigning the same time to RFID readings produced in one epoch and taking average of multiple location updates in an epoch to produce a single update. Therefore, we consider only synchronized streams in the rest of the chapter.

Output Event Stream. The goal of data transformation is to translate noisy, primitive data streams from a mobile RFID reader into clean, precise event streams with location information. In the output stream, each event reports the location of an object as follows: `(time, tag_id of` O_i`, (x, y, z) of` O_i`)`. Events are output for not only observed objects but also objects with missed readings. In other words, the output stream not only augments the input streams with object locations but also mitigates the effect of missed readings.

Finally note that as the reader moves, it may observe an object several times from different locations. Combining such multiple readings provides valuable information about the object location. To avoid fluctuating values in the output, our system outputs an event for an object only at particular points: for example, within t seconds

R_t	True reader location at time t. Vector containing (x, y, z) position and orientation.
$\hat{R}_t$	Noisy observation of reader location at time t.
O_{ti}	True location of object i at time t. Vector containing (x, y, z) position.
$\hat{O}_{ti}$	Binary variable indicating whether object i is observed at time t
S_i	True location of shelf tag i
$\hat{S}_{ti}$	Binary variable indicating whether shelf tag i is observed at time t
$\mathbf{R}$	Matrix of all true reader locations $[R_1 R_2 \dots R_T]$
$\hat{\mathbf{R}}$	Matrix of all observed reader locations $[\hat{R}_1 \hat{R}_2 \dots \hat{R}_T]$
$\mathbf{O}_t$	Matrix of all true object locations at time t
$\mathbf{O}$	Matrix of all true object locations at all time steps
$\hat{\mathbf{O}}_t$	Binary vector $[\hat{O}_{t,1} \dots \hat{O}_{t,M}]$ of all readings at time t
$\hat{\mathbf{O}}$	Matrix of all object readings at all time steps

Table 3.1. Summary of notation used in RFID modeling and inference.

after an object was read, upon completion of a shelf scan, or upon completion of a full area scan. The choice of when to output reports is left to the discretion of the application.

3.3.2 A Probabilistic Data Generation Model

We present a probabilistic model that captures how raw data streams are generated by a mobile RFID reader from the true state of the world. Given the complexity of the problem, our model incorporates the motion of the reader, the object dynamics, and most importantly, the noisy sensing of objects and reader locations.

Formally, the world is modeled as a vector of **random variables**, which are represented as nodes in Figure 3.1. There are two types of variables: *evidence variables* that we observe in the data, and *hidden variables* that we wish to infer from the information contained in the evidence. In our application, the hidden variables are the true reader location R_t and the object locations O_{ti}, which are represented by the unshaded nodes in Figure 3.1. The evidence variables are the reported reader location $\hat{R}_t$ and the object readings $\hat{O}_{ti}$, which are indicated by the shaded nodes in Figure 3.1. For definitions of all the notation used in this section, see Table 6.1.

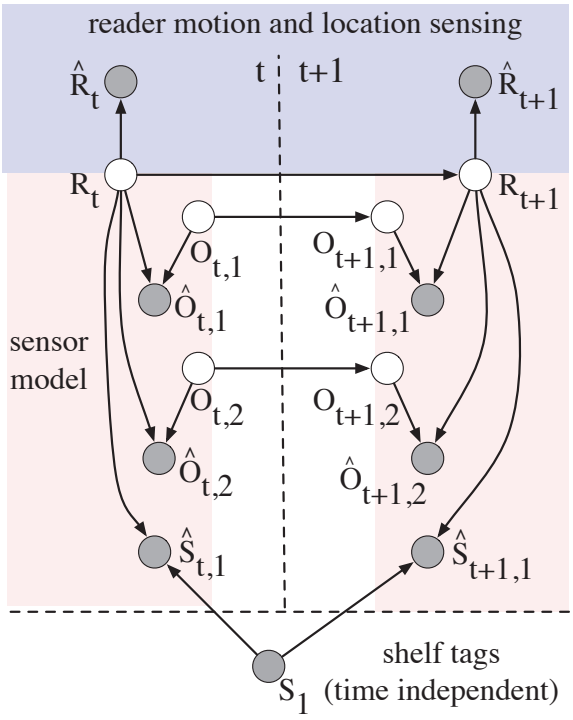


Figure 3.1. Model of reader and object locations. The shaded region at top contains the reader motion model and reader location sensing model. The lightly-shaded region at bottom contains the RFID sensor model.

The goal of this section is to define a joint probability distribution $p(\mathbf{R}, \mathbf{O}, \hat{\mathbf{R}}, \hat{\mathbf{O}})$ over both hidden and observed variables. Then, given observed values $\hat{\mathbf{R}}$ and $\hat{\mathbf{O}}$, this joint model induces a conditional distribution $p(\mathbf{R}, \mathbf{O}|\hat{\mathbf{R}}, \hat{\mathbf{O}})$ over the true locations, which can be used to predict the objects' locations. We describe various components of the proposed model in Section 3.3.2.1, how they can be combined into a single joint distribution in Section 3.3.2.2, and how model parameters can be estimated in Section 3.3.2.3.

3.3.2.1 Components of the Model

Our joint model over the entire world is divided into four components that separately model different aspects of the domain. We explain each of the component in detail below.

RFID sensor model: Given that the read rate of an RFID reader is less than 100%, it is natural to model the reader's sensing region in a probabilistic manner: each point in the sensing region has a non-zero probability that represents the likelihood of an object being read at that location. To determine the probabilistic values for different points, we can represent the sensing region as the *likelihood of reading a tag* based on the factors including the distance and angle to the reader.

Formally, we introduce a flexible parametric model that describes how the read rate of an RFID reader decays with distance and angle. Given the true location R_t of the reader and O_{ti} of the object i, the sensor model is a conditional distribution $p(\hat{O}_{ti}|O_{ti}, R_t)$ that models the probability of reading the tag of the object i. If we denote the reader location by the vector $[r_t^x, r_t^y, r_t^z]$, and the reader angle in relation to the reference coordinate frame by r_t^ϕ, then we can compute the distance d_{ti} and the angle θ_{ti} between the reader and the tag as follows:

$$\begin{aligned} \delta &= O_{t,i} - [r_t^x, r_t^y, r_t^z] \\ d_{ti} &= \sqrt{\delta^T \delta} \\ \cos\theta_{ti} &= \frac{\delta^T [\cos r_t^\phi, \sin r_t^\phi]}{d_{ti}} \end{aligned}$$

Empirically, we have found that the read rate decreases approximately quadratically with distance and with angle, so that the probability can be written as a function like $\sum_{c=0}^{2} a_c (d_{ti})^c + \sum_{c=1}^{2} b_c (\theta_{ti})^c$, where the $\{a_c\}$ and $\{b_c\}$ are coefficients. But strictly speaking, this quadratic function cannot be a probability distribution, because it is not restricted to $[0, 1]$. To fix this, we compose the quadratic function with the sigmoid function $f(x) = 1/(1 + e^{-x})$, which has the effect of squashing a real number into the interval $(0, 1)$. This yields a *logistic regression* model, which is a standard technique for probabilistic binary classification from the statistics literature. Putting this together, the sensor model is captured by:

$$p(\hat{O}_{ti} = 0|d_{ti}, \theta_{ti}) = \frac{1}{1 + \exp\{\sum_{c=0}^{2} a_c (d_{ti})^c + \sum_{c=1}^{2} b_c (\theta_{ti})^c\}} \tag{3.1}$$

The coefficients a_c and b_c are real-valued model parameters that are learned from data in a calibration step, discussed in Section 3.3.2.3 below. We expect that the distance coefficients $\{a_c\}$ will be negative, so that the read rate decreases with distance. We use the same sensor model for both the object tags and the shelf tags. The only difference is that for the shelf tags, we write the distribution as $p(\hat{S}_i = 0|d_{ti}, \theta_{ti})$, but the same model and coefficients are used in both cases.

As our results in Section 3.3.4 show, our sensor model is a flexible parametric form that can fit a variety of sensing regions, including conical and spherical regions (examples of learned sensor models are shown in Figures 3.5 and 3.7(b)).

Reader motion model: This model describes how the reader moves. We assume that the reader moves with a constant velocity that varies somewhat over time. In other words, the new location is the old location plus a noisy version of the average velocity. Formally, the new location R_t can be computed from the old location as $R_t = R_{t-1} + \Delta + \epsilon$, where Δ is the average velocity of the reader, and ϵ is the noise. The motion noise ϵ is modeled by a Gaussian random vector with zero means and diagonal covariance matrix Σ_m.

Reader location sensing model: This model describes the noise in our observations of the reader's location. For example, an RFID-equipped robot may compute its location by dead reckoning, that is, basically by counting how many times its wheels have revolved. We assume that this measurement noise is Gaussian with mean μ_s and covariance Σ_s. A more complex noise model is not necessary here, because errors in the reader location can be corrected by information from the static shelf tags as shown in our experiments.

Object location model: Objects in a warehouse are assumed to be stationary but can occasionally change locations; the object location can change with a proba-

bility α at each time t, in which case the new location is distributed uniformly across all shelves. This model can be written as a conditional distribution $p(O_{ti}|O_{t-1,i})$. It contains no distinguishing information about the object's new location, but such information is not needed: The object location model is used to temporarily create samples that will be weighted based on the actual observations in the inference process; the new object location will be eventually inferred from the readings from that location.

3.3.2.2 Formal Definition

Now we examine how the component models can be combined to define a joint model over the entire domain. By way of illustration, we first describe *how the data would be generated if the world behaved according to our model*: Assume that the initial reader location R_1 is known. Sample initial object locations $\mathbf{O}_1$ from a uniform distribution over the shelf. Then for each time step t, perform the following five steps. (1) Generate the new reader location R_t from the previous location R_{t-1} by sampling from the reader motion model $p(R_t|R_{t-1})$. (2) Generate a noisy observation $\hat{R}_t$ of the reader location from the reader location sensing model $p(\hat{R}_t|R_t)$. (3) Generate new object locations $\mathbf{O}_t$ from the object location model $p(\mathbf{O}_t|\mathbf{O}_{t-1})$. (4) Decide whether each object is observed using the sensor model. Each object i is observed with probability $p(\hat{O}_{ti}|R_t, O_{ti})$. (5) Decide whether each shelf tag is observed using the sensor model. Each shelf tag i is observed with probability $p(\hat{S}_{ti}|R_t, S_t)$.

We next give the formal description of our model. Any distribution that can be sampled in the manner above can be factorized into a product of the local probability distributions.

$$p(\mathbf{R}, \hat{\mathbf{R}}, \mathbf{O}, \hat{\mathbf{O}}|\mathbf{S}) = p(R_1, \mathbf{O}_1) \prod_t p(R_t|R_{t-1}) p(\hat{R}_t|R_t) \times \prod_{i \in \mathcal{O}} p(O_{ti}|O_{t-1,i}) p(\hat{O}_{ti}|R_t, O_{ti}) \times \prod_{i \in \mathcal{S}} p(\hat{S}_{ti}|R_t, S_t). \quad (3.2)$$

The factorization of Eq. (3.2) can be depicted as a directed acyclic graph called a *directed graphical model* or a *Bayesian network*, as shown in Figure 3.1. Our model is a particular case of a dynamic Bayesian network (DBN) [66], but with conditional probability functions specially designed for our problem.

3.3.2.3 Parameter Estimation Using Learning

In this section, we describe the self-calibration step that aims to estimate the model parameters from the observed data. The parameters of our model are the coefficients $\{a_c\} \cup \{b_c\}$ of the sensor model, the average reader velocity Δ, the variance Σ_m of the reader velocity, and the mean μ_s and variance Σ_s of the noise in reader location sensing. The sensor model in particular depends not only on the type of the reader used, but also on the specifics of the environment such as metal objects and density of tags. For example, readers perform dramatically differently when there are metal shelves, or when many tags are close together. One way to calibrate the sensor model is to perform calibration in the lab [44, 32, 50], in which the read rate is measured when a reader and an RFID tag are placed at various known distances and angles. Such manual lab calibration is not only tedious but also inaccurate in real deployments due to the change of environmental factors.

An important benefit of having a flexible parametric model is that we can automatically learn the model parameters using a small training data set collected from the same environment in which the system is to be fielded. The training data includes the observed reader locations and readings of a small set of tags, some of which are shelf tags with known locations. We perform parameter estimation using *Expectation-Maximization (EM)*, a standard method for parameter estimation in the presence of

hidden variables. In Section 3.3.4, we show that only a small number of shelf tags (e.g., less than 20) are needed to learn accurate sensor models.

3.3.3 Efficient, Scalable Inference over Streams

As noisy, raw data streams emanate from a mobile RFID reader, the task of translating them into a clean, precise event stream with location information is treated as an inference process in our work. Inference is essentially to *estimate the true locations of objects for each time t even if there are no readings returned for some of the objects*. Formally speaking, from the joint distribution $p(\mathbf{R}, \mathbf{O}, \hat{\mathbf{R}}, \hat{\mathbf{O}})$ defined previously over both the physical world and noisy readings, inference is to compute the conditional distribution $p(\mathbf{R}, \mathbf{O}|\hat{\mathbf{R}}, \hat{\mathbf{O}})$. This conditional distribution can be used to predict true object locations and optionally the true reader location.

Exact inference for our model is very challenging, because the true conditional distribution has a complex shape. Instead, we sample from the distribution approximately using a generic machine learning algorithm called *particle filtering*. How to apply particle filtering to our particular problem is described in Section 3.3.3.1. However, a naive implementation of particle filtering does not scale to the very large number of objects that would be expected in a real warehouse. We then propose optimizations to improve accuracy and performance as described in Section 3.3.3.2.

3.3.3.1 Particle Filtering

In this section, we describe the main intuitions behind sampling-based inference for our probblem. We also give a formal description of how the generic particle filtering algorithm [31] is applied to our particular probability distribution, which provides a technical context for the later extensions.

The basic idea is to maintain a weighted list of samples, each of which contains a hypothesis about the true location of each object as well as a hypothesis about the true reader location. Each sample has an associated weight, representing the

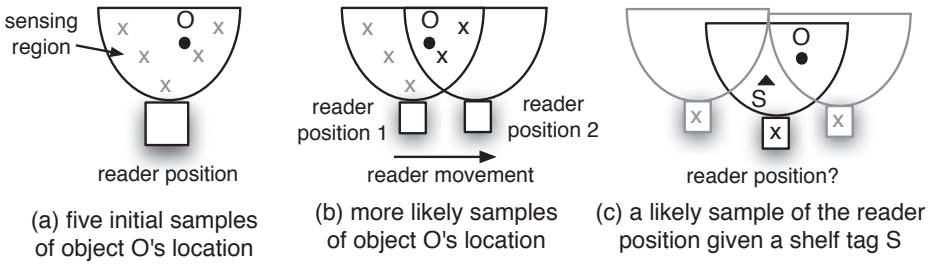


Figure 3.2. Weighting samples of object and reader locations.

likelihood of the sample being true. The weight of a sample is assigned based on the following intuitions.

As Figure 3.2(a) shows, if a reader detects the tag of object O once, the tag must be in the vicinity of the reader. We can generate multiple samples about the tag location in the reader's sensing region (or a slightly larger area) but cannot further distinguish these samples. However, if the reader detects the tag again from a nearby position, then the samples that reside in the intersection of the sensing regions at the two reader positions will be assigned higher weights (Figure 3.2(b)). Regarding the reader location, samples are weighted based on the likelihood of seeing all observed objects from that location. Of particular importance are the shelf tags with known locations. As Figure 3.2(c) shows, an observed shelf tag S can be used to distinguish good samples of the reader location, from which the reader can detect the shelf tag, from those bad samples of the reader location, from which the reader cannot.

At the next time step, these samples are updated to reflect expected changes of reader and object locations. Their weights are adjusted based on the new observations from that step. At any point, we can use this weighted list of samples as a distribution over the hidden variables, i.e., the true object locations and reader location, given the observations—exactly the result that inference aims to compute.

Formally, we denote a set of samples (termed *particles* in the literature) at time t using $x_t^1, \cdots, x_t^{(J)}$. We denote the j-th particle by a vector $x_t^{(j)} = (R_t^{(j)}, O_{t,1}^{(j)}, \ldots, O_{t,n}^{(j)})$, where $R_t^{(j)}$ is a hypothesis about the reader location and $O_{t,i}^{(j)}$ is a hypothesis about

an object location. Let the weight of $x_t^{(j)}$ be $w_t^{(j)}$. The particle filtering algorithm in our application is:

Step 1 Initialization. Generate a set of initial particles $\{x_1^{(j)} | j = 1 \dots J\}$ from the prior distribution $p(R_1, \mathbf{O}_1)$.

Step 2 Update. Let the vector y_t contain all of the observations at time t. Then for each time step t:

- *Sampling*. For each particle $x_{t-1}^{(j)}$, generate a new particle $x_t^{(j)}$ from a *proposal distribution* $q(x_t | x_{t-1}^{(j)}, y_t)$. The proposal distribution is an arbitrary distribution chosen to be easy to sample from. In this work, we use the reader motion model and object location model for sampling.
- *Weighting*. Compute a new particle weight

$$w_t^{(j)} = C w_{t-1}^{(j)} \cdot \frac{p(x_t^{(j)} | x_{t-1}^{(j)}, y_t)}{q(x_t^{(j)} | x_{t-1}^{(j)}, y_t)}, \tag{3.3}$$

  where C is a constant with respect to j, chosen so that $\sum_j w_t^{(j)} = 1$. This weight adjusts for the fact that the particles were sampled from the proposal distribution, rather than the true distribution of the model.
- *Re-sample* from the particles to reproduce the ones of high weight. Each of the new particles is selected by sampling from the set of old particles with replacement. A particle is selected with probability equal to its weight.

Step 3 Inference output. At any time step, the posterior distribution over the hidden variables can be estimated by a weighted average of the particles. More formally,

$$p(O_{ti} \,|\, \hat{\mathbf{R}}_{1\dots t}, \hat{\mathbf{O}}_{1\dots t}) \approx \sum_{j=1}^{J} w_t^{(j)} \mathbf{1}_{\{O_{ti} = O_{ti}^{(j)}\}}, \tag{3.4}$$

where $\mathbf{1}_{\{a=b\}}$ is an indicator function that is 1 if and only if $a = b$. A similar formula is used for the reader location. From these distributions, it is easy to compute any desired statistics, such as the mean, the variance, or a confidence region.

Sensible initialization of the particles is also important, because otherwise many samples will begin far away from the object's actual location. In this work, we create new particles for an object when it is seen for the first time, or at a location far away from the previous location that it was observed. At the current location, we initialize the particle locations using a uniform distribution over the read range of the reader location. The width of the region for initialization is chosen to be an over-estimate of the true range of the reader. We call this initialization *sensor model based* initialization.

A subtlety arises when an object is detected in a new location not far from the previous location. This can result from object movement within a small area (e.g., shuffling on a shelf) or erroneous readings due to the reflection of radio waves by a metal object. If the distance between the old and new locations is very small, we just use the existing particles and weight them as before. If the distance is significant, we keep half of the old particles and "move" the other half by initializing them at the new location. This way, the particles will spread out, but over time weighting and resampling will favor the particles close to the object's true location.

3.3.3.2 Optimizations for Accuracy and Performance

In this section, we describe three novel ways to augment the basic particle filtering algorithm to improve both accuracy and performance, including *particle factorization*, *spatial indexing*, and *belief compression*. The main ideas are:

- We first propose an advanced technique, particle factorization, to reduce the number of samples needed for accurate inference for a large number of objects.
- We then augment the factorized particle filter with spatial indexing structures to limit the set of objects that are actually processed at each time step.

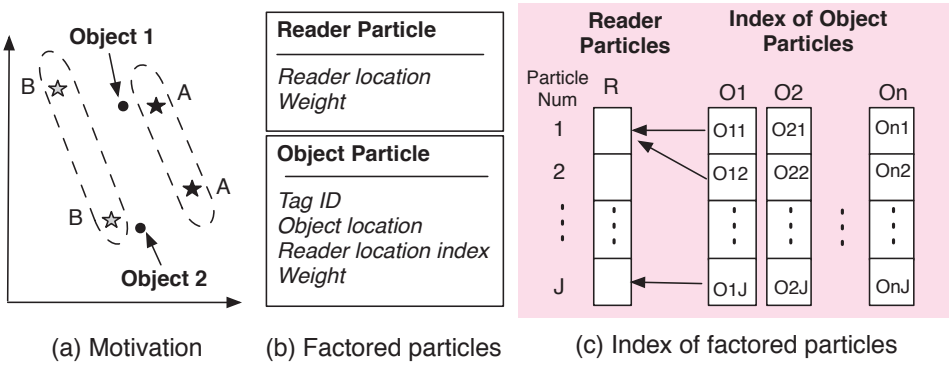


Figure 3.3. Motivation and data structures for factored particles.

- At some point in inference, the samples for an object may stabilize in a small region. In this case, we compress the sample representation of the object location into a parametric distribution to save both space and time needed for inference.

A. Particle Factorization

As mentioned above, every particle includes a sample of the locations of all objects. To get good accuracy, intuitively we expect to use a large number of particles when there are many objects. This is because even if a particle contains good location estimates for some objects, it may contain bad locations for other objects, simply through random chance in the sampling procedure. Figure 3.3(a) illustrates an example of this: Particle A (the dark stars) contains a good sampled location for Object 1 but not for Object 2. On the other hand, particle B (the light stars) contains a good sampled location for Object 2 but not Object 1. As the number of objects grows, it becomes more likely that most particles will happen to have sampled a bad location for some object. One way to overcome this problem is simply to use more particles, but this becomes prohibitively expensive, e.g., exponential, when there are large numbers of objects.

We propose an advanced technique, called *particle factorization*, that enables the particle filter to scale dramatically in the number of objects. The idea is to break a

large particle over all the objects into smaller particles over individual objects. This allows us to combine good particles from different objects and, essentially, to represent an exponentially large number of unfactored particles in the amount of space linear in the number of objects. The challenge is to ensure that the operations required by the particle filter can still be performed in this factored representation.

First we describe the data structures that we use to maintain these factored particles. As shown in Figure 3.3(b), we maintain a list of reader particles, each of which contains a hypothesis about the reader location and an associated weight. Each object particle contains a hypothesis of the object location and the reader location, a weight, and the object's tag id. We also maintain an index of object particles that maps from an object's tag id to the list of object particles for that tag id; further, each object particle refers to the corresponding reader particle via the contained pointer (see Figure 3.3(c)).

In addition to maintaining factorized particles, we also maintain factorized weights. Each reader particle $R_t^{(j)}$ has an associated weight $w_{rt}^{(j)}$. The reader particle also has a list of associated object particles $O_{ti}^{(j,1)} \ldots O_{ti}^{(j,K)}$ for each object i. Each of these object particles has a weight $w_{ti}^{(jk)}$. The semantics of the factored weights is: If we were to expand the factored representation into the exponentially-long list of unfactored particles, then the weight of the unfactored particle is the factored reader weight times all of the factored object weights.

Now we explain how these data structures can be used to efficiently implement the factorized particle filter. First, the sampling step can be performed entirely on the factored representation, which includes sampling the new reader location and sampling the object locations. To sample from the proposal distribution, for each reader particle, we sample a new reader location from the reader motion model, and then for each associated object particle, we sample its location from the object location model. Second, the weights of the new particles can also be computed

in a factored manner. The important point is that in the factored representation, the weight of a particle for object i does not depend on weights of particles for any other objects. To compute the new weights, the new incremental weight for each reader particle $w_{rt}^{(j)}$ can be computed as $p(\hat{R}_t|R_t^{(j)}) \prod_{i \in \mathcal{S}} p(\hat{S}_{ti}|R_t^{(j)}, S_t)$. The new incremental weight for an object particle $O_{ti}^{(j,k)}$ is $p(\hat{O}_{ti}|R_t^{(j)}, O_{ti}^{(j)})$. It can be shown that this weighting step is equivalent to the standard particle filtering weight step applied to the full set of unfactored particles. Mathematically, this is because our proposal distribution and our model factorize in the same way as our data structures do. To see this, consider the weight update in Formula (3.3) for unfactored particles:

$$\begin{aligned} w_t^{(j)} &= C w_{t-1}^{(j)} \cdot \frac{p(R_t^{(j)}, \mathbf{O}_t^{(j)}|R_{t-1}^{(j)}, \mathbf{O}_{t-1}^{(j)}, \hat{R}_{t-1}, \hat{\mathbf{O}}_{t-1})}{q(R_t^{(j)}, \mathbf{O}_t^{(j)}|R_{t-1}^{(j)}, \mathbf{O}_{t-1}^{(j)}, \hat{R}_{t-1}, \hat{\mathbf{O}}_{t-1})} \\ &= C w_{t-1}^{(j)} p(\hat{R}_t|R_t^{(j)}) \prod_{i \in \mathcal{S}} p(\hat{S}_{ti}|R_t^{(j)}, S_t) \prod_{i=1}^{N} p(\hat{O}_{ti}|R_t^{(j)}, O_{ti}^{(j)}) \\ &= C w_{t-1}^{(j)} \cdot w_{rt}^{(j)} \prod_{i=1}^{N} w_{ti}^{(j)} \end{aligned} \tag{3.5}$$

where in the second line, we substitute definitions; and in the last line we simply define $w_{rt}^{(j)}$ and $w_{ti}^{(j)}$ to be the corresponding terms from the previous equation. This equation shows that the weights can be computed separately for each object, with the same result as if the weight had been computed for the exponential number of unfactored particles that is implicit in our representation. Finally, performing resampling in this representation is more complicated. First we describe resampling for the object particles. Recall that each reader particle is associated with a list of object particles. For each of these lists, we perform resampling separately, sampling an object particle with probability proportional to its weight. When we resample reader locations, on the other hand, we want to favor reader particles that are associated with good object particles. To accomplish this, we consider not only the weight of the reader particle, but also the aggregate weight of its associated object particles. Formally, we resample a reader particle $R_t^{(j)}$ with probability $w_{rt}^{(j)} \prod_{i=1}^{N} \sum_{k=1}^{K(j)} w_{ti}^{(j,k)}$. When we

resample a reader location particle, we copy the locations of all the associated object particles. Although this is a computationally expensive operation, resampling the reader location occurs infrequently, so this cost is amortized.

Our factorization scheme is related to that of [68], but the main difference is that [68] avoids the issue of factorized weights by resampling at every time step. In contrast, by maintaining factorized weights, our method avoids the cost of resampling at most time steps.

B. Spatial Indexing

Even with factored particles, the inference algorithm presented so far must process all the objects in the world at every time step. This is because the weighting step described in Section 3.3.3.1 is performed for all objects, whether their tags were read or not. In this section, we introduce spatial indexing as a further approximation that dramatically reduces the processing cost. It is important to note that spatial indexing is possible only after the particles have already been factorized.

The main insight is that even if the number of objects is large, only a much smaller number of them are near the reader at any given time. If we can restrict the processing to only those objects near the reader, a significant amount of work can be saved. This intuition is more precisely described by the diagram in Figure 3.4(a), which classifies objects based on their distance from the reader location at time t (x axis) and the result of RFID sensing at t (y axis). There are four cases:

Case 1: If an object is read at time t, no matter how far it is from the reader, it should be processed in inference.

Case 2: If an object is not read at t but was read before near the current reader location, the object needs to be processed so that the particle filter can downweight the particles of the object that are very close to the current reader location.

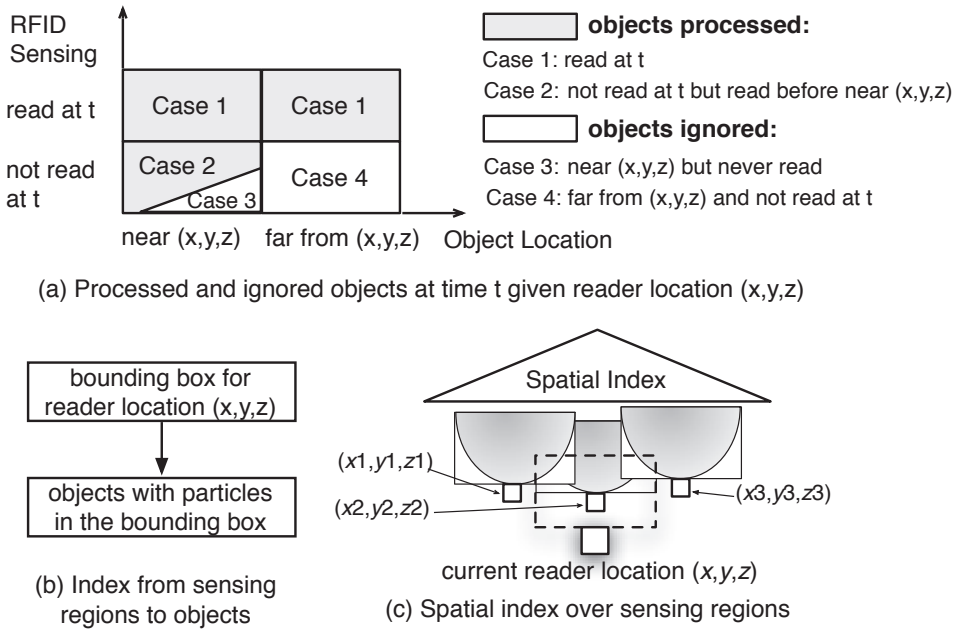


Figure 3.4. Intuitions and data structures for spatial indexing.

Case 3: If an object is near the reader but has never been detected from its current location, it is simply invisible to the inference procedure since RFID sensing is the only means of observing the world.

Case 4: Last, the object is far from the reader and indeed not detected at t. According to our sensor model, such objects have a very small (but nonzero) read probability, but rounding this probability to zero appears to be a good approximation.

Therefore, we design a spatial index to distinguish Case 2 from Case 4 so that we can save work for objects belonging to Case 4. For each reported reader location, we construct a bounding box of the sensing region. Then our index has two components. The first component, shown in Figure 3.4(b), maps from bounding boxes to the set of objects that have at least one particle within the bounding box. The second component, shown in Figure 3.4(c), is a standard spatial index (a simplified R*-tree [8]) over the bounding boxes.

At each time during inference, we construct a bounding box of the current sensing region and probe the spatial index to retrieve all potentially overlapping bounding

boxes inserted in the past. For each of those boxes, we retrieve all contained objects. This gives us the full set of objects belonging to Case 2. We run particle filtering as usual, but restrict sampling and weighting only to the objects in Cases 1 and 2.

C. Belief Compression

We next present a compression technique that can be embedded in our factorized particle filter to further reduce space consumption and improve inference speed. Recall that a weighted set of particles for each object defines a distribution over the object's location. The main advantage of the particle representation is the ability to represent arbitrary distributions. For example, when an object is first detected, its location could be anywhere within a large and oddly-shaped area. But as more readings arrive, often the location particles stabilize to a small region. If this occurs, the object location could be represented much more compactly by a parametric distribution. For example, the particle-based representation may require 1000 particles, but a three-dimension Gaussian requires only 9 real numbers to store its parameters. Therefore, compression to the parametric distribution saves considerable space. Compression can also save time as it often allows inference to use fewer particles on the compressed representation.

Per-object based compression. We first describe how an object's particles can be compressed. Suppose that a weighted set of particles over the location of object i defines a distribution $\hat{p}(O_{t,i})$ as in Eq. (3.4), and we wish to compress this into a Gaussian $q(O_{t,i})$ with mean μ and covariance matrix Σ. This can be done by minimizing the KL divergence $\mathrm{KL}(\hat{p}\|q)$, which is a standard measure of "distance" between distributions. When q is Gaussian, the KL amounts essentially to a weighted average of the squared distance between μ and the particles comprising $\hat{p}$. It can be shown that the optimal choice of q uses the sample mean and empirical covariance matrix, that is, $\mu = \sum_j w_{t,i}^{(j)} O_{t,i}^{(j)}$ and $\Sigma = \sum_j w_{t,i}^{(j)} (O_{t,i}^{(j)} - \mu)(O_{t,i}^{(j)} - \mu)^T$. The KL

divergence at these parameters measures how much is lost by compression, in the sense of the expected squared error (e.g., in squared feet) of the resulting Gaussian.

Several methods are possible for choosing individual objects to compress. One possibility is to compress an object whenever its tag has not been read for several time steps. This is applicable if an object leaving the read range means that it will not be observed for a long time; we implement this decision in the experiments below. An alternative method is to rank the uncompressed objects by the KL of the compressed representation, and compress the objects that would have the least compression error. This method can be further augmented with a threshold. That is, we only compress the particle representation if the KL is below the threshold.

Decompression (sampling) and re-compression. Later on, when a compressed object has its tag read again, we need to perform the particle filtering steps on the compressed representation. To do this, we sample a small number of particles from the Gaussian to decompress the representation. Empirically, we find that many fewer particles are required for accurate inference after decompression than for the original particle filter, because the compressed representation tends to be well-behaved. When the object leaves scope, if its particles are still well-represented by a Gaussian, it can be re-compressed.

A final note is that if the beliefs for *all* objects are compressed, then the proposed technique is an instance of the Boyen-Koller algorithm [11]. However, by employing compression only for selected objects, this technique can combine the benefits of the Gaussian and particle-based representations.

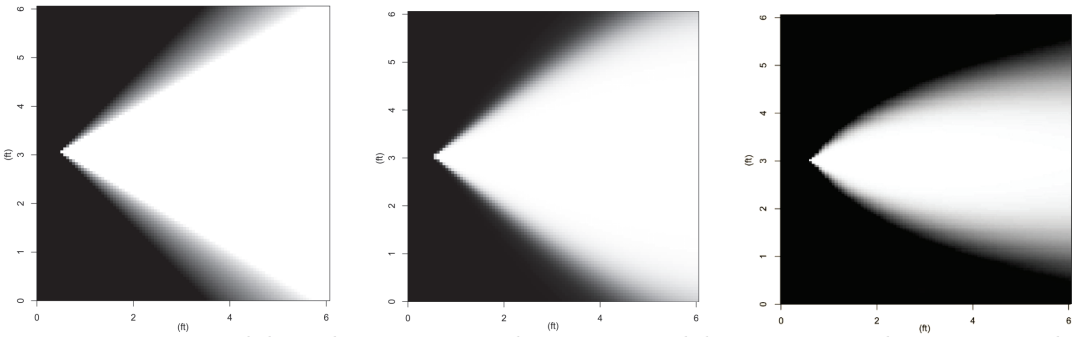
3.3.4 Experimental Results

We have implemented the proposed inference techniques in a prototype system in Java. In this section, we present a detailed analysis of this system using both real traces from mobile RFID readers and large-scale synthetic data in a simulated

warehouse scenario. Overall, the experimental results show that our system can (1) offer clean event streams with accurate location information (e.g., within a range of a few inches) and is robust to noise; (2) offer significant error reduction (e.g., an average of 54%) over SMURF [50], a state-of-the-art RFID data cleaning technique; (3) scale to tens of thousands of objects at a constant rate of over 1500 readings per second, while naive particle filtering cannot scale beyond 20 objects.

Query and Metrics. In these experiments, we ran the location update query described in Section 3.3.1 over the event stream generated by our system. This query examines the most recent event of each object, and if the location in this event differs from the previous event, outputs the tag id and new object location. To avoid fluctuating values in output, our system produced a location event 60 seconds after an object came into the scope of the reader during the current scan (although inference was running in real time). The accuracy of query output was measured using *inference error*, which is the average distance between reported object locations and true object locations. The performance metric is the average time that our system takes to process each RFID reading, indicating our system throughput.

Simulator. To obtain early insight into factors on performance and perform scalability tests, we developed a simulator for a warehouse scenario that produces synthetic RFID streams with various controlled properties. The simulated warehouse consists of consecutive shelves aligned on the y axis, with objects evenly spaced on the shelves. Both shelves and objects are affixed with RFID tags. For simplicity, we assume the same height for all tags and hence ignore the z axis. An RFID reader is mounted on a robot that moves down the y axis facing the shelves. In every epoch, it travels about 0.1 foot (which can be varied), stops, senses its current location and reads objects on the current shelf with added noise, and sends both its sensed location and the RFID readings to our system.



(a) True sensor model used by the simulator

(b) Learned sensor model using 20 shelf tags

(c) Learned sensor model using 4 shelf tags

Figure 3.5. Sensor model calibration.

RFID readings were generated using a cone-shaped sensor model as shown in Figure 3.5(a) (where white is for high read rates). The sensor model has a 30 degree open angle for the major detection range that has a uniform read rate, RR_{major}, and an additional 15 degree angle for the minor detection range whose read rate degrades from RR_{major} down to 0. The parameters for data generation include: (1) RR_{major}, by default 100%, (2) read frequency RF, by default once every second, (3) the Gaussian model for reader motion, by default μ_m=0, σ_m=.01 for both x and y dimensions, and (4) the Gaussian model for reader location sensing, by default μ_s=0, σ_m=.01 for both x and y. Each trace contains readings from a single pass of scan of all the tags unless stated otherwise.

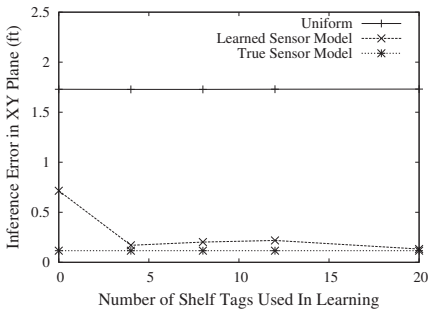
A. Model Calibration and Initial System Evaluation

In the section, we evaluate our system for its ability to calibrate the probabilistic model based on the characteristics of an RFID deployment, and test its sensitivity to some main factors. As a baseline, we also ran a method called *uniform* that uniformly randomly samples an object's location over the overlapping area of the sensor model and the shelf. This baseline is used as a bound on the worst-case inference error. We used simulated data in this set of experiments.

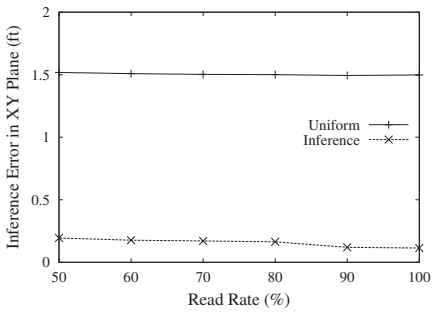
Learning RFID sensor model. As noted in Section 3.3.2.1, the most challenging part of modeling is the sensor model because it varies with the type of reader, environmental noise, etc. To evaluate our probabilistic sensor model, we used a small trace containing readings of 20 tags to learn the model using EM. To investigate the amount of information needed for learning, we varied the number of tags with known locations, assumed to be shelf tags, from 0 to 20. When fewer than 20 tags were used as shelf tags, the rest of the tags were treated as object tags whose true locations are unknown. Figure 3.5(a) shows the true sensor model used in simulation and Figure 3.5(b) shows the sensor models learned with 20 shelf tags. As observed, the sensor model learned from 20 shelf tags is very close to the true model. Such approximation degrades only gradually as we reduce the number of shelf tags. We also observe that even with 4 shelf tags, we can learn a sensor model quite close to the true model, as showed in Figure 3.5(c). When no shelf tags are used, the learned model deviates from all others because EM in this case is likely to be stuck in some local maxima.

After training, we used the learned sensor models to perform inference over a test trace with 10 object tags and 4 shelf tags, using 1000 particles per object. Most learned models (except the one using 0 shelf tag) result in small inference errors that are comparable to the results using the true model, and much better than the baseline, as shown in Figure 3.6(a). This shows that our system can indeed learn accurate sensor models from small traces with a few tags of known locations.

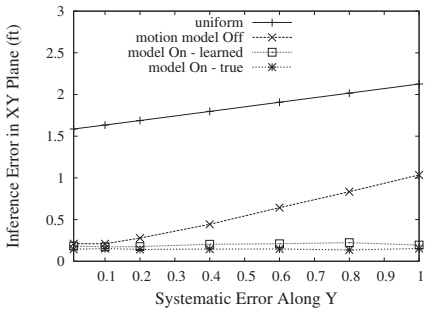
Handling RFID sensing noise. We then investigate the sensitivity of our system to RFID sensing noise, for which we varied the read rate in the reader's major detection range, RR_{major}, from 100% to 50%. Figure 3.6(b) shows the results using a trace with 16 object tags and 4 shelf tags. Our system again performs much better than the baseline, and degrades its accuracy only slowly as RR_{major} is reduced. This is because inference can intelligently exploit the facts from the past to smooth noisy



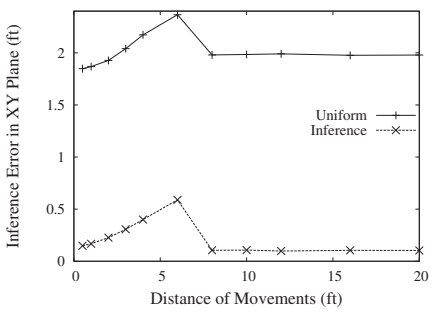
(a) Using sensor models learned with varying numbers of shelf tags



(b) Varying read rate in the reader's major detection range



(c) Varying systematic error along Y (μ_s^y) in reader location sensing (σ_s^y=.2)



(d) Moving objects: varying moving distances

Figure 3.6. Inference evaluation for synthetic RFID data.

object readings and derive object locations, hence not highly sensitive to the changes of the read rate.

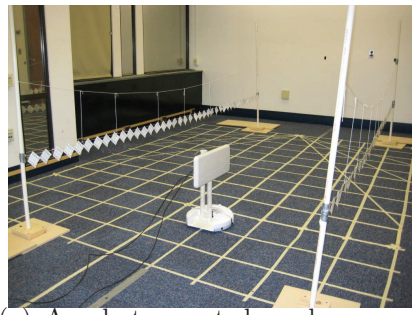
Handling reader location noise. We next evaluate our system's ability to handle reader location noise. We generated traces by varying the parameters of the reader location sensing model: the systematic error along the y axis μ_s^y was varied from 0 to 1, indicating a constant distance between the measured location and the true location; the random noise σ_s^y was set to 0.01 or 0.2, denoting little or high variation. Given the amount of noise present, we used 5000 particles per object to stabilize the performance. Figure 3.6(c) shows the results of $\sigma_s^y = 0.2$ (the figure for $\sigma_s^y = 0.01$ is similar, hence omitted).

Our system's ability to correct reader location noise is demonstrated by the difference between the curve ("motion model On-true"), which is our system using the true location sensing parameters, and the curve ("motion model Off"), which is a simplified method that uses the reported location as true location in inference. As μ_s^y increases, our system is very effective in correcting the systematic error, mostly via the evidence of shelf tags. In contrast, the lack of motion model leads to degradation almost linearly in μ_s^y. Finally, the curve ("motion model On-learned") shows that we can very well approximate the best performance by learning the parameters of the location sensing model from a small training trace.

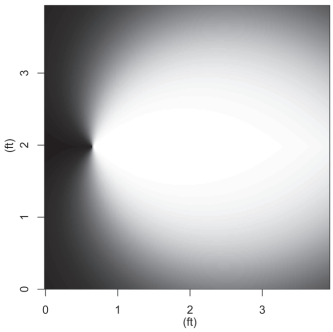
Handling moving objects. Next, we evaluate how the inference accuracy is affected by moving objects. The simulated data was generated by choosing a case of objects after some time interval and moving it to a new location. The time interval used here was 1600 seconds (our system is insensitive to this value). We varied the distance that the objects traveled from 0.5 to 20 ft. As shown in Figure 3.6(d), the inference error is sensitive to the middle range of distance, e.g., from 2 to 6 ft, where the new location is relatively close to the old location. In this case, it is hard to tell if an object has moved or not due to the reasons explained in Section 3.3.3.1. Since our method temporarily spreads the particles between the new and old locations, its accuracy is affected before the object is read enough times in the new location. When the distance of movement is large, our method discards all the old particles and recreates them from the new location, resulting in low inference error and insensitivity to further increased distance.

B. Evaluation Using a Real RFID Lab Deployment

To evaluate our system in real-world settings, we generated a lab RFID deployment as shown in Figure 3.3.4. We erected two parallel shelves (assumed to be along the y axis), containing 80 EPC Gen2 Class 1 tags spaced four inches apart. Each shelf has five evenly-spaced reference tags whose true positions are known. We constructed a



(a) A robot-mounted reader scanning two rows of tags



(b) Learned sensor model for our lab RFID reader

Timeout (ms)	Our System			SMURF (improved)			Uniform Sampling		
	X(ft)	Y(ft)	XY(ft)	X(ft)	Y(ft)	XY(ft)	X(ft)	Y(ft)	XY(ft)
250 (SS)	0.16	0.36	**0.39**	0.33	0.61	**0.69**	0.33	1.32	1.36
500 (SS)	0.18	0.47	**0.51**	0.33	0.60	**0.68**	0.33	1.57	1.60
750 (SS)	0.20	0.50	**0.54**	0.33	0.68	**0.76**	0.33	1.63	1.66
250 (LS)	0.21	0.37	**0.43**	1.30	0.61	**1.44**	1.33	1.32	1.87
500 (LS)	0.20	0.48	**0.52**	1.31	0.59	**1.44**	1.33	1.57	2.06
750 (LS)	0.21	0.49	**0.53**	1.31	0.68	**1.48**	1.32	1.63	2.09

(c) Inference error of three algorithms. SS denotes a small imagined shelf (0.66x4ft) and LS a large imagined shelf (2.6x4ft).

Figure 3.7. Evaluation of our inference technique, an improved version of SMURF, and uniform sampling using a real RFID lab deployment.

mobile reader by mounting a bi-static antenna connected to a ThingMagic Mercury5 RFID reader on an iRobot Create robot. The robot was programmed to scan one row of tags and turn around to scan the other, at a speed of .1 foot/sec with readings performed once per second. The robot computed its location using dead reckoning, with error in reported location up to 1 foot away from its true location. To emulate various read rates, we varied the reader's timeout setting—the amount of time a tag is given to respond after the initial signal is sent by the reader—from .25 to .75 second.

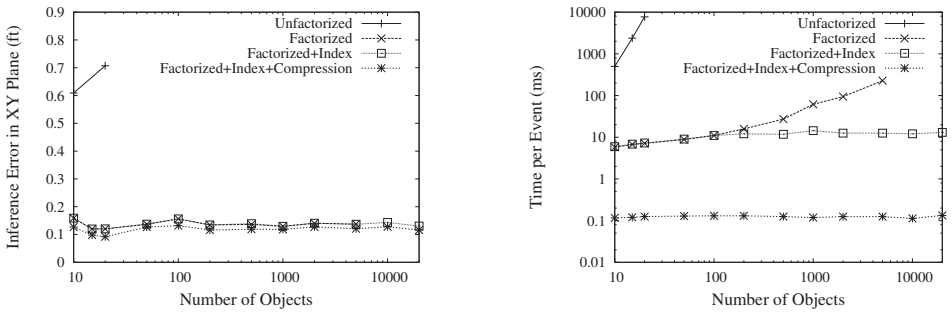
We used the shelf tags to create a training trace to learn the sensor model for our antenna. The result in Figure 3.7(b) shows that our antenna's read area is spherical with a wide minor range, whose read rate is inversely related to an object's angle from

the center of the antenna; this agrees well with manually calibrated sensor models for similar Thingmagic readers [62].

We next compare our system to SMURF [50] using our lab traces. SMURF is an adaptive smoothing technique that for each epoch, decides if a tag has moved away from the sensing area when there is a missed reading. Given that SMURF cannot directly translate RFID readings into location events, we augmented it with additional sampling: In each epoch, if SMURF decides that the tag is still in range using smoothing, a location of the tag is obtained by randomly sampling over the intersection of the read range and the shelf. At some point, if SMURF decides that the tag is no longer in scope, all sampled locations generated in those consecutive epochs are averaged to produce a location estimate. Since SMURF cannot learn the sensor model from data, we further offer the read range based on our learned model to enable sampling of the tag location.

Figure 3.7(c) shows results of our system, the improved SMURF, and uniform. The first three rows of results are from runs using a small imagined shelf, and the next three rows using a large imagined shelf. Since the read range can be large, such shelf information helps restrict the area for location sampling in all three algorithms. As can be seen, the accuracy of our system is within 0.39 to 0.54 foot. The error of SMURF is 1.3 to 1.7 times of our system when the shelf area is small and over 2.7 times when the shelf area is large. Overall, our system offers an average of 49% error reduction over SMURF.

These differences are due to two reasons: First, SMURF cannot correct the error in reported reader location present in our traces. While smoothing is effective, sampling of object location is always performed from the reported reader location. This explains the difference between our system and SMURF along y where the robot drifted significantly away from the reported location. Second, object location sampling we added to SMURF is rather primitive compared to the sampling-based inference used



(a) Scalability test - inference error (b) Scalability test - CPU time

Figure 3.8. Scalability results for synthetic RFID data.

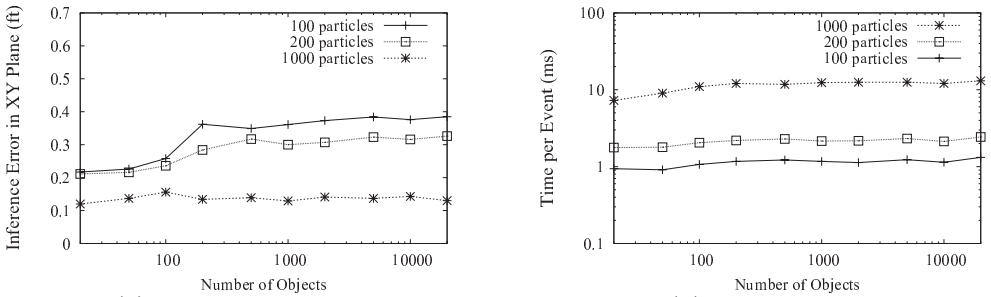
in our system. The difference in their effects is shown by the error along x: the error of SMURF is strictly half of the shelf size in x, as inaccurate as uniform sampling.

C. Scalability Evaluation Using Simulation

We next show how our system improves over basic particle filtering in scalability while maintaining high accuracy using the three advanced techniques presented in Section 3.3.3. In scalability tests, we assume that the application has an accuracy requirement of within .5 foot from the true location. We created synthetic streams from two rounds of scan of a large warehouse. All measurements were obtained from a 3Ghz dual-core Xeon processor with 6GB memory for use in Java.

Varying number of objects. We increased the number of objects from 10 to 20,000 and ran all three advanced techniques as well as the basic filter. For each technique, we selected the number of particles so as to meet the .5 foot accuracy requirement. For our factorized filter, we used 1000 particles for each object. The basic filter, however, uses unfactorized particles and needs a very large number of them, as explained in Section 3.3.3.2. We used up to 100,000 particles for the basic filter for which the experiment managed to finish.

Figures 3.8(a) and 3.8(b) report on the inference error and average time taken to process each reading (on a log scale). As can be seen, given 20 objects, the



(a) Inference error (b) CPU Time

Figure 3.9. Result on the accuracy and performance tradeoff of particle filtering.

basic filter takes about 10 second to process each reading, by using 100,000 particles yet still violating the accuracy requirement. The factorized filter, by using 1000 particles per object, well meets the accuracy requirement and improves processing cost significantly. However, this cost still degrades fast as the object count increases. Adding a spatial index to the factorized filter reduces the objects processed at each time to a small number, yielding a much reduced cost at a constant 10 msec per reading. Finally, belief compression is applied whenever an object leaves the scope of the reader. Then inference over the compressed representation in a subsequent round of scan in the warehouse can use fewer particles (in this case only 10) after decompression, leading a drastically reduced cost of 0.1 msec per reading. Neither spatial indexing nor belief compression causes obvious degradation in accuracy.

D. Exploring Accuracy and Performance Tradeoff

In the above experiments, we used a large number of samples, i.e., 1000 particles (before any belief compression) for inference, so that the accuracy is high. In general, sampling-based inference presents a fundamental tradeoff between accuracy and performance: more samples yield higher accuracy but also higher computation cost. Figures 3.9(a) and 3.9(b) show the results of performing inference for a noisy trace of RFID readings, with varying number of particles for inference. In particular, these figures confirm that the inference error reduces but the computation cost increases as

we increase the number of particles used for each object. Hence, given the application requirements, the system should choose the number of particles that yields the best performance. We now discuss one way to achieve this using the information of the shelf tags with known locations. The idea is to treat the shelf tags as other object tags, and also run inference for these shelf tags to estimate their locations. Then since we know the true locations of these tags, we can determine the inference accuracy, which in turns helps decide if the number of particles used is sufficient for the required accuracy level. That is, if the accuracy is low, we can increase the number of particles and run inference again until we reach a desired accuracy level. On the other hand, we can reduce the number of particles to reduce inference time. This idea can be implemented in real time, just as the time the shelf tags are read, which results in the dynamical exploration of speed and accuracy. The detailed implementation and evaluation is however left to future work.

3.4 Alternative Approaches to Data Capture and Transformation

This section examines some alternative techniques that can be applied to clean and transform raw input data to high-level information in other sensing applications.

A. Samples. In many applications, the observed raw data is comprised of samples of the true data, e.g., measurements of temperature, light. Here, the data of interest is revealed through the samples; however, the user may be also interested in the underlying distribution that generates the samples, to quantify the uncertainty of subsequent processing. Given these samples, a model of choice, e.g., Gaussian mixture models (GMMs), can be generated using existing tools for density estimation or function fitting. If an application models data using other distributions, e.g., the Gamma distribution, it is easy to generate samples from this distribution and then

fit a GMM as described above. More details about model fitting using GMMs are deferred to Chapter 4.

B. Correlated time series. Time series data is prevalent in many applications including our weather monitoring application. Values in a time series are temporally correlated so cannot be viewed as samples to fit e.g., GMMs. The following two techniques can be applied in this case.

Fast Fourier transform. The fast Fourier transform (FFT) translates a correlated data sequence in the time domain to an uncorrelated sequence in the frequency domain. The latter is essentially a discrete distribution that can be used to fit a GMM. Although FFT has $O(n \log n)$ complexity, where n is the sequence length, doing so for short subsequences of data does not incur high overhead. In fact, the CASA system has already applied FFT to the streams arriving at 175 Mb/sec.

Autoregressive moving average model. Another method is to use the autoregressive moving average (ARMA) model, which restricts data correlation to the past n time steps. Given such models, we can perform sampling at the frequency of once every n+1 values and feed the samples to fit our models of choice, e.g., GMMs.

We now describe this technique in more detail in the context of the tornado detection application. Here the raw data stream contains pulse data from each radar and the transformed stream contains a tuple for each voxel that has moment data including reflectivity, velocity, etc. for that voxel.

Our idea of modeling the data generation process and augmenting the transformed stream with a distribution for each tuple still applies. However, this application presents two challenges to graphical modeling that aims to completely characterize the data generation process. The first challenge is the complexity of the data generation process. As mentioned in Section 1.1, the quality of observed data is affected by many factors such as environmental noise, instability of transmit frequency, and inaccuracy of the system clock, the positioner, and the antenna. Precisely describ-

ing a data generation process involving all these factors requires significant domain knowledge. The second challenge is that the data volume in this application is extremely high, e.g., 1.66 million data items (205Mb) per second. It is an open question if sampling-based inference, even with optimizations, can keep up with such stream speeds. We observe that in this application, the transformation from the raw data to the moment data is deterministic and based on precise formulas (unlike the RFID application). Hence, we can obtain the transformed moment data stream and characterize its uncertainty using a relatively simple time series model. For a concrete example, consider the velocity data for a particular voxel. We denote the obtained velocity series using variables O_1, ..., O_t, and their true values using variables X_1, ..., X_t. We can describe the uncertainty of velocity data using $p(X_1, ..., X_t|O_1, ..., O_t)$. To do so, we consider the ARMA model that captures how X_t relates to the previous variables (autoregression) and the noise factors in the recent period (moving average) [13]. Formally,

$$X_t = \sum_{i=1}^{p} a_i X_{t-i} + \sum_{i=1}^{q} b_i \epsilon_{t-i} + \epsilon_t + C$$

where ϵ_t is the noise term for time t, and C is a constant. There are well known numeric methods that given observed data, find the ARMA(p,q) model together with the coefficients that best fits the data. These fitting methods, however, may take many passes over the data to find the best fitting. Hence, they may still be slow for the stream volume in the CASA system.

For improved efficiency, we reduce the overhead of modeling to the minimum using two techniques. First, we model X_t simply using the moving average model (MA) with no autoregression. This assumption is likely to hold for a short sequence of data: due to frequent sampling, a short sequence of data tends to describe the same phenomena, hence obviating the need of autoregression, but with correlated noise factors. As such, the work needed for modeling is to identify sequences where the MA assumption holds. Based on statistical theory, sequences obeying the MA

assumption can be identified by computing their k-lag autocorrelations, which can be performed using at most two scans of the input sequence. Second, if we know that the first query operator on the transformed stream is aggregation such as average and sum, which is true in the CASA system, we do not need to fit the MA model precisely. This is because we can use the Central Limit Theorem to obtain asymptotic results for aggregation, disregarding the precise input distributions, as long as the MA assumption holds.

CHAPTER 4

DATA MODELS AND QUERY SEMANTICS

This chapter presents the data models that lay the foundation for query processing in the CLARO system. As mentioned in Chapter 1, CLARO aims to support continuous-valued uncertain data. The data model employed for this uncertain data is Gaussian mixture models. When this data is propagated through different operations, it may generate discrete distributions as illustrated later in this chapter. Hence, CLARO also extends this data model to a more general data model, called the mixed-type model, to capture the results of these operations.

4.1 Related Work

This section gives an overview of existing data models for uncertain data in both the contexts of probabilistic databases and data streams.

Probabilistic databases have been an area of intensive recent research [4, 9, 12, 20, 23, 56, 72, 79, 82, 86, 98]. In a probabilistic database, each tuple exists with a probability; such tuple existence is essentially characterized by a Bernoulli distribution. Each tuple may further contain a distribution over a set of values, or a discrete distribution. Given such tuple models in a *discrete and finite* domain, a probabilistic database is a probability distribution over all database instances called *possible worlds* [23]. This approach however is not directly applied to continuous-valued uncertain data considered in CLARO, since the values of continuous distributions are uncountable and the probability of each possible world is simply zero.

Probabilistic stream processing has also gained research attention recently. Existing work [22, 48, 53] adopts the finite and discrete tuple model as in probabilistic databases. Specifically, each uncertain tuple or attribute is specified by its possible finite set of values and the corresponding probabilities. Due to the nature of the sensing and scientific applications mentioned in Chapter 1, which consider continuous-valued data, it is neither straightforward nor precise to adopt these discrete data models.

Models and views of sensor data. Recent work on sensor networks [28, 43] builds statistical models to capture correlation among attributes and their changes over time. Given a query, such models enable reduced costs of data acquisition and communication. FunctionDB [93] transforms discrete sensor observations into continuous functions and supports querying over these functions. This has some similarity to this thesis work; however, the choice of data models particularly enables more efficient processing to suit application-specific requirements, as shown in Chapter 5.

Probabilistic databases with continuous uncertainty. Two recent workshop papers [2, 91] consider the extension of probabilistic databases to support continuous-valued attributes. They propose modeling uncertain attributes by continuous random variables (or their probability density functions) and give examples for Gaussians and multivariate Gaussians. While mainly presenting the motivation or initial design, they made similar arguments to this thesis work for a suitable model for continuous random variables and the need to compute distributions of query answers. However, these lines of work currently do not have a complete discussion on the data models, and lack processing algorithms and performance evaluation.

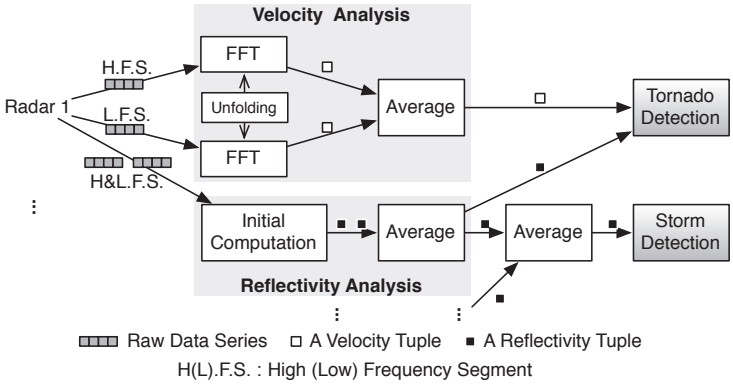


Figure 4.1. Simplified stream processing in the CASA radar system

4.2 Gaussian Mixture Model

To support continuous-valued data, the CLARO system employs a data model based on Gaussian Mixture distributions, which can capture a variety of uncertainties and further allow fast relational processing. We describe this data model next.

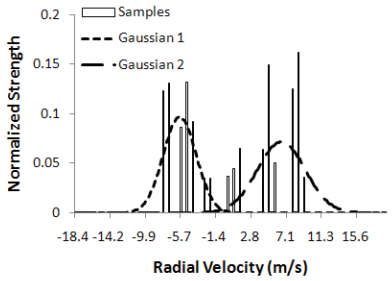
Gaussian Mixture Models (or distributions), abbreviated as GMMs, are traditionally used for data clustering and density estimation. As an instance of probability mixture models, a GMM describes a probability distribution using a convex combination of Gaussian distributions.

Definition 4.2.1. *A Gaussian Mixture Model for a continuous random variable X is a mixture of m Gaussian variables X_1, X_2, $\cdots$, X_m. The probability density function (pdf) of X is:*

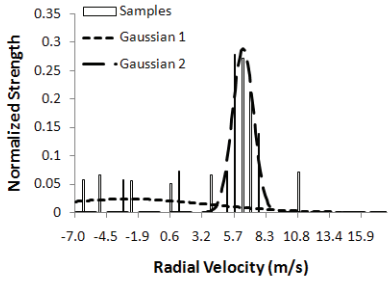
$$f_X(x) = \sum_{i=1}^{m} p_i f_{X_i}(x),$$

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

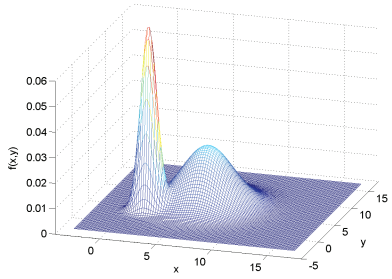
where $0 \leq p_i \leq 1$, $\sum_{i=1}^{m} p_i = 1$, and each mixture component is a Gaussian distribution with mean μ_i and variance σ_i^2.



(a) CASA: Velocity distribution after FFT in Area(430, 281.9) in a tornadic event



(b) CASA: Velocity distribution after FFT in Area(430, 282.3) in a tornadic event



(c) RFID: Location distribution of a recently moved object detected using RFID readers

Figure 4.2. Gaussian Mixture Models for real-world data collected from the target applications of CLARO

Definition 4.2.2. *A multivariate Gaussian Mixture Model for a random vector* $\mathbf{X}$ *naturally follows from the definition of multivariate Gaussian distributions.*

$$f_{\mathbf{X}}(\mathbf{x}) = \sum_{i=1}^{m} p_i f_{\mathbf{X}_i}(\mathbf{x}),$$

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

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

The CLARO system adopts Gaussian Mixture Models due to several key benefits of these models. First, GMMs are a natural extension of Gaussian distributions, which are widely used in scientific sensing and financial applications. They can be easily

accepted by end users such as the CASA scientists, who have collaborated in the tornado detection case study in this work.

Second, theoretical results have shown that GMMs can approximate any continuous distribution arbitrarily well [39]. Hence, they are suitable for modeling complex real-world distributions. For the tornado detection application, a detected bimodal distribution of velocity at the boundary between a positive velocity area and a negative velocity area is shown in Figure 4.2(a). In contrast, Figure 4.2(b) shows a velocity distribution in a positive velocity area, where one Gaussian component captures the high concentration of velocity and the other captures the noise widely spread across the entire spectrum. For the RFID application, Figure 4.2(c) shows the inferred location distribution of a recently moved object. Here, the bivariate, bimodal GMM represents the possibilities of the old and new locations using two mixture components, each component is a bivariate Gaussian modeling the joint distribution of x and y locations.

The third benefit of GMMs is efficient computation based on Gaussian properties and advanced statistical theory. First, the mean and variance of GMMs can be computed directly from those of the mixture components:

$$E[X] = \sum_{i=1}^{m} p_i E[X_i] \tag{4.1}$$

$$Var[X] = \sum_{i=1}^{m} p_i (Var[X_i] + (E[X_i])^2) - (E[X])^2 \tag{4.2}$$

Furthermore, the cumulative distribution function (cdf) of a GMM with a single variable has an analytic expression based on the known error function. Values of the error function can be approximated very accurately using numerical methods. In fact, these values are precomputed in most numerical libraries. Hence, computing $\int_a^b f_X(x)dx = F_X(b)\text{-}F_X(a)$ using the cdf incurs little cost. Other computational

benefits of GMMs, such as the characteristic functions, product distributions, and linear transformation, are described in the later relevant sections.

Gaussian Mixture Models can be generated from real-world data in a variety of ways. A discussion of how to generate distributions, which is applicable to GMMs, from different types of input data are presented in Chapter 3. We now outline a few methods that can be used to obtain GMMs. Recent work [52] and this thesis work, as shown in Chapter 3, have employed *graphical models* to infer distributions from noisy raw data. Since these distributions are often represented using weighted samples, GMMs can be generated from these samples using standard density estimation or function fitting methods. *Time series* techniques can also be used to generate GMMs from temporally correlated input data . In our case study of tornado detection, a Fast Fourier Transform (FFT) is used to translate a correlated data sequence in the time domain to an uncorrelated sequence in the frequency domain. The latter is essentially a discrete sample that can be used to fit a GMM.

4.3 Mixed-type Data Model

Input model (GMM-based). An uncertain database or data stream is an infinite sequence of tuples that conform to the schema $\mathbf{A}^d \cup \mathbf{A}^p$. The attributes in $\mathbf{A}^d$ are deterministic attributes, such as the tuple id and the fixed x-y location of a sensor. The attributes in $\mathbf{A}^p$ are continuous-valued uncertain attributes, such as the temperature and the wind velocity in an area. In each tuple, the attributes in $\mathbf{A}^p$ are modeled by a vector of *continuous random variables* $\mathbf{X}$. If the schema further has that the attributes in $\mathbf{A}^p$ can be partitioned into independent attributes, A_i^p, and groups of correlated attributes, $\mathbf{A}_j^p$, we can model A_i^p using a Gaussian mixture distribution, denoted by $f_i(x_i)$, and model $\mathbf{A}_j^p$ using a multivariate Gaussian mixture distribution, denoted by $f_j(\mathbf{x}_j)$. Then the *tuple distribution* can be written as:

$$f_{\mathbf{X}}(\mathbf{x}) = \prod_i f_i(x_i) \prod_j f_j(\mathbf{x}_j),$$

which is a multivariate Gaussian mixture distribution.

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

Definition 4.3.1. *Given a tuple with m continuous uncertain attributes, denoted by $\mathbf{A}^x$, n discrete uncertain attributes, denoted by $\mathbf{A}^y$, and other deterministic attributes $\mathbf{A}^d$, its* mixed-type distribution *g is a pair (p, f): $p \in [0, 1]$ is the tuple existence probability (TEP), and f is the joint density function for all uncertain attributes, defined as $f(\mathbf{x}, \mathbf{y}) = f_{\mathbf{A}^x|\mathbf{A}^y}(\mathbf{x}|\mathbf{y}) \cdot \mathbb{P}[\mathbf{A}^y = \mathbf{y}]$. Further, g characterizes a random vector $(\mathbf{X}, \mathbf{Y}, \mathbf{Z})$ over $(\mathbb{R}^m \times \mathbb{U}^n \times \mathbf{A}^d) \cup \{\perp\}$, where*

$$\mathbb{P}[(\mathbf{X}, \mathbf{Y}, \mathbf{Z}) = \perp] = (1 - p),$$
$$\mathbb{P}\left[\mathbf{X} \subseteq I, \mathbf{Y} = \mathbf{y}, \mathbf{Z} = \mathbf{A}^d\right] = p \cdot \int_I f(\mathbf{x}, \mathbf{y}) d\mathbf{x},\ I \subseteq \mathbb{R}^m, \mathbf{y} \in \mathbb{U}^n.$$

In the above definition, $\perp$ denotes the non-existence case of the tuple. The input model is a special case of the definition where $p = 1$ and $n = 0$.

Several notes on the mixed-type model can be made as follows. First, this data model combines the tuple-level uncertainty (i.e., TEP) with the attribute-level uncertainty. In fact, the TEP requires *every* attribute of the tuple, when used in query processing, to be modeled by a random variable: if an attribute was deterministic before, it is now modeled by a Bernoulli variable for taking its original value with probability p and $\perp$ otherwise; for the uncertain attributes, their random variables now model

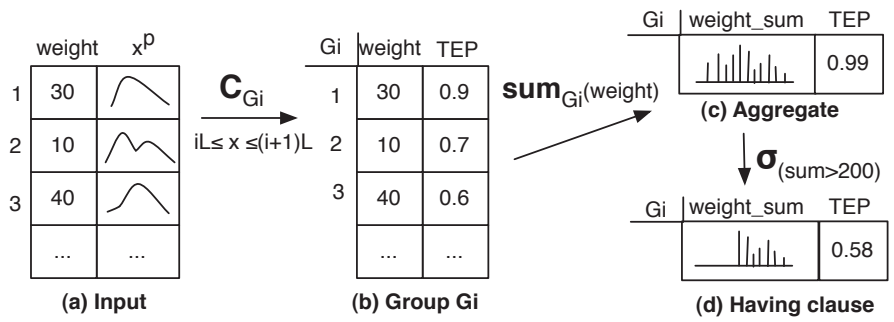


Figure 4.3. Execution of Q1 in the mixed-type model.

the joint event that the tuple exists and the attributes follow a distribution. Second, discrete uncertain attributes can emerge as derived attributes in relational processing, for example, as the result of aggregating a set of Bernoulli variables. Third, we have a general definition of the joint attribute distribution. In any implementation, it can be factorized based on the independence among attributes where each individual distribution is captured by a known parametric distribution like Gaussian mixture models or an approximate representation as proposed in Chapter 5.

In some scenarios, tuples in a stream can be correlated. Inter-tuple correlation can be modeled using joint tuple distributions or lineage [9]. The current data model in CLARO does not include such correlations for two reasons: First, while raw data is often temporally correlated, the methods that the CLARO system employs to transform raw data to tuples with distributions, such as graphical models and Fast Fourier Transform, have already taken such correlation into account. Second, given stringent performance requirements, stream systems may sometimes have to sacrifice inter-tuple correlations. For instance, the CASA tornado detection system ignores spatial correlation in any data processing before the final tornado detection, and existing probabilistic stream systems such as [52] ignore inter-tuple correlation, all for performance reasons. This work can be viewed as an optimization of the general systems when query processing does not produce correlated intermediate results. A thorough treatment of tuple correlations in stream processing is subject to future work.

We now consider an example of the mixed-type data model. Figure 4.3 illustrates the execution of query Q1, mentioned in Section 1.1, under the mixed-type model. There are three input tuples to the query, where the weight is a deterministic attribute, and the x location is a continuous-valued uncertain attribute. The group-by operation involves repeated conditioning operations on the input tuples, with a different condition for each group. For instance, the condition of the i^{th} group is $x \in [iL, (i+1)L]$, where L denotes the length of a unit area. This conditioning operation for the i^{th} group results in the table depicted in Figure 4.3(b): The truncated distribution for the x attribute is omitted since it is not used later in the query, but the probability mass covered by the truncated distribution in each tuple becomes its existence probability (i.e., TEP) in this group. The TEP translates the aggregate, $\texttt{sum}(weight)$, into a weighted sum of Bernoulli variables. The aggregate result includes a discrete distribution of the weight sum. Finally, the Having clause, modeled by a selection in relational algebra, conditions the tuple in Figure 4.3(c) with the predicate $\texttt{sum}(weight) > 200$. This will yield the reduced support of the distribution of the weight sum (where the support is the region where the pdf is non-zero) and reduced TEP of the aggregate result, as illustrated in Figure 4.3(d).

4.4 Formal Semantics of Relational Processing under Mixed-type Model

We now propose the formal semantics of relational operations under the mixed-type data model. (Note that for mixed-type tuples, a continuous uncertain attribute can follow any distribution, not restrictedly a Gaussian mixture model.) The formal semantics is crucial because it states the intended answer of each operation under the chosen data model, hence ensuring the *correctness* of query processing. A key observation is that the possible worlds semantics (PWS) does not apply to continuous random variables. First, the values of a continuous random variable are uncountable.

Second, the probability of each possible world is simply zero. Hence, we cannot construct possible worlds by enumerating values of a continuous random variable and merge the results of the possible worlds to get the result distribution. To address this issue, we propose to use *measure theory* to quantify the probabilities associated with subsets of values taken from a random variable. We first state the definition of probability space [16].

Definition 4.4.1. ***Probability Space.*** *In measure theory, a probability space of a random variable X is a triple $(S_X, \mathcal{F}_X, P_X)$ where S_X is the sample space consisting of all possible values of X, $\mathcal{F}_X$ is the σ-field over S_X, and P_X is the probability measure capturing the probability of any set in the σ-field.*

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

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

Definition 4.4.2. ***Probability Space of Mixed-type Distributions.*** *Consider a random vector $\mathbf{X}$ described by a mixed-type distribution (p, f) where p is the existence probability and f is the density function over $\mathbb{R}^m$. The probability space for $\mathbf{X}$ is characterized by: (1) the sample space $S_{\mathbf{X}} = \mathbb{R}^m \cup \{\perp\}$, where $\perp$ denotes the non-existence case, (2) the σ-field $\mathcal{F}_{\mathbf{X}}$ over $S_{\mathbf{X}}$, and (3) the probability measure $P_{\mathbf{X}}$ such*

that given any set A in the σ-field $\mathcal{F}_{\mathbf{X}}$, $P_{\mathbf{X}}(A) = (1-p)\mathbb{1}(\perp \in A) + p\int_{A\setminus\{\perp\}} f(\mathbf{x})d\mathbf{x}$, where $\mathbb{1}(\cdot)$ is the indicator function.

We next use measure theory to define the semantics of the relational operations. As known, the relational operations consider sets of tuples, or more precisely, take a set of tuples (e.g., a relational table) as input, and return another set of tuples as output. Denote the set of input tuples $\mathcal{I}$ and the set of output tuples $\mathcal{O}$. Our goal is to define the probability space of the output tuples in $\mathcal{O}$, given the probability space of input tuples in $\mathcal{I}$.

4.4.1 Projection

Projection is performed for each input tuple in the set $\mathcal{I}$ separately to get the corresponding output tuple in $\mathcal{O}$. Now consider an input tuple in $\mathcal{I}$, denoted by a random vector $(\mathbf{X}, \mathbf{Y})$, where $\mathbf{X}$ and $\mathbf{Y}$ correspond to the attributes in that input tuple ($\mathbf{X}$ and $\mathbf{Y}$ can represent more than one attribute). Let $(p, f_{\mathbf{XY}})$ denote the mixed-type distribution of this tuple. We now consider the projection of $(\mathbf{X}, \mathbf{Y})$ onto $\mathbf{Y}$, i.e., projecting out $\mathbf{X}$.

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

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

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

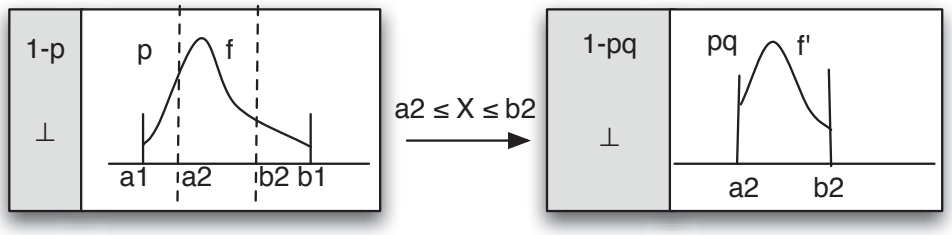


Figure 4.4. Selection under the mixed-type model

4.4.2 Selection

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

1. If $A = \{\perp\}$, then

$$P_{\bar{\mathbf{X}}}(A) = (1 - p) + p \int_{\mathbb{R}^{|\mathbf{X}|} \setminus I} f_{\mathbf{X}}(\mathbf{x}) d\mathbf{x} = 1 - pq.$$

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

Figure 4.4 illustrates the result of a selection of a tuple, with one uncertain attribute X. This uncertain attribute follows a mixed-type distribution (p, f) where the support of f is $[a_1, b_1]$. The sample space here is the real line and $\perp$ The selection on X using the condition $a_2 \leq X \leq b_2$ results in another distribution with reduced support $[a_2, b_2]$ and reduced TEP as defined above in the probability measure. In Section 5.2, we will describe the steps to obtain the result distributions.

4.4.3 Cross Product

Cross product operates on two sets of tuples, say $\mathcal{I}$ and $\mathcal{I}'$. Specifically, cross product involves pairing each tuple in $\mathcal{I}$ with each tuple in $\mathcal{I}'$. In this work, we assume the tuples in $\mathcal{I}$ and $\mathcal{I}'$ are independent of each other.

Consider two independent random vectors, $\mathbf{X}$ and $\mathbf{Y}$, representing a tuple in $\mathcal{I}$ and a tuple in $\mathcal{I}'$ respectively. Let their corresponding probability spaces be $(S_{\mathbf{X}}, \mathcal{F}_{\mathbf{X}}, P_{\mathbf{X}})$, and $(S_{\mathbf{Y}}, \mathcal{F}_{\mathbf{Y}}, P_{\mathbf{Y}})$. The cross product of $\mathbf{X}$ and $\mathbf{Y}$ corresponds to the joint distribution of the pair $(\mathbf{X}, \mathbf{Y})$. We now characterize the probability space of $\mathbf{X} \times \mathbf{Y}$, denoted as $(S_{\mathbf{XY}}, \mathcal{F}_{\mathbf{XY}}, P_{\mathbf{XY}})$. The probability space $S_{\mathbf{XY}}$ is $(\mathbb{R}^{|\mathbf{X}|} \times \mathbb{R}^{|\mathbf{Y}|}) \cup \{\perp\}$. Due to our convention of all-or-none existence among the variables, we define that the cross product exists when both $\mathbf{X}$ and $\mathbf{Y}$ exists. Therefore,

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

$$P_{\mathbf{XY}}(A) = p_{\mathbf{X}} p_{\mathbf{Y}} \iint_A f_{\mathbf{X}}(\mathbf{x}) f_{\mathbf{Y}}(\mathbf{y}) d\mathbf{x} d\mathbf{y}.$$

4.4.4 Join using Probabilistic Views

We now consider equi-join under the mixed-type data model. We note that equi-join is rare for continuous uncertain data because any pair of two values from two continuous uncertain tuples has a probability of 0. However, a special case is that one wants to (i) join two sets of tuples R and S on the attributes $\mathbf{X}$, (ii) then retrieve the attributes $\mathbf{Y}$ from S where (iii) $\mathbf{Y}$ depend on $\mathbf{X}$. We illustrate this operation using a concrete example. The query below triggers an alert when a flammable object is exposed to a high temperature. This query takes two inputs: a location stream with attributes ($time$, obj_id, $(x, y)^p$), where p denotes a probabilistic attribute, for flammable objects, and a temperature sensor stream with attributes ($time$, $sensor\_id$,

(x, y), *temp*), and joins them based on the location (x, y). It then returns the temperature of each object. The query is written as if the x and y locations were precise.

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

We first introduce the concept of a probabilistic view. Let the attributes $\mathbf{Y}$ depend on some other attributes $\mathbf{X}$ as follows. For a given $\mathbf{x}$, there is a distribution of $\mathbf{Y}$, $f_{\mathbf{Y}}(\mathbf{y}|\mathbf{X} = \mathbf{x})$. Then we say that $\mathbf{Y}$ *view-depends* on $\mathbf{X}$ and the collection of these distributions for all values of $\mathbf{x}$ is a *probabilistic view*. We denote the existence of the view with $p_{\mathbf{Y}|\mathbf{X}=\mathbf{x}}$. For $\mathbf{x}$ where the view is defined, $p_{\mathbf{Y}|\mathbf{X}=\mathbf{x}} = 1$; otherwise, $p_{\mathbf{Y}|\mathbf{X}=\mathbf{x}} = 0$. Now given a tuple with the attributes $\mathbf{X}$ following a distribution $(p_{\mathbf{X}}, f_{\mathbf{X}})$, the join of this tuple with the view is characterized by the joint distribution that pairs each value of $\mathbf{X}$ with the corresponding distribution of $\mathbf{Y}$ from the view.

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

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

$$P_{\mathbf{XY}}(A) = p_{\mathbf{X}} \iint_A p_{\mathbf{Y}|\mathbf{X}=\mathbf{x}} f_{\mathbf{X}}(\mathbf{x}) f_{\mathbf{Y}|\mathbf{X}=\mathbf{x}}(\mathbf{y}) d\mathbf{x} d\mathbf{y}.$$

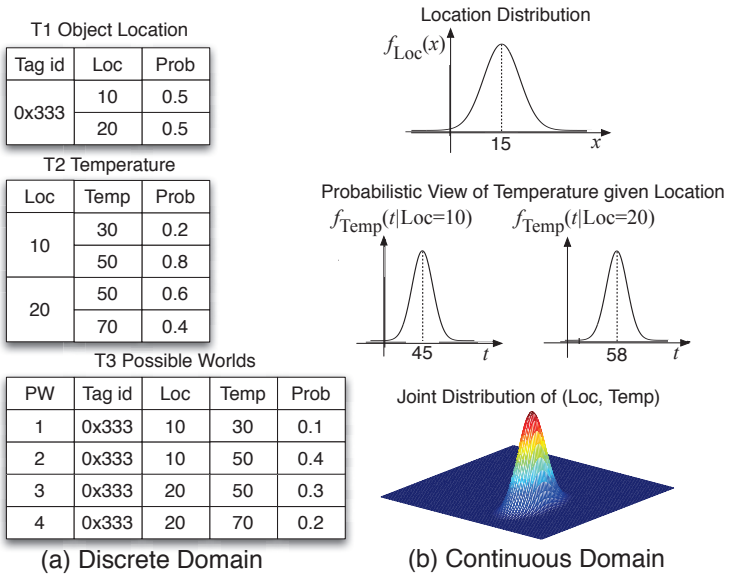


Figure 4.5. Compare equi-joins in the discrete domain (using PWS) and in the continuous domain (using a probabilistic view).

Figure 4.5 illustrates the execution of the above example query for both discrete and continuous domains, assuming one-dimensional location x. The known *possible worlds semantics* is used for the discrete case. We now illustrate the continuous case.

Let $(p_i, f_{Xi}(x))$ be the mixed-type distribution for object i, where $f_{Xi}(x)$ is the distribution of its location. Assume that at location x, a temperature sensor observes the temperature $f_{T|x}(t)$. The collection of all of these observations forms a probabilistic view of temperature given object location. In general, a probabilistic view can be characterized with both a distribution and an existence probability $p_{T|x}$. Depending on the implementation choice, $p_{T|x}$ can be set to 1, if there are enough observations for the view. Then, the query computes the temperature of each object in the location stream, which is a join with the probabilistic view. Using the above definition, we can quantify the probability space of the joint distribution of location and temperature for each object i with existence probability p_i and density function $f_i(x,t) = f_{Xi}(x) \cdot f_{T|x_i}(t)$. For the general case when the view may not exist for some locations (i.e., $p_{T|x} \leq 1$), let $q_i = \int_{\mathbb{R}} f_{Xi}(x) p_{T|x} dx$; then q_i denotes the existence

probability of the view given object i. In this case, the new TEP of the join result is $p'_i = p_i q_i$. [1]

4.4.5 Aggregation

Since aggregation is performed on a set of input tuples $\mathcal{I}$ given a common attribute, we first project each tuple in $\mathcal{I}$ on the aggregate attribute—the random variables considered in this section denote the aggregate attribute of the input tuples.

SUM. First we consider the sum of two tuples, $Y = X_1 + X_2$, where X_1 and X_2 denote the aggregate attribute in the two tuples, respectively. Under our assumption, X_1 and X_2 are independent. $X_i, i = \{1, 2\}$ follows a mixed-type distribution (p_{X_i}, f_{X_i}) and has the probability space $(\mathbb{R} \cup \{\perp\}, \mathcal{F}_{X_i}, P_{X_i})$, $i = 1, 2$. Then, the sum Y has probability space characterized with the same sample space $S_Y = \mathbb{R} \cup \{\perp\}$. Since X_1 and X_2 can either exist or not, there are four combinations of how X_1 and X_2 contribute to the sum. The probability measure is hence defined for any $A \in \mathcal{F}_Y$ as follows.

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

$$p_{X_1}(1 - p_{X_2}) \cdot \int_{x \in A} f_{X_1}(x)dx + (1 - p_{X_1})p_{X_2} \cdot \int_{x \in A} f_{X_2}(x)dx + p_{X_1}p_{X_2} \cdot \int_{x_1 + x_2 \in A} f_{X_1}(x_1)f_{X_2}(x_2)dx_1dx_2.$$

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

COUNT. In our model, `count` is equivalent to the `sum` of Bernoulli random variables. The above semantics for `sum` can be directly adapted to `count` by replacing the probability density functions (pdfs) with the probability mass functions (pmfs), and replacing integration with summation. Also, the sample space of `count` is the set

[1] The processing techniques for join using probabilistic views were presented in [97].

of natural numbers $\mathbb{N}$ and does not include $\perp$ — `count` is 0 when none of the tuples exists. This is the same as the possible worlds semantics.

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

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

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

$$
\begin{aligned}
P_Y(A) &= p_{X_1}(1 - p_{X_2})(1 - p_{X_3}) \int_{x \in A} f_{X_1}(x_1) dx_1 \\
&+ (1 - p_{X_1}) p_{X_2} (1 - p_{X_3}) \int_{x \in A} f_{X_2}(x_2) dx_2 \\
&+ (1 - p_{X_1})(1 - p_{X_2}) p_{X_3} \int_{x \in A} f_{X_3}(x_3) dx_3 \\
&+ p_{X_1} p_{X_2} (1 - p_{X_3}) \int_{(x_1 + x_2)/2 \in A} f_{X_1}(x_1) f_{X_2}(x_2) dx_1 dx_2 \\
&+ p_{X_1} (1 - p_{X_2}) p_{X_3} \int_{(x_1 + x_3)/2 \in A} f_{X_1}(x_1) f_{X_3}(x_3) dx_1 dx_3 \\
&+ (1 - p_{X_1}) p_{X_2} p_{X_3} \int_{(x_2 + x_3)/2 \in A} f_{X_2}(x_2) f_{X_3}(x_3) dx_2 dx_3 \\
&+ p_{X_1} p_{X_2} p_{X_3} \int_{(x_1 + x_2 + x_3)/3 \in A} f_{X_1}(x_1) f_{X_2}(x_2) f_{X_3}(x_3) dx_1 dx_2 dx_3
\end{aligned}
$$

If there are more than 3 random variables, the semantics is defined similarly by enumerating a number of terms exponential in the number of input tuples. This hence gives the semantics of aggregation for any table by considering all tuples in that table.

4.4.6 Group-by Aggregation

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

Now consider queries Q1 and Q2 in Section 1.1, which are group-by aggregation queries. Since they are similar, we discuss Q2 here. Each object in the input stream is characterized with the distributions of its location $f_X(x)$ and luminosity $f_Y(y)$; the tuple existence probability p is assumed to be 1. The objective is to define the probability space of `max(S.luminosity)` for each group. Let the condition of the i-th group be $x \in [iL, (i+1)L]$, where L denotes the group length. For object i, the selection probability is $q_i = \int_{x \in [iL,(i+1)L]} f_{Xi}(x)dx$. The new TEP of object i in this group is $p_i = q_i$. The result distribution of X_i has a probability space defined as in Section 4.4.2. Then we can characterize the distribution of `max`(S.luminosity), which is $M_i = \texttt{max}_k(f_{Y_k}(y) \cdot \mathbb{1}(k \in Group(i)))$, using the semantics for `max`.

4.4.7 Equivalence to Possible Worlds Semantics

For discrete random variables characterized by probability mass functions (pmfs), instead of probability density functions (pdfs), the above definitions can still apply by modifying the definition of probability measure, i.e., replacing integration with summation in the formulas for probability measure. This is the same as the possible worlds semantics (PWS), which has been defined for discrete random variables in existing work [23]. Hence, our proposed semantics is consistent with the PWS. In general, for mixed-type distributions involving both discrete and continuous attributes, the formulas for probability measures include both integration and summation.

CHAPTER 5

RELATIONAL PROCESSING OF CONTINUOUS UNCERTAIN DATA

After uncertain tuples are generated from the raw data from each sensor device and captured by continuous distributions, as described in Chapter 3, they go through various operators to produce final results. This chapter addresses common relational operators, including selection, projection, aggregation, and join, which provide general support for the target applications of the CLARO system. Given uncertain data, CLARO quantifies the result uncertainty of each query operator by computing a distribution for each result tuple.

This chapter starts by surveying existing techniques for relational processing of uncertain data. Then the new techniques for processing uncertain data under the data models of the CLARO system are presented. These techniques can be applied to both data streams and stored databases. More specifically, this chapter covers standard relational operations with a main focus on aggregates since they are complex operations without efficient existing solutions. Since aggregation has high complexity in general, CLARO aims to devise exact solutions or fast approximate solutions for performance. Finally, query planning, which considers multiple relational operations in the context of complex queries, is discussed at the end of the chapter. Given that some operations may produce approximate results, CLARO quantifies the errors of the subsequent operations after the first approximation, and then provisions an error

bound for each of those approximate operations so that an overall user-specified query accuracy requirement can be met. [1]

5.1 Related Work

There has been a recent surge of research on probabilistic databases [5, 7, 12, 17, 20, 23, 56, 59, 67, 81, 98, 101] and probabilistic stream processing [22, 48, 53]. In Chapter 4, we have stated the related work regarding data modeling. In this section, we survey closely related work from the data processing perspective.

As mentioned in Chapter 4, most of existing work models each tuple using a *discrete* random variable and evaluates queries over such tuples in a set of possible worlds. Under this *possible worlds semantics*, the following lines of work present different approaches to query processing.

Processing discrete uncertain data. Query evaluation applies a query to each possible world, and adds the probabilities of all possible worlds that return the same answer, yielding a distribution of possible query answers. Due to the discrete and finite nature, the query result distribution can be obtained by directly using axioms of probability [9, 23]. Existing work [23, 24] has identified cases when one can compute the result distribution directly from the probabilities of base tuples, which are the input to the query, and when one has to consider all possible worlds. For efficiency, existing studies attempt to avoid the computation of the result distribution by simply returning a ranked list of results [77] or using lineage to decouple and postpone the computation of result probabilities [9, 82]. These techniques are not directly applicable to the CLARO system, which aims to compute full result distributions in an efficient way.

[1] The work in this chapter was originally presented in [95, 96, 97].

Computing moments of aggregates. The existing line of work [22, 47, 49] considers inputs modeled by discrete distributions and aims to characterize the moments of the result distribution, such as mean, variance and some higher order moments. In particular, the objective is to compute aggregates for probabilistic data streams in the one-pass data stream model by considering the expectation of `max` and `min` [22, 47, 49], the expectation and variance of `sum`, and some higher moments of `count` [22]. However, this thesis work aims to compute the full result distributions, and hence cannot use these techniques.

Processing continuous uncertain data. There has been significantly less work on continuous random variables [20, 38, 46, 88]. The work [20] considers aggregation over n random variables (e.g., n uncertain tuples) and handles two variables at a time using convolution, resulting in a total of $(n-1)$ integrals. Since the number of tuples for a single aggregation can be large, this algorithm is inefficient for stream processing. The two studies [38, 46] examine sampling techniques to handle continuous random variables, which is discussed more closely below. The work [88] considers the discretization approach for common relational operations while excluding aggregation. The approximation error resulting from discretization is not quantified in [88].

Sampling techniques. The two papers [38, 46] consider sampling by generating samples over the distribution of n random variables, runing query processing using these samples, and collecting the results of these samples into a result distribution. A main concern with these algorithms is that they may need a large number of samples to achieve accuracy for arbitrary real-world distributions, hence can be too slow for high-volume streams in the sensing applications, e.g., up to 200Mb/sec for radar data.

The above techniques do not directly apply to the problem tackled in CLARO for three reasons: (i) The continuous nature of sensor data indicates that such data is better modeled using *continuous* random variables and its techniques are fundamentally different from those for discrete variables. (ii) The goal to capture result uncertainty

dictates the need of sufficient knowledge about the *entire* result distribution—such distributions are particularly important for computation of composed operators. This need precludes CLARO from using existing techniques that compromise the result distribution for simplified query processing. (*iii*) A main objective of CLARO is to support processing with high throughput while satisfying accuracy requirements, hence precluding existing solutions that have high cost or yield approximations that are hard to bound.

5.2 Basic Relational Processing under Mixed-type Model

We have presented the formal semantics, which states the intended answers of relational operations under the mixed-type data model in Section 4.4. In this section, we describe how the standard operations including selections, projections, and joins can be evaluated. As will be seen, for a substantial subset of operations, there are exact, closed-form solutions.

5.2.1 Selections

We first consider selections under the mixed-type data model. A selection involves applying a condition on some attribute of a mixed-type tuple. In Section 4.4.2, we define the semantics of selection by characterizing its result distribution using probability space. We now state how to obtain this result distribution.

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

1. Compute the selection probability q, which is the probability mass of f in the selection range I, $q = \int_{S \cap I} f(\mathbf{x}) d\mathbf{x}$.

2. Compute the new tuple existence probability, $\bar{p} = p \cdot q$.

3. Truncate the joint distribution so that its support is restricted to the intersection of the original support S and the selection range I, $\bar{S} = S \cap I$.

4. Normalize the truncated distribution, $\bar{f}(\mathbf{x}) = f(\mathbf{x})/q$.

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

5.2.2 Projections

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

5.2.3 Joins

The (traditional) type of join pairs tuples from two inputs for *inequality* comparison. (Note that the equality comparison of two continuous random variables corresponds to events with probability 0.) This join is modeled by a *cross-product* followed by a selection [88].

The CLARO system supports such joins with closed-form result distributions under the mixed-type model. More specifically, if two join attributes are uncertain and follow mixed-type distributions (p_1, f_1) and (p_2, f_2), then the join result follows a mixed-type distribution (p, f) where $p = p_1 \cdot p_2$, and $f = f_1 \cdot f_2$. If f_1 and f_2 are

GMMs, then f is a multivariate GMM. Then any subsequent selection for a specific region can be performed as detailed in Section 5.2.1. [2]

5.3 An Evaluation Framework for Aggregation

In this section, we address aggregation of continuous-valued uncertain tuples. The nature of computation for aggregates such as `sum` and `avg` is multivariate integration, which is inherently expensive. Figure 5.1 shows an example of `avg` of continuous random varibles. A state-of-the-art approach is *integral-based* [20], which integrates two variables at a time, resulting in the use of n-1 integrals to aggregate n variables. An alternative sampling-based approach [38, 88] generates samples from the input distributions and computes aggregates from these samples. However, it is hard to know the right number of samples to exploit the tradeoff between accuracy and performance, as we will show in the experiments. Another approach is to discretize continuous distributions and use existing algorithms for discrete distributions to compute `sum` and `avg` [53]. This has a time complexity $O(nD^3)$, where n is the number of tuples and D is the domain size of each tuple, hence becoming inefficient for large domains like what continuous variables require.

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

---

[2] Other types of join in CLARO were presented in [97].

We next present an evaluation framework including metrics and objectives that we will be used in the below approximation algorithms for aggregation.

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

The *point-based Variation Distance (VD)*, similar in idea to the VD in [38], is used as a distance metric for two continuous distributions.

Definition 5.3.1. *Given two probability density functions (pdf's) $f(x)$ and $g(x)$, the VD is defined as:*

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

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

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

Definition 5.3.2. *Given two one-dimensional cumulative distribution functions (CDF's)* $F, G : \mathbb{R} \to [0, 1]$*, the* KS *distance is defined as:*

$$\mathrm{KS}(F, G) = \sup_x |F(x) - G(x)|.$$

The following proposition states the relationship between the two distance metrics. The proof is shown in the Appendix.

Proposition 5.3.1. *The KS distance of two CDF's,* KS*(F, G) and the variation distance of the corresponding pdf's,* VD(f, g)*, satisfy* KS*(F, G)* $\leq$ VD(f, g)*. In some cases,* KS*(F, G) can be arbitrarily smaller than* VD(f, g)*.*

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

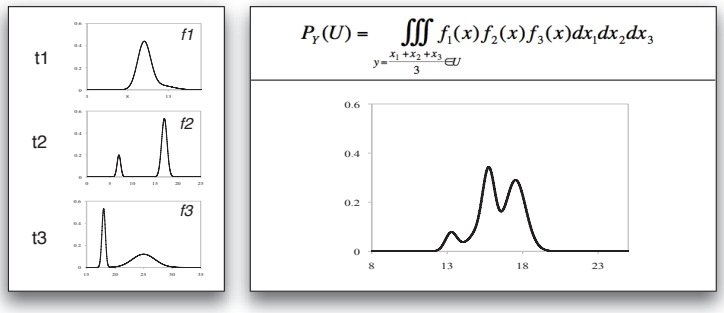


Figure 5.1. Aggregation of continuous random variables

In the following sections, we derive approximation algorithms using the KS distance. We also support the variation distance to compare our techniques with a state-of-the-art technique that uses this metric [38].

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

Definition 5.3.3. *A (randomized) algorithm computes an* (ϵ, δ) *approximation if the* KS *distance between the approximate distribution and its corresponding exact distribution is at most* ϵ *with probability* $1 - \delta$*.*

$\delta = 0$ corresponds to deterministic algorithms.

5.4 Aggregation under Gaussian Mixture Model

We now address aggregation of uncertain tuples whose existence is certain, i.e., the existence probabilities are 1, and the tuples follow Gaussian mixture models. This includes the input model of CLARO, hence a common case in the target applications. We focus on `sum` and `avg` because they are crucial to these applications but have not been sufficiently addressed in the literature.[3]

[3]The technique for `min` and `max` in CLARO is similar to that in [20], hence omitted in this thesis.

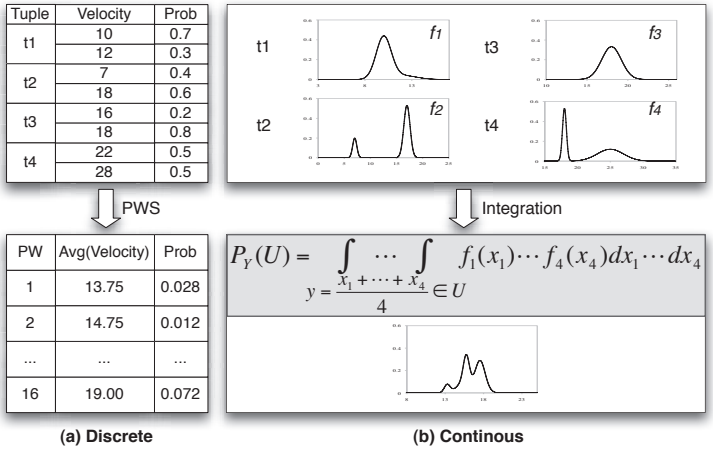


Figure 5.2. Aggregation in the discrete setting (using PWS) and in the continuous setting (using integration).

5.4.1 A Basic Algorithm

We first introduce *characteristic functions* and describe a basic algorithm to derive the result distribution for `sum` of a set of tuples. The modification to `avg` is straightforward and hence omitted in the following discussion.

In probability theory, characteristic functions (CFs) are used to "characterize" distributions. Specifically, the CF of a random variable U is defined as (chapter 2, [16]):

$$\Phi_U(t) = E[e^{iUt}], \tag{5.1}$$

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

$$f_U(x) = \frac{1}{2\pi} \int_{-\infty}^{+\infty} e^{-itx} \Phi_U(t) dt. \tag{5.2}$$

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

$$\begin{aligned}\Phi_U(t) &= E[e^{iUt}] = E[e^{i(X_1+X_2+...+X_n)t}] \\ &= E[e^{iX_1t}e^{iX_2t}...e^{iX_nt}] \\ &= \Phi_{X_1}(t)\Phi_{X_2}(t)...\Phi_{X_n}(t) \end{aligned} \tag{5.3}$$

That is, the CF of U can be written as the product of the CFs of the input tuples based on the independence assumption. This suggests a simple algorithm for `sum`: (1) Get the CF of each input tuple and take the product of these functions according to Eq. 5.3. (2) For a given value x, apply the inverse transformation at x to yield $f_U(x)$ according to Eq. 5.2. In particular, we call the inverse transformation in the second step a *parameterized integral* because it takes an argument x.

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

$$f(x) = p_1\frac{1}{\sigma_1\sqrt{2\pi}}e^{-\frac{(x-\mu_1)^2}{2\sigma_1^2}} + p_2\frac{1}{\sigma_2\sqrt{2\pi}}e^{-\frac{(x-\mu_2)^2}{2\sigma_2^2}},$$

its CF can be written directly as:

$$\Phi_X(t) = p_1e^{i\mu_1t-\frac{1}{2}\sigma_1^2t^2} + p_2e^{i\mu_2t-\frac{1}{2}\sigma_2^2t^2}.$$

Thus, Step 1 of the above algorithm does not involve any integration. The only integral required is the one for inverse transformation in Step 2. This analysis holds for all common distributions whose characteristic functions are known. This gives a boost in performance compared to the two-variable convolution method, which requires n-1 parameterized integrals [20].

The main drawback of this approach is that the formula of the result distribution involves an unresolved parameterized integral, which requires a high cost to compute and hence can be inefficient for our data stream applications. To get sufficient knowledge of the result distribution (e.g., calculating its mean and variance), one needs to

repeat the inverse transformation for a large number of points. To understand the cost of such repeated integration, we used a **numerical solution** called adaptive quadrature [83] to compute integrals. The task is to average over 10 tuples and compute the pdf values for 20 points. Even with manual optimizations, the throughput obtained is less than 200 tuples/second. This indicates that this technique is inefficient for the applications of CLARO. Moreover, it is unknown if the result distribution is a GMM.

5.4.2 Exact Derivation of Result Distributions

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

Theorem 5.4.1. *Let each $X_i, (i = 1..n)$ be a mixture of i_m components identified by the parameters $(p_{i_j}, \mu_{i_j}, \sigma_{i_j}), (j = 1..i_m)$. The result distribution for $U = \sum_{i=1}^{n} X_i$ is a Gaussian mixture of $\prod_{i=1}^{n} i_m$ components, each of which corresponds to a unique combination that takes one component from each input Gaussian mixture $\{i_j\}, (i = 1..n, j \in \{1..i_m\})$ and is identified by (p_k, μ_k, σ_k):*

$$p_k = \prod_{i=1}^{n} p_{i_j};\ \mu_k = \sum_{i=1}^{n} \mu_{i_j};\ \sigma_k = \sqrt{\sum_{i=1}^{n} \sigma_{i_j}^2}. \tag{5.4}$$

The theorem can be proved by mathematical manipulation of the inverse transformation formula, as shown in the appendix. The result subsumes the well-known linear property of Gaussian distributions. However, in the context of GMMs, we are not aware of any state-of-the-art books on mixture models [35, 39] showing this result.

This technique gives an exact solution so the accuracy is guaranteed. Let N be the number of input tuples and M be the average number of mixture components in each

input tuple. The result formula size is then $O(M^N)$. Computing each component takes $O(N)$ thus, the time complexity is $O(NM^N)$. As such, the result formula grows exponentially in the number of aggregated tuples, raising a scalability issue with this technique. We next describe approximation techniques to address this issue.

5.4.2.1 Approximation of Result Distributions

We next propose to approximate the exact result distribution by performing function fitting in the Characteristic Function (CF) space. This is based on the property that the CF of `sum` can be compactly represented as a product of n individual CFs (Eq. 5.3), rather than an exponential number of components (Eq. 5.4). Our goal is to find some Gaussian mixture distribution whose CF best fits this product function.

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

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

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

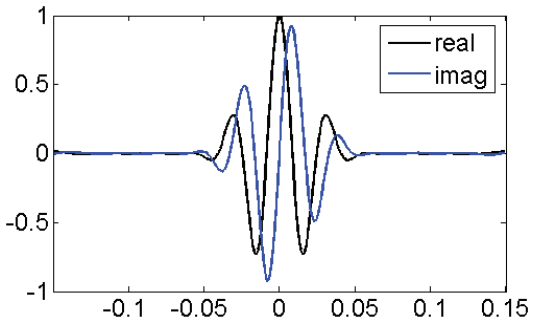


Figure 5.3. Example characteristic function for `sum` of 10 tuples.

of tuples n, the number of components per tuple (Steps 1 and 2), and the size of the result distribution K (Steps 3 and 4).

Optimizations. We further employ a suite of optimizations based on statistical theory to improve performance and accuracy. The first optimization regards the choice of an appropriate range in the domain of the CF $\Phi_{\texttt{sum}}(t)$ for fitting. The formula of $\Phi_{\texttt{sum}}(t)$ indicates that it approaches 0 fast as t moves from the center 0. Figure 5.3 shows an example CF for `sum` of 10 tuples, with both the real and imaginary parts of the CF. Given this observation, we set the range for fitting to be a small region centered around 0 so that the points taken can better capture the shape of the function to be fitted.

The second optimization concerns the initial guess of the parameters of a K-component Gaussian mixture. Due to the oscillating behavior of the CF, the fitting result is quite sensitive to this initial guess and can get stuck in local optima. We use Theorem 5.4.1 to precompute a small number of result components whose means are spread out and use them as the initial guess for fitting.

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

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

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

Relation to the Central Limit Theorem. The Central Limit Theorem (CLT) is a special case of the CF fitting algorithm. The CLT states that the sum of a sufficiently large number of independent random variables is normally distributed [16]. This gives an asymptotic result but the CF fitting algorithm dynamically determines when this result can apply. For example, a weather monitoring system sometimes requires a small number of stream segments to be averaged, for which our algorithm determines that the CLT does not apply, whereas when the number of tuples is sufficiently large (e.g., greater than 20), the result distribution starts to become a single smooth Gaussian.

5.4.2.2 Hybrid Solution

The two algorithms for aggregation, exact derivation and CF fitting, can be combined into a **hybrid** solution to exploit their advantages. When the number of tuples is small, we use exact derivation since it is fast and its formula is not complex. When the number of tuples is large enough, we switch to CF fitting. This way, we take the advantage of each algorithm in the range it performs best. We observe that the switching points among the two mainly depend on the number of tuples and less so on other data characteristics, as shown in Section 5.6.1.

5.5 Aggregation under Mixed-type Model

We have considered aggregation when the existence of tuples is certain. However, in the presence of conditioning operations, e.g., selections, the existence probabilities

of tuples become less than 1, precluding the above closed-form solution and its approximation for aggregation. In this section, we seek to directly devise approximation algorithms for aggregation of conditioned tuples.

5.5.1 Approximate Representation for CDFs

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

Definition 5.5.1. *Given a set of points* $P = \{(x_1, y_1), \ldots, (x_k, y_k)\}$ *where* $x_1 \leq x_2 \leq \ldots \leq x_k$ *and* $0 \leq y_1 \leq \ldots \leq y_k = 1$*, StepCDF*$_P$ *is the piecewise constant function that interpolates between the points whereas LinCDF*$_P$ *is a piecewise linear function that interpolates between the points:*

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

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

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

For all standard aggregates, the existence probability of the aggregate result, p, can be computed exactly. Specifically, for `count`, $p = 1$; for `sum`, `avg`, `max` and `min`, an

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

5.5.2 Bounded-Error Monte-Carlo Simulation

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

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

Theorem 5.5.1. *For any aggregate A for which there exists an exact algorithm Φ for computing aggregate A on a non-probabilistic stream, the proposed randomized algorithm Φ^* computes an (ϵ, δ) approximation of the distribution of A on a probabilistic stream. The space and update time used by Φ^* is only a factor $O(\epsilon^{-2} \log \delta^{-1})$ greater than the space and update time required by Φ.*

This proof of the theorem follows directly from the Dvoretsky-Kiefer-Wolfowitz theorem from statistics. We see that this theorem applies to aggregates such as `sum`, `count`, `avg`, `min`, and `max`. This theorem subsumes existing work based on Monte Carlo sampling [38, 46, 88] since it can determine the number of samples sufficient for meeting an accuracy requirement, in contrast to taking the number of samples as an input parameter to the algorithm.

5.5.3 Distributions of MAX and MIN

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

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

A useful property of `max` is that $F_t^M(x)$ can be easily computed for any specific value of x, if x is known ahead of time, because $F_t^M(x) = \prod_{i \in [t]} \mathbb{P}[Y_i \leq x]$. Consequently, it suffices for the algorithm to maintain a value c_x, initially 1, for each x in the universe, and on processing the t-th tuple we update c_x with $c_x \cdot \mathbb{P}[Y_t \leq x]$. This computes the exact distribution of `max` with the update cost per tuple $O(\mathbb{U})$, which is inefficient for stream processing. Probabilistic databases compute the distribution of `max` based on the *extensional semantics* [23], with the total cost of $O(t\mathbb{U})$ for a relation of t tuples; further, this is not an incremental algorithm.

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

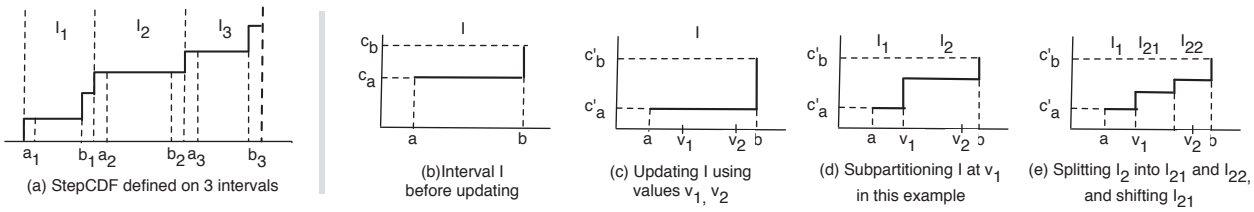


Figure 5.4. StepCDF and illustration of the basic steps of the MAX algorithm

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

Approximate Representation with Invariants. We employ an approximate representation based on StepCDF for $\tilde{F}_t^M$. The universe is partitioned into consecutive intervals: $[1, n] = \cup_i [a_i, b_i]$, where $a_{i+1} = b_i + 1$. For each interval $[a, b]$, we maintain c_a and c_b to be the *estimates of cumulative probabilities* at a and b. Each interval $[a, b]$ is then viewed as a *broad step*, which contains a straight line from a to $b-1$ and possibly a jump at b if $c_b \neq c_a$, as illustrated in intervals I_1 and I_3 in Figure 5.4(a). This yields a StepCDF defined over the point set $\{a_1, b_1, a_2, b_2, \ldots\}$.

The algorithm has the following invariants. At any point, given any interval $[a_i, b_i]$ and a constant parameter ϵ' (see Theorem 5.5.2 on how to set ϵ' as a function of the accuracy requirement ϵ) , we have:

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

Invariant (1) guarantees that the estimates of the two ends of an interval are close, so the estimate errors for the points in between can be bounded. Invariant (2) ensures that the estimates of any two adjacent intervals are separated by at least a certain

factor. Given the range $[0, 1]$ of CDF's, the number of intervals to be maintained is hence bounded, which in turn gives an upper bound on the time and space required for the algorithm.

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

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

1. Updating and Pruning: For each interval $I = [a, b]$ in the current max distribution, update its estimates with the new tuple: $c'_a = c_a \cdot \mathbb{P}[Y_t \leq a]$ and $c'_b = c_b \cdot \mathbb{P}[Y_t \leq b]$ (see Figures 5.4(b) & (c)). If after updating, $c'_b < \epsilon$, discard the interval. Note that after updating, the ratio between the estimates of the two ends can only increase.

2. Subpartitioning: This step is performed to ensure that Invariant 1 is satisfied. If updating with the new tuple results in $c'_b > c'_a(1 + \epsilon')$ for some interval $I = [a, b]$, we subpartition that interval into subintervals $I_1 = [a_1, b_1], \ldots, I_k = [a_k, b_k]$ with $a_1 = a$, $a_{i+1} = b_i + 1$, so that Invariant 1 holds (see Figure 5.4(d)). The implementation ensures that the interval is not partitioned excessively. Then, for each $x \in \{a_1, b_1, a_2, b_2, \ldots, b_k\}$, we update c_x as $c_x \mathbb{P}[Y_t \leq x]$.

3. Adjusting: This step deals with a subtle issue regarding the efficiency of the algorithm. If, among the intervals after subpartitioning, there exists an interval I_i, whose width is greater than half of the width of the original interval I, we split it into two intervals I_{i1}, I_{i2} with equal width. This step ensures that each new interval is at most half the width of I. However, this results in I_{i1} and I_{i2} having the same estimates; to ensure Invariant 2, one of the interval is shifted by a factor $\sqrt{1 + \epsilon'}$. Figure 5.4(e) illustrates this step.

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

Lemma 5.5.1. *For any interval $I = [a, b]$ of generation g and net shifting effect s, after t tuples have been processed, for $v \in \{a, b\}$,*

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

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

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

Lemma 5.5.2. *At any step in the algorithm, the number of intervals is bounded as follows: $|\mathcal{I}| \leq 2\log(\epsilon^{-1})/\log(1+\epsilon')$.*

Lemma 5.5.3. *The maximum generation of an interval is* $\log \mathbb{U}$.

Theorem 5.5.2. *The algorithm for* `max` *maintains an $(\epsilon, 0)$ approximation for F_t^M where $\epsilon' = \epsilon(1 + 0.5\epsilon e^{\epsilon})^{-1}(\log \mathbb{U} + 1)^{-1}$. The space use is $O(\epsilon^{-1} \log \mathbb{U} \ln \epsilon^{-1})$ and the update time per-tuple is $O(\min(\lambda t, \epsilon^{-1} \log \mathbb{U} \ln \epsilon^{-1}) + \lambda)$.*

Supporting Continuous Distributions. When input tuples are modeled by continuous random variables, e.g., Gaussian distributions for object locations, a general approach is to consider a real universe of size 2^{64}. The complexity is then proportional to $\log \mathbb{U} = 64$. In most applications, the universe size depends on the range and precision of measurements, often with smaller values of $\mathbb{U}$ and the number of values per tuple λ further less than $\mathbb{U}$. This combined effect can yield a fast algorithm (as shown in Section 5.6.2).

5.5.4 Distributions of SUM and COUNT

In this section, we consider the aggregates `sum` and `count`. Since `count` is a special case of `sum`, we focus on `sum` in the discussion below. We define the random variable $S_t = \sum_{i \in [t]} Y_i$ and let F_t^S be the corresponding CDF, where Y_i is the random variable corresponding to the i-th tuple. If the mean and variance of each Y_i are bounded, then the Central Limit Theorem (CLT) states that the distribution of S_t tends towards a Gaussian distribution as t goes to infinity. Later, we quantify the rate at which the distribution converges and use this to achieve an algorithmic result when there are a sufficiently large number of tuples. But for many applications, this asymptotic result cannot be applied. In the probabilistic databases where input tuples are modeled by discrete distributions, the exact distribution of `sum` can be computed using possible worlds semantics, which has an exponential complexity in the number of tuples [23]. We instead present a deterministic algorithm that efficiently computes the approximate distribution of `sum`.

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

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

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

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

SUM Algorithm. We now present a deterministic algorithm for maintaining a good approximation of F_t^S. We assume that each Y_t takes values from a finite set V_t of size at most λ, where the universe size is still $\mathbb{U}$. We treat the non-existence value

$\perp$ as if 0 since this does not affect the value of `sum`. In this case, it is easy to see that F_t^S satisfies $F_t^S(x) = \sum_{v \in V_t} F_{t-1}^S(x - v)\mathbb{P}[Y_t = v]$. Unfortunately even when $\lambda = 2$, the complexity of exactly representing F_t^S is exponential in t. Hence, to achieve space and time efficiency, we use approximate representations using quantiles as introduced above. The challenge is to quickly update the point set when each tuple arrives. We focus on the LinCDF representation with quantiles but the following algorithm also applies to StepCDF. (We observed empirically that LinCDF typically performed better.)

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

$$F(x) = \sum_{v \in V_t} \text{LinCDF}_{P_{t-1}}(x - v)\mathbb{P}[Y_t = v] \tag{5.5}$$

In this step, for each $v \in V_t$, we shift the point set P_{t-1} for the previous sum distribution by v and scale it by $\mathbb{P}[Y_t = v]$. We then compose these new point sets into λk points, in particular, using linear interpolation for the LinCDF representation. See Figure 5.5 for an illustration of this step. Now F contains a set of λk points, which we call $\bar{P}_t$. Next, simplify F to ensure efficiency in later processing while meeting the error bound ϵ' provisioned for this tuple (Theorem 5.5.3 shows how to set ϵ' by default, which is further optimized in our implementation.) To do this, we compute the k quantiles of F and return LinCDF_{P_t} where $P_t = \{(F^{-1}(i\epsilon'), i\epsilon') : 1 \leq i \leq k\}$.

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

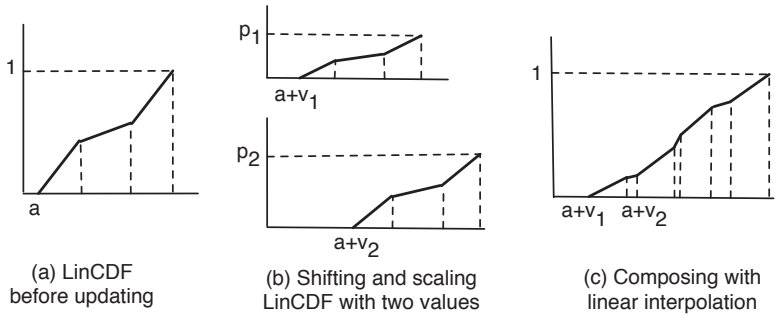


Figure 5.5. Updating step of the SUM algorithm

Theorem 5.5.3. *We can maintain an* $(\epsilon, 0)$ *approximation for* F_t^S *using* $O(\frac{1}{\epsilon'})$ *space and* $O(\frac{\lambda}{\epsilon'}\log(\frac{\lambda}{\epsilon'}))$ *time per tuple, where* $\epsilon' = \epsilon/t$.

Optimizations. We further develop three optimizations of the basic algorithm: 1) *Adaptive number of quantiles.* We observe empirically that the number of quantiles, k, needed at each step to satisfy the error bound, ϵ', is smaller than the proven bound, $1/\epsilon'$. Hence, we consider a variant of the algorithm that computes the updated set of λk points, then computes the k quantiles, and then reduces the number of quantiles, e.g., by half, if the error bound ϵ' is still met. 2) *Biased quantiles.* For distributions that are close to Gaussian, we observe that using a set of *biased quantiles* gives a better approximation. However, to meet a KS requirement, we theoretically need more biased quantiles than uniform quantiles. We propose to use both sets of quantiles in the algorithm. (3) *Central Limit Theorem.* For sufficiently large t, the distribution of F_t^S can be approximated by a Gaussian distribution. To exploit this, we just need to compute a few moments of each input distribution and check if the asymptotic result holds according to the Berry-Esseen theorem [25].

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

error from this tuple is bounded by $\epsilon_1 + \epsilon_2$ (which is due to the triangle property of the distance metric). We next discuss on how to set ϵ_1 and ϵ_2. Recall that the SUM algorithm with optimizations computes λk points and then adaptively chooses a subset of size k' that satisfies the KS error of ϵ_2, where $k' \leq 1/\epsilon_2$. Hence the cost is also proportional to $O(\lambda k') = O(1/\epsilon_1 \cdot 1/\epsilon_2)$. In practice, due to the use of mixed quantiles, k' is often smaller than $1/\epsilon_2$, especially when the distribution becomes smooth, which gives an incentive to set $\epsilon_1 > \epsilon_2$ as we observe empirically.

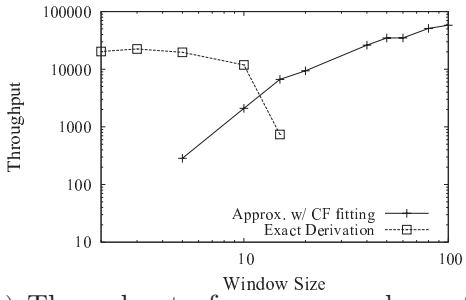
5.6 Experimental Results for Aggregation

In the following set of experiments, we evaluate our algorithms for aggregation described in Sections 5.4 and 5.5, and compare them to a histogram-based sampling technique [38].

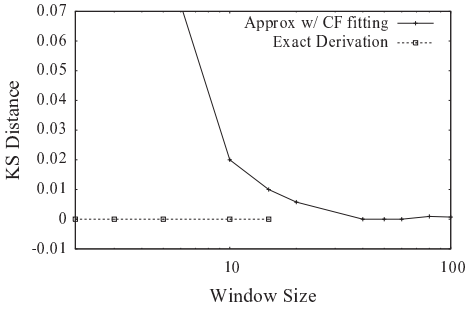
5.6.1 Aggregation under Gaussian Mixture Models

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

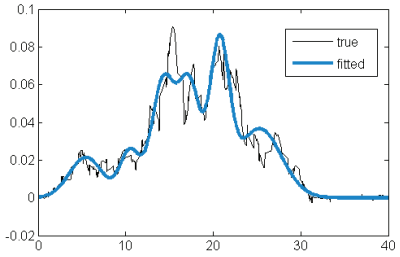
Expt 1: Compare our algorithms. We first compare two algorithms, exact derivation and approximation using CF fitting, which constitute our hybrid solution. We vary the window size, or the number of tuples under aggregation, N, since it directly affects the result distribution and the computation needed. We run the algorithms to get 100 measurements for each setting and take the average.



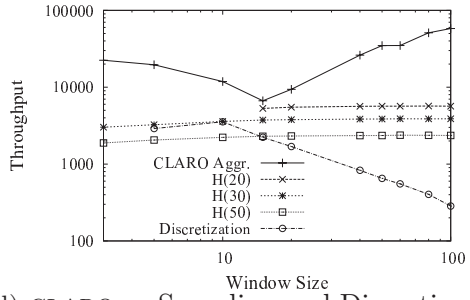
(a) Throughput of approx. and exact algorithms



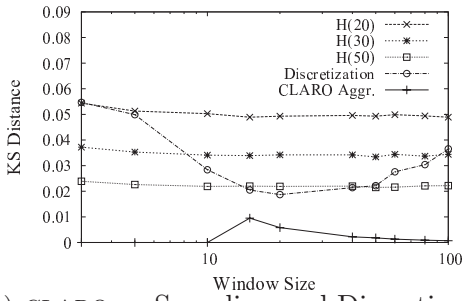
(b) Accuracy of approx. and exact algorithms



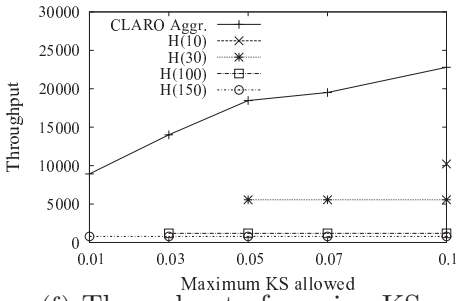
(c) A fitted distribution for 5 tuples



(d) CLARO vs Sampling and Discretization (Throughput)



(e) CLARO vs Sampling and Discretization (Accuracy)



(f) Throughput of varying KS

Figure 5.6. Experimental results for aggregation under GMMs, and histogram-based sampling $H(k)$ (with μ=50) and discretization.

Figure 5.6(a) shows the throughput results in the number of tuples processed per second. As expected, the throughput of exact derivation is high when N is small, e.g., up to 10, but deteriorates quickly afterwards because the exact result formulas generated grow exponentially in N. In contrast, CF fitting works well for large numbers of N, e.g., after 10. This is due to the smoother result distributions in this

range, hence easier to fit, and the one-time fitting cost being amortized over more tuples. We observe that both algorithms satisfy the requirement of KS ≤ 0.05 except for CF fitting in the hardest range. The hardest range is 5 to 10 tuples, where the result distributions are complex and require a mixture of many components to fit, hence low throughput. For performance purposes, we restrict the maximum number of components for each fitted distribution to be 10 (hence, the accuracy requirement may be violated in this range). An example of the true and fitted distributions for 5 tuples is shown in Figure 5.6(c). From 15 tuples onwards, the result distributions become smoother with fewer peaks.

We also run experiments using the VD metric and other workloads, and observe the same trends in accuracy, throughput, and similar crossing points between the two algorithms. (More details are shown in [96].)

The above results suggest the configuration for the hybrid solution. When the number of tuples N is 10 or below, we use exact derivation. After that, we switch to CF fitting. In addition, when N is large enough (e.g. > 30), the result distributions are mostly a smooth Gaussian and can be computed directly using the Central Limit Theorem (CLT). In Expt 3 below, we will use this as an optimization when $N \geq 30$.

Expt 2: Compare to histogram-based sampling and discretization. We now compare our hybrid solution with the histogram-based sampling algorithm [38] and the discretization approach. Similarly to the algorithm for joins, the sampling algorithm (1) generates $k \cdot \mu$ samples for each tuple, (2) performs aggregation over them to get $k \cdot \mu$ result samples, and (3) sorts the result samples and builds a histogram with k buckets and μ samples for each bucket. Since we find the accuracy of this algorithm to be more sensitive to k, we vary k among 20, 30, and 50 while fixing μ to 50. For discretization, we approximate continuous distributions using discrete points as in joins, and then use the algorithm in [53] to compute the distribution of `avg`.

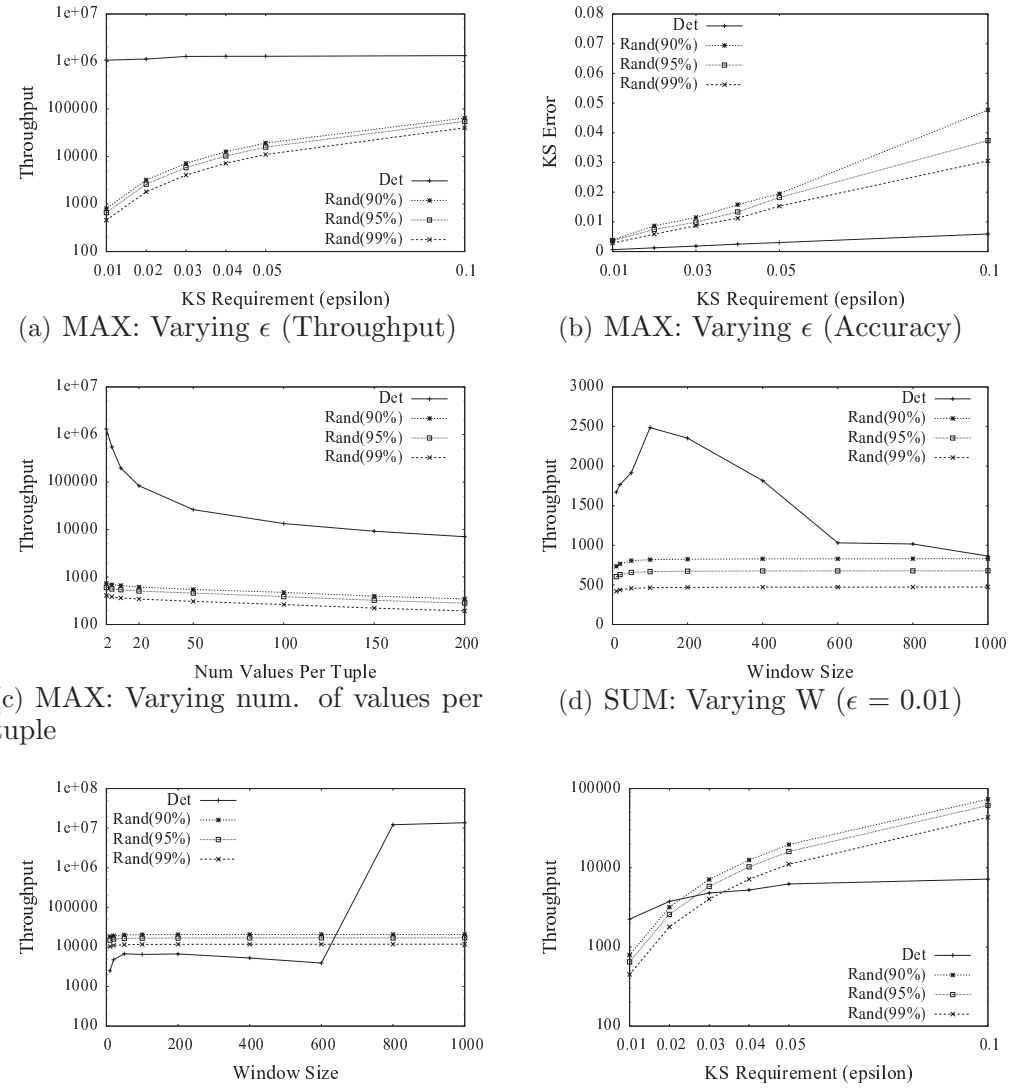
Figures 5.6(d) and 5.6(e) show the results of the three algorithms. Our hybrid algorithm outperforms all settings of histograms in both throughput and accuracy. For accuracy, only histograms with $k \geq 30$ ensures KS ≤ 0.05; $k = 20$ violates this in the "hard" range of 5 to 15 tuples (hence their throughput is omitted). The discretization approach offers no accuracy guarantee like the histogram method. So we manually varied the number of points and chose the best setting that met our accuracy requirement. The throughput of this approach is shown to be even lower than that of histograms, especially when N is large. These results confirm the advantages of our algorithm over sampling and discretization since we can adapt to a given accuracy requirement while optimizing for throughput.

Expt 3: Vary the KS requirement. To further study our adaptivity to accuracy requirements, we vary KS from 0.01 to 0.1. The window size N is chosen randomly from the range $[2, 50]$, so that we can examine different ranges of the hybrid solution. Figure 5.6(f) shows the throughput (where the KS requirement is met). Our algorithm outperforms the histogram algorithm for all values of the KS requirement by at least three times. Moreover, we can adapt to given accuracy requirements while it is unknown if a setting of the histogram algorithm can satisfy these requirements in advance.

5.6.2 Aggregation under Mixed-type Model

We now evaluate the techniques proposed for the CLARO system when the tuple existence is uncertain. We use simulated uncertain data streams, whose parameters used in this study are: the accuracy requirement (ϵ, δ), the (tumbling) window size W, the number of values per tuple λ including the non-existence case (by default $\lambda = 3$), and the universe size $\mathbb{U}$ (by default, $\mathbb{U}$=10^6).

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



(a) MAX: Varying ϵ (Throughput)

(b) MAX: Varying ϵ (Accuracy)

(c) MAX: Varying num. of values per tuple

(d) SUM: Varying W ($\epsilon = 0.01$)

(e) SUM: Varying W ($\epsilon = 0.05$)

(f) SUM: Varying ϵ (Throughput)

Figure 5.7. Experimental results for MAX, SUM under mixed-type models.

where 1-δ=0.9, 0.95, or 0.99. The reported results are averaged from a large number of measurements, i.e., 500, after the warmup phase.

We first vary the error bound ϵ in a common range [0.01, 0.1]. W is uniformly sampled from [10, 1000]. Figure 5.7(a) shows the throughput of the algorithms. The deterministic algorithm, $\mathrm{D_{max}}$, is 10 to 1000 times faster than the randomized algorithm, Rand, for all ϵ values tested. This is because $\mathrm{D_{max}}$ can use a small number of intervals to approximate the distribution (e.g., 20-50), whereas Rand uses hundreds

to tens of thousands samples, hence worse performance. We also observe that $\mathrm{D}_{\max}$ is more accurate than Rand.

We next study the effect of the number of values per tuple, λ. We vary λ from 2 to 200, and set $W = 100$ and $\epsilon = 0.01$. Figure 5.7(c) shows the throughput results. As expected, the cost of $\mathrm{D}_{\max}$ increases with λ due to the costs of the first two steps of $\mathrm{D}_{\max}$ depending on λ. However, the number of intervals in the approximate `max` distribution does not increase linearly in λ—it is bounded according to Theorem 5.5.2. Overall, the throughput of $\mathrm{D}_{\max}$ is better than that of Rand by at least one order of magnitude.

Evaluation of SUM. We evaluate the performance of the deterministic algorithm for `sum`, $\mathrm{D}_{\mathrm{sum}}$, using the optimizations shown in Section 5.5.4, and the randomized algorithm, Rand.

We vary W from 10 to 1000 for two values of ϵ, 0.01 and 0.05. Figures 5.7(d) and 5.7(e) show the throughput of both algorithms. For $\epsilon = 0.01$, $\mathrm{D}_{\mathrm{sum}}$ is faster than Rand in all settings because Rand uses a number of samples increasing quadratically in $1/\epsilon$, but $\mathrm{D}_{\mathrm{sum}}$ uses much less. The throughput of $\mathrm{D}_{\mathrm{sum}}$ decreases with W because the additive error bound of $\mathrm{D}_{\mathrm{sum}}$ requires provisioning error bounds to W tuples. For $\epsilon = 0.05$, $\mathrm{D}_{\mathrm{sum}}$ is slightly slower than Rand for $W \leq 600$ due to the reduced benefit from ϵ. However, for larger values of W, CLT applies, yielding a high throughput of millions of tuples per second. If we keep increasing ϵ, CLT starts to apply earlier, e.g., when $W = 150$ for $\epsilon = 0.1$.

We then vary ϵ from 0.01 to 0.1. W is uniformly taken from $[1, 100]$, so that CLT cannot be applied. Figure 5.7(f) shows the throughput.

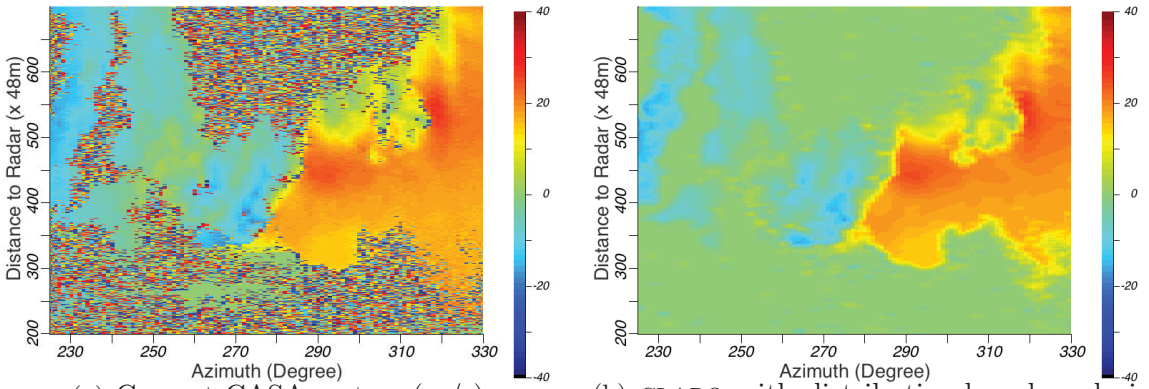
$\mathrm{D}_{\mathrm{sum}}$ is faster than Rand for the high-precision range [0.01, 0.02]. This confirms that to gain high accuracy, Rand needs a very large number of samples and hence degrades the performance quickly. When we do not require high accuracy, Rand can be used for good throughput.

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

5.6.3 Case Study: Tornado Detection

We now demonstrate the effectiveness of capturing uncertainty using distributions in a real-world tornado detection system [58] [4]. We first modified the velocity analysis module in Figure 4.1 to generate velocity distributions in GMMs. In the FFT module, the current system takes a weighted average from the discrete FFT distribution f_F. Instead, we apply a model-based analysis: **Step 1 Strength Filter.** If the radar signal strength is below a threshold, we output zero and skip Step 2. **Step 2 GMM Fitting.** Create a Gaussian distribution from the mean and variance of f_F. Return the distribution for output if it passes the goodness test against f_F, for both the Gaussian shape and high concentration around the mean. If the goodness test fails, fit a mixture of two Gaussians from f_F and remove any component with a large variance as noise. **Step 3 Smoothing.** We average the distributions of high and low frequency stream segments and across neighboring regions. For `avg` over GMMs, we apply the techniques in Section 5.4.2 to compute the result distribution. Since the

[4] We acknowledge that this experiment was done by Boduo Li, whom we worked with in this case study.



(a) Current CASA system (m/s).

(b) CLARO with distribution-based analysis (m/s).

Figure 5.8. Radial velocity maps of a true tornadic region from CASA and CLARO.

Table 5.1. Result of a real tonadic dataset of 947s from 84 scans.

	Analysis Time	Detection Time	False Positives
CASA	182.1 s	4486 s	2137
Step1	180.06 s	640 s	1125
Step3	170.78 s	956 s	1650
Step1+3	176.01 s	441 s	313
Step1+2+3 (claro)	**581.9 s**	**392 s**	**9**

current tornado detection algorithm does not take distributions as input, we feed the mean of each result distribution to the detection algorithm.

This case study used a real tornadic dataset collected in Oklahoma on May 8, 2007, containing raw data of 84 radar scans in 947 seconds. As true velocity changes gradually in space and the tornado detection algorithm expects smooth input, we first examine the spatial smoothness of velocity. The comparison between Figures 5.8(a) and 5.8(b) shows that our techniques yield much smoother velocity maps. Specifically, the Strength Filter removes most colorful dots (i.e., noise) produced from the regions with weak signals (indicating the lack of interesting weather events); the GMM Fitting smoothes data by removing noise in the regions with strong signals; the Smoothing step finally smoothes data across regions, which is especially important for the boundaries between weak-signal regions and strong-signal regions.

We measure the analysis speed, detection speed, and detection result quality. To explore the effect of each step, we show the breakdown of these measurements in Table 5.1. As shown in rows 1-3, both Step 1 and Step 3 can significantly reduce the detection time because data is smoother, but have only a limited effect on false positives. We further combine Steps 1 and 3 as shown in row 4, resulting in further reduction of detection time and false positives. While tornado detection can now be performed at stream speed, the remaining 313 false positives still result in a poor quality of detection results. When we turn on Step 2 for model fitting and model-based analysis, the number of false positives drops to 9 across all 84 scans as shown in row 5. The reason for this remarkable effect is that Step 2 removes noise in the regions with strong radar signals on which the detection algorithm focuses. Although the analysis time increases due to model fitting, given pipeline parallelism, the overall system can still run at stream speed since each of the analysis phase and the detection phase is faster than the radar sensing speed. As such, our model fitting and model analysis approach is shown to provide high-quality detection results while enabling stream-speed data analysis and tornado detection.

5.7 Query Planning under Mixed-type Models

In this section, we examine query planning for complete queries, which involves the arrangement of different operators in a query and considers how to handle errors due to the mix of different operators.

5.7.1 Arranging Operators in a Query Plan

We first discuss the arrangement of relational operators in a query plan using the mixed-type data model. For queries that involve only joins, projections, and aggregates, we have shown that for continuous uncertain attributes modeled by Gaussian mixture models (GMMs), there are exact, closed-form solutions for the result distri-

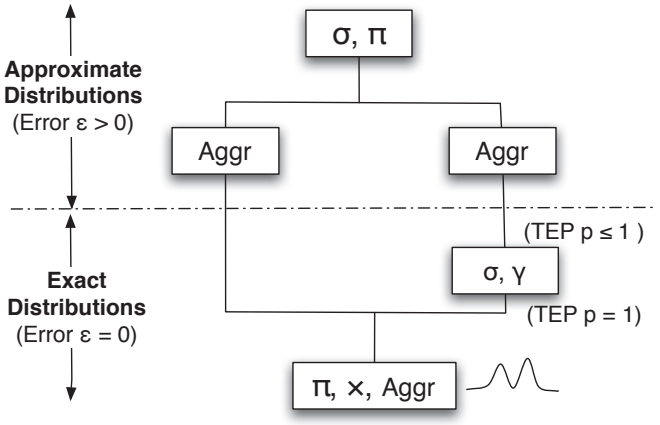


Figure 5.9. Query plan arrangement in the mixed type model.

butions in Sections 5.2 and 5.4. When the above queries are extended with selections, placing selections before joins, projections, and aggregates in a query plan can result in conditioned (or precisely, mixed-type) distributions, hence not in GMMs any more. The implications of this on other relational operations depend on *commutativity*. As in traditional databases, projections and joins commute with selections [73]. Therefore, the GMM-based solutions can still be applied if we postpone selections after the joins and projections in a query plan. However, aggregates do not commute with selections, hence these solutions cannot be applied to aggregates after selections. Similarly, group-bys condition distributions when evaluating the groups, thus precluding GMM-based solutions for subsequent aggregates. Then, we can resort to the approximations proposed in Section 5.5 to compute the distributions of aggregates.

The above discussion suggests the arrangement of relational operators in a query plan, as depicted in Figure 5.9, where the operators contained in the same box can be arranged in any order. In particular, the bottom part of the query plan computes exact distributions, using the exact algorithms and the definition of conditioning operations, i.e., selections, group-bys (denoted as γ). Errors start to occur at the aggregation operator where an approximation algorithm is used, (see Sections 5.4 and 5.5), and will propagate to the subsequent operators.

5.7.2 Query Planning

We now consider query planning that computes approximate answers with bounded errors for complex queries. The CLARO system supports a `Select-From-Where-Group by-Having` block. We can compute multiple aggregates that are independent by invoking the approximation algorithms separately. (Computing correlated aggregates is a harder problem and we discuss some directions for it in the future work section.) More specifically, the cases that we support include: (1) apply selection or group-by on some uncertain attributes and then compute a single aggregate, (2) compute multiple aggregates on independent attributes when tuple existence is certain. In both cases, the aggregates computed can be used in `Having` or returned in `Select`. The examples of the first case are queries Q1 and Q2 (as shown in Section 1.1, where group-bys introduce tuple existence probabilities (TEPs) and the uncertain attributes become correlated through these TEPs. Then, we can compute the marginal distribution for a single aggregate. The second case includes not having `Group by` or having `Group by` on deterministic attributes (e.g., query Q3 below) since this still retains TEPs equal to 1. In this case, the aggregates of the independent attributes are independent and can be computed using our algorithms.

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

Extended Approximation Metric. We first extend the KS metric to a general case when both the TEP and uncertain attributes in a mixed-type tuple are approximate. The extension, adopted from the KS definition for multi-dimensional CDF's [63], considers all complementary distribution functions. We denote an *ordering* of

random variables, $\mathbf{X} = (X_1, X_2, ..., X_n)$, to be a vector $\mathbf{o} = (o_1, o_2, ..., o_n)$, where $o_i = \{\leq, \geq\}$. Given a constant vector, $\mathbf{x} = (x_1, x_2, ..., x_n)$, $\mathbb{P}_{\mathbf{X}}[\langle \mathbf{o}, \mathbf{x} \rangle] = \mathbb{P}[\bigwedge_i (X_i \ o_i \ x_i)]$. $\mathbb{P}_{\mathbf{X}}[\langle \mathbf{o}, \mathbf{x} \rangle]$ can be computed via integration of the joint density function (pdf) of $\mathbf{X}$.

Definition 5.7.1. *Let G=(p, f) and $\tilde{G}$=$(\tilde{p}, \tilde{f})$ be two mixed-type distributions of $\boldsymbol{X}$ and $\tilde{\boldsymbol{X}}$, where each contains n attributes, respectively. The* mixed-type KS*, termed* KSM*, between G and $\tilde{G}$ is defined as:*

$$\text{KSM}(G, \tilde{G}) = \max(|p - \tilde{p}|, \max_{\boldsymbol{o}}(\sup_{\boldsymbol{x}} |p \cdot \mathbb{P}_{\boldsymbol{X}}[\langle \boldsymbol{o}, \boldsymbol{x} \rangle] - \tilde{p} \cdot \mathbb{P}_{\tilde{\boldsymbol{X}}}[\langle \boldsymbol{o}, \boldsymbol{x} \rangle]\,|)).$$

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

Remark 5.7.1. *Let G=(p, F) and $\tilde{G}$=$(\tilde{p}, \tilde{F})$ be two mixed-type distributions where F and $\tilde{F}$ are the cumulative distributions of a single attribute. The* KSM *between G and $\tilde{G}$ becomes:*

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

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

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

Remark 5.7.2. *Let G and $\tilde{G}$ be the multivariate distributions of $\boldsymbol{X}$ and $\tilde{\boldsymbol{X}}$, where each contains n independent attributes. The* KSM *between G and $\tilde{G}$ is:*

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

The following proposition characterizes the KSM of the joint distribution in terms of individual KS's. Its proof is shown in the appendix.

Proposition 5.7.1. *Let G=(p, f) and $\tilde{G}$=$(\tilde{p}, \tilde{f})$ be two mixed-type distributions of attributes $\boldsymbol{X} = (X_1, X_2, ..., X_n)$. If $p = \tilde{p} = 1$, X_i's are independent of each other, and each X_i is bounded with a* KS *error ϵ_i,* $\text{KSM}(G, \tilde{G}) \leq \sum_i \epsilon_i$.

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

Definition 5.7.2. *An approximate query answer set, $\tilde{S}$, is called (ϵ, δ)-approximation of the exact query answer set, S, if $\tilde{S}$ and S contain the same set of tuples and the KSM between any tuple in $\tilde{S}$ and its corresponding tuple in S is at most ϵ with probability $1 - \delta$.*

Our discussion below focuses on (ϵ, δ)-approximation of query answers. A variant, $(\epsilon, \delta, \alpha)$-approximation, further quantifies the approximation when a query gives a threshold, α, for filtering answers with low existence probabilities.

Query Planning: Error Propagation. The goal of query planning is to find a query plan that meets the (ϵ, δ) approximation objective for a given query. To the best of our knowledge, our work is the first to quantify errors for the complex

queries as described before. We first perform a bottom-up analysis of a query plan, focusing on how errors arise and propagate through operators. In our query plans, errors begin at the first aggregation that applies (ϵ, δ)-approximation as proposed in Section 5.5. For post-aggregate operations, the earlier approximation errors now affect the estimates of both the tuple existence probability and distributions of derived aggregate attributes. Below, we focus on selection and projection as post-aggregation operators.

Selection. We quantify the approximation errors propagated through selections, e.g., in the `Having` clause in the next proposition.

Proposition 5.7.2. *Selection on an attribute with (ϵ, δ)- approximation using a range condition ($x \leq u$, $x \geq l$, or $l \leq x \leq u$) is $(2\epsilon, \delta)$-approximation. If the selection uses a union of ranges, the approximation error is the sum of error, $2\epsilon_i$, incurred for each range i.*

The proof of this proposition is presented in the appendix.

When selections are applied for multiple independent aggregates, the above proposition applies for each selection independently. Note that in this case, the TEP would be factorized into these attributes, and its KSM error is bounded by the sum of KSM error of the independent attributes (this is a simple generalization of Proposition 5.7.1 that uses KSM instead of KS).

Projection. Projection does not change the tuple existence probability. That is, if a derived attribute whose existence probability is approximate is projected out, its error is transferred to the existence probability of the result tuple; hence, the KSM of the tuple does not change. This also holds for the case where multiple derived attributes are projected out, since one attribute can be projected out at a time. A special, but trivial, case is when an approximate derived attribute having an exact existence probability is projected out, the KSM error of the result tuple is reduced by the KSM error of that attribute, as indicated by Proposition 5.7.1.

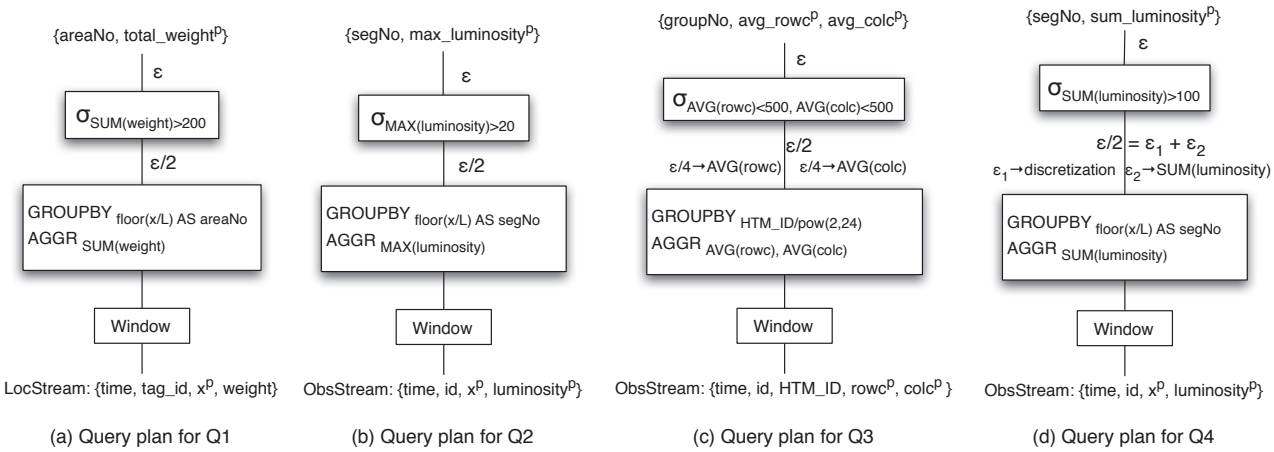


Figure 5.10. Query planning for queries Q1-Q4

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

Computing a single aggregate. Revisit query Q1, from Section 1.1, whose query plan is illustrated in Figure 5.10(a). The query plan first performs the group-by operation, which computes the tuple existence probability of an object in each group, and then computes `sum` of weight for each group using the SUM approximation algorithm, with error ϵ_1. After that, the selection, $\texttt{sum}(weight) > 200$, is applied to each group. Proposition 5.7.2 bounds the error of the selection by $2\epsilon_1$. Therefore, given the target error bound, ϵ, the approximate `sum` should have an error bound $\epsilon_1 = \epsilon/2$.

Query planning for Q2 that computes the maximum luminosity per area, except for computing `max`, is similar to that of Q1, as shown in Figure 5.10(b). These two queries are examples of case (1) we support.

Computing multiple independent aggregates. Query Q3 below is a modified query taken from the Sloan digital sky survey (SDSS) example queries. Q3 groups object into HTM buckets, a deterministic attribute, and computes two independent averages of *rowc* and *colc* and returns the groups when these averages are in a certain range. This corresponds to case (2) above.

```
Q3:  Select   HTM_ID/power(2,24), AVG(rowc), AVG(colc)
     From     Galaxy
     Group by HTM_ID/power(2,24)
     Having   AVG(rowc) ≤ 500 and AVG(colc) ≤ 500
```

If the target accuracy requirements is ϵ, we can assign an error bound of $\epsilon/2$ to each average according to Proposition 5.7.1. Then due to the effect of selection, the error bound assigned to each average before selection is set to $\epsilon/4$. Note that error provisioning remains the same if we use other aggregates than `avg`.

Discretization of continuous distributions. Recall that Q2 computes `max` of *luminosity*, a continuous attribute. Due to the partitioning scheme of the MAX algorithm, we do not need to discretize the distribution of the input tuples in advance. Now consider Q4, a slightly different version of Q2, that computes `sum`(*luminosity*). This query is to detect regions with high cumulative luminosity. Due to the effect of selection after `sum`, given a target error bound ϵ, the approximation of `sum` can have an error bound $\epsilon_0 = \epsilon/2$. Since `sum` is computed for continuous random variables, we need to use discretization as discussed in Section 5.5.4.

The error for `sum` is the sum of the discretization error and the error given to the SUM algorithm; therefore, we can assign error bounds ϵ_1 and ϵ_2 for them respectively, where $\epsilon' = \epsilon_1 + \epsilon_2$. Next we need to allocate the error bound ϵ_1 to individual tuples, given the fact that the error accumulates across tuples. If n is the number of tuples in a given group, then each tuple can be uniformly assigned an error bound of ϵ_1/n. The allocation of ϵ_2 to each tuple is performed in the SUM algorithm. (See Section 5.5.4 for the discussion on how to choose the error bounds ϵ_1 and ϵ_2.) We have discussed

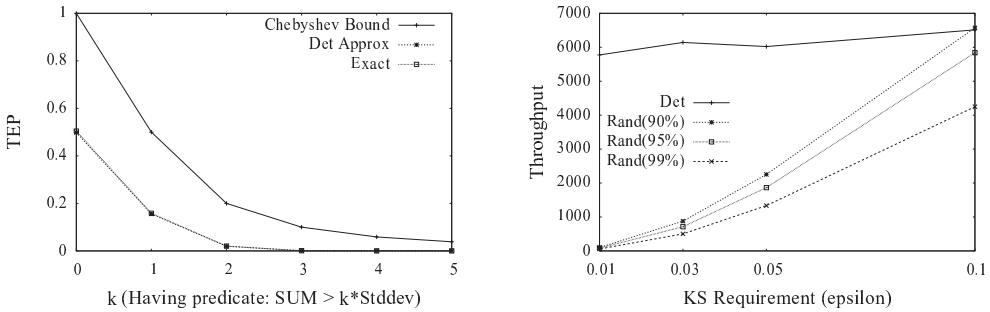
discretization to compute one aggregate, i.e., case (1). The discretization for case (2), when multiple aggregates are computed, is similar, hence omitted.

5.8 Experimental Results for Query Planning

We now evaluate the performance of four queries, Q1 to Q4, whose query plans are shown above.

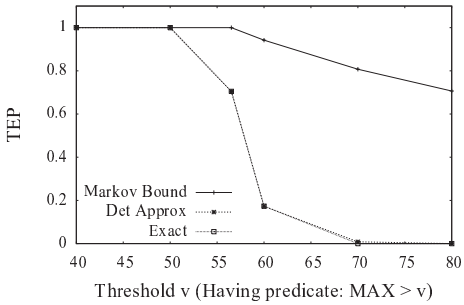
Expt 1: Q1. To run this query, we first obtain a stream of inferred object locations, each of which is modeled by a Gaussian distribution, by running inference [94] over a raw RFID reading stream. This query computes the `sum` of object weights per group and checks if it exceeds 200. Although the weight of an object is deterministic, each object belongs to a group with a probability, resulting in the `sum` of Bernoulli variables, or $\lambda = 2$. This is a common case for aggregation on a deterministic attribute under tuple uncertainty. Given a query accuracy requirement ϵ, the predicate "`sum` > 200" requires assigning an error bound $\epsilon/2$ to the SUM algorithm. The measurements are averaged over 500 time windows, where a time window can contain new location tuples and trigger the computation of group-by aggregation.

We first compare our deterministic algorithm (with $\epsilon = 0.05$) with an alternative method that uses only the moments of the `sum` distribution to estimate the TEP when evaluating the `having` predicate "`sum` $> v$". This method cannot return the distribution of `sum` in the query result, so we restrict the comparison to computing TEP only. Since the mean and variance of `sum` can be computed from the input tuples using the linearity property, we use the Chebyshev's inequality to derive an upper bound of the TEP. Figure 5.11(a) shows the estimates of the TEP as we vary the threshold v in the predicate. As can be seen, using the Chebyshev's inequality can be very inaccurate, thus confirming the need to use the `sum` distribution to compute the TEP.

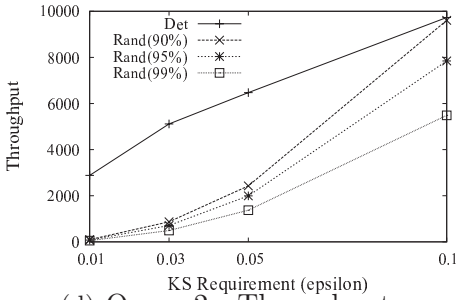


(a) Query 1 - Estimating TEP

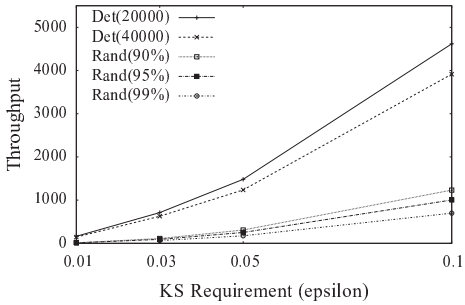
(b) Query 1 - Throughput



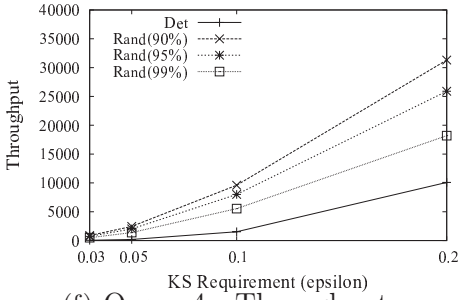
(c) Query 2 - Estimating TEP



(d) Query 2 - Throughput



(e) Query 3 - Throughput



(f) Query 4 - Throughput

Figure 5.11. Experimental results for query planning.

We next compare the performance of the deterministic algorithm, D_{sum}, and the randomized algorithm, Rand, to compute query result distributions. (For this query and the below queries, we verify that the measured accuracy always satisfies the accuracy requirement, i.e., $\leq \epsilon$, and omit the accuracy plots here.) Figure 5.11(b) shows that D_{sum} is faster than Rand. (For clarity, we omit the error bars, or standard deviations, in the plot, but they are less than 18% of the reported means.) This is because smaller error bounds are provisioned to the aggregates to account for the

`having` predicate, which causes Rand to use more samples. Also, since λ is 2 in this query, the cost of $\mathrm{D}_{\mathrm{sum}}$ is smaller compared to Figure 5.7(f).

Expt 2: Q2. For the next three queries, we use a dataset from the Sloan digital sky survey (SDSS) project [92], where the uncertain attributes are modeled by Gaussian distributions. Q2 computes the maximum of luminosity per group and selects groups where $\texttt{max}(\mathit{luminosity}) > 20$. The main difference from Q1 is that the aggregate attribute is continuous. Hence, we set the universe size $\mathbb{U} = 40000$, assuming a high measurement precision of three decimal places. The reported measurements are taken from 100 batches after the warmup phase. (We note that the running time across batches varies little, i.e., the standard deviation is always less than 10% of the mean of the running time.)

We again consider an alternative method that estimates the TEP of result tuples using only the moments of the `max` distribution and summarize the result here. Since the state-of-the-art technique [47] can only compute the mean of `max`, we use the Markov's inequality to derive an upper bound for the TEP. We observe that using this technique can give inaccurate estimates, e.g., the error of the TEP can be as high as 0.6, as shown in Figure 5.11(c).

We now compare our deterministic and randomized algorithms, $\mathrm{D}_{\mathrm{max}}$ and Rand. $\mathrm{D}_{\mathrm{max}}$ outperforms Rand under all chosen accuracy requirements ϵ, as shown in Figure 5.11(d), especially for high ϵ. This confirms that $\mathrm{D}_{\mathrm{max}}$ still performs well for large numbers of values per tuple λ by bounding the number of intervals in the distribution of `max`. Compared to Figure 5.7(a), the throughput decreases for small ϵ, because this is the case when the update time is roughly proportional to $1/\epsilon$ (since λ is large), as shown in Theorem 5.5.2.

Expt 3: Q3. This query computes $\texttt{avg}(\mathit{rowc})$ and $\texttt{avg}(\mathit{colc})$ for objects grouped according to the deterministic attribute HTM_ID. The result TEP of an object in a group after group-by is deterministic (either 0 or 1). Since *rowc* and *colc* follow

Gaussian distributions in the dataset, their `avg` are also Gaussian and can be computed exactly with high throughput of millions tuples per second. As a variant, we compute `max` instead and observe that using D_{max} is 2 to 10 times faster than using Rand for this query (it is similar to Figure 5.11(e), except for provisioning smaller error bounds).

Expt 4: Q4. This query is similar to Q2, but computes `sum`(luminosity). Figure 5.11(f) shows the throughput of two algorithms, D_{sum} and Rand. Since *luminosity* is continuous-valued, we use discretization before computing `sum`. Therefore, D_{sum} has two types of errors: errors from estimation of the input tuple, or discretization errors, and errors from approximating `sum`. Both errors accumulate with the number of tuples, having D_{sum} provision a small error bound per tuple. We observe that D_{sum} has a poorer performance than Rand, which indicates that Rand is useful for computing the `sum` of continuous distributions or distributions with a large number of possible values.

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

5.9 An Experiment Validating the Two-layer Approach

In the previous sections, we have evaluated the proposed techniques for each layer of processing in our system separately. In this section, we perform an experiment to validate the complete system. That is, we consider the end-to-end solution from raw

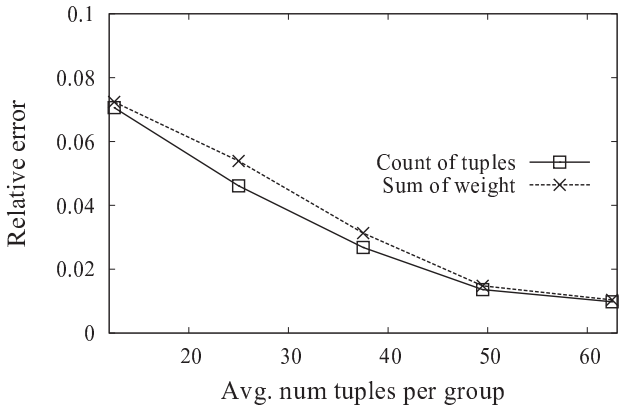


Figure 5.12. Experiment on validating the two-layer architecture

data to final query results, to evaluate the accuracy of our techniques. This is hence considered as a validation of our two-layer approach.

In this experiment, we use a synthetic RFID dataset so that the ground truth of the object locations is known. We use our simulator described in Section 3.3.4 to generate a trace with a fairly large number of objects, i.e., 1000 objects. We first run our techniques in Section 3 to infer the object locations, which are captured with Gaussian distributions (we consider only y locations and do not use moving objects in this experiment).

We then run two group-by aggregation queries for relational processing on the inferred object locations. The first one is to count the number of objects per group. This is a simple query that involves computing the tuple existence probability of each (location) tuple in each group. Since the TEP is a Bernoulli random variable, the count for each group is a distribution. For the purpose of this experiment, we return the mean of this distribution as the estimate count. We measure the relative error, which is defined to be $|est\_count - true\_count|/true\_count$, and report the error averaged over all groups. We vary the group size, which in turn affects the number of tuples per group. Figure 5.12 shows the accuracy result, the error bars, or the standard deviations, of each reported quantity are measured to be 40%-60% of the

mean values. We observe that the relative errors are generally small, i.e., less than or equal to 8% for most of group sizes. The only case that the error is higher, e.g., 17%, is when the groups are very small, containing just a few tuples. Note that when the groups are large, the support of a location distribution is likely to be contained in a group, hence making estimate counts equal to the true ones. When the groups are smaller, each location tuple is estimated to belong to more groups , which reduces the accuracy of our estimated count for each group. Overall, we observe that the accuracy of this query is high, which indicates that the distributions of the inferred locations have means close to the true locations, and have small uncertainty, i.e, small variances.

In the second query, we compute the sum of weights for each group, which involves the approximation algorithm for `sum`, as shown in Section 5.5.4. We set the accuracy requirement for `sum`, $\epsilon = 0.1$. This query is more complex than the above since there are two sources of error involved here: the RFID inference error and the approximation error from computing `sum`. Again, we use the mean of the distribution of `sum` as the estimated sum. Figure 5.12 also shows the relative errors of the estimated sum of weight for different group sizes. We observe that the errors are still low even when this involves two error sources. This example hence validates that our two-layer approach can yield good accuracy for query processing.

CHAPTER 6

SUPPORTING USER-DEFINED FUNCTIONS ON UNCERTAIN DATA

Chapter 5 has addressed relational processing on uncertain data. This chapter is motivated by the observation that real-world applications, such as scientific computing and financial analysis, make intensive use of *user-defined functions (UDFs)*, not expressible in terms of relational operations. Unfortunately, the support for UDFs on uncertain data is largely lacking in the literature. In the following, the problem of computing UDFs on uncertain data is considered in more detail. Then a learning-based approach using Gaussian processes is proposed to address this problem. After that, this chapter includes a performance evaluation of the proposed techniques and a comparison to the standard Monte Carlo sampling approach.

6.1 Overview

In many scientific applications, user-defined functions (UDFs) are used to process and analyze data via complex, domain-specific algorithms, which are not easy to express in relational operations. These functions can vary in complexity from simple to very complicated [18, 45]. In practice, UDFs can be provided in any form of external code, e.g., C programs, and hence treated mainly as *black boxes* in traditional databases. These UDFs are often expensive to compute due to the complexity of processing. Unfortunately, the support for UDFs on uncertain data is largely lacking in today's data management systems. Consequently, in the tornado detection application [58], detection errors cannot be distinguished from true events due to lack

of associated confidence scores. In other applications such as computational astrophysics [89], the burden of characterizing UDF result uncertainty is imposed on the programmers: we observed that the programmers of the Sloan digital sky surveys manually code algorithms to keep track of uncertainty in a number of UDFs. These observations have motivated us to provide system support to automatically capture result uncertainty of UDFs, hence freeing users from the burden of doing so and returning valuable information for interpreting query results appropriately.

To better explain our work, let us consider two concrete examples of UDFs from Sloan digital sky surveys (SDSS) [89]. As discussed in Chapter 1, nightly observations of stars and galaxies in SDSS are inherently noisy as the objects can be too dim to be recognized in a single image. However, repeated observations allow the scientists to model the position, brightness, and color using continuous distributions, e.g., mainly Gaussian distributions. For example, the processed data can be in the form of $(objID, pos^p, redshift^p, ...)$ where pos and $redshift$ are two uncertain attributes. Then, queries can be issued to detect dynamic features or properties of the stars and galaxies. We consider some example UDFs taken from an astrophysics package [6]. Query Q1 below computes the age of each galaxy given its redshift using the UDF $GalAge$. Since $redshift$ is uncertain, the output $GalAge(redshift)$ is also uncertain.

```
Q1:  Select G.objD, GalAge(G.redshift)
     From Galaxy G
```

A more complex example of using UDFs is shown in query Q2, which computes the comoving volume of two galaxies whose distance is in some specific range. This query invokes two UDFs $ComovingVol$ and $Distance$ on uncertain attributes $redshift$ and pos respectively, together with a selection predicate on the output of the UDF $Distance$.

```
Q2:  Select G1.objID, G2.objID, Distance(G1.pos, G2.pos)
            ComovingVol(G1.redshift, G2.redshift, AREA)
```

```
From Galaxy AS G1, Galaxy AS G2
Where Distance(G1.pos, G2.pos) ∈ [l, u]
```

Problem Statement. We aim to provide a general framework to support UDFs on uncertain data, where the functions are given as black boxes. Specifically, given an input tuple modeled by a vector of random variables, $\mathbf{X}$, that is characterized by a joint distribution (either continuous or discrete), and a univariate, black-box UDF f, our objective is to characterize the distribution of $Y = f(\mathbf{X})$. In the example of Q2, after the join between G_1 and G_2, each tuple carries a random vector, $\mathbf{X} = \{G_1.pos, G_1.redshift, G_2.pos, G_2.redshift, \ldots\}$, and two UDFs produce $Y_1 = Distance(G_1.pos, G_2.pos)$ and $Y_2 = ComovingVol(G_1.redshift, G_2.redshift, \text{AREA})$.

Given the nature of the UDFs, exact derivation of result distributions may not be feasible and hence approximation techniques will be explored. A related requirement is that the proposed solution must be able to meet user-specified accuracy requirements. In addition, the proposed solution must be able to perform efficiently in an online fashion, for example, to support online interactive analysis over a large data set or data processing on real-time streams (e.g., to detect tornados or anomalies in sky surveys).

Challenges. Supporting UDFs as stated above poses a number of challenges: (1) UDFs are often computationally expensive. For such UDFs, any processing that incurs repeated function evaluation to compute the output will take a long time to complete. (2) When an input tuple has uncertain values, computing a UDF on them will produce a result with uncertainty, which needs to be characterized by a distribution. Computing the result distribution, even when the function is known, is a non-trivial problem. Existing work in statistical machine learning (surveyed in [10]) uses regression to estimate a function, but mostly focuses on deterministic input. For uncertain input, existing work [41] computes only the mean and variance of the result, instead of characterizing the full distribution, and hence is of limited use if

the output distribution is not Gaussian (which is often the case). Other work [70] computes approximate result distributions without bounding approximation errors, and hence cannot address user accuracy requirements. (3) Further, most of our target applications require using an online algorithm for characterizing result uncertainty of a UDF, where "online" means that the algorithm does not need an offline training phase before processing data. Relevant machine learning techniques such as [41, 70] belong to offline algorithms. In addition, a desirable online algorithm should operate with high performance in order to support online interactive analysis or data stream processing.

Contributions. We present a complete framework for handling user-defined functions on uncertain data. Specifically, the main contributions include:

1. An approximate evaluation framework and baseline approach (in Sections 6.2 and 6.3): We propose a general approximation framework for computing UDFs on uncertain data, including several accuracy metrics and approximation objectives that can be easily used to answer common user questions, i.e., range queries, over the UDF output. We present a baseline approach based on Monte Carlo sampling to compute output distributions of a UDF. We also discuss optimizations for improved performance when the UDF output is filtered with selection predicates.

2. A learning approach using Gaussian processes (in Sections 6.4 and 6.5): We explore a learning-based approach by modeling UDFs using a machine learning technique called *Gaussian processes* (GPs). The key idea is that over time, we can use past function evaluations to build an approximate model of the black-box function, and use the model to avoid most expensive function evaluations in the future. We choose the GP technique due to its abilities to model functions and quantify the approximation in such function modeling.

Given the GP model of a UDF and uncertain input, our contribution lies in computing output distributions with *error bounds*. In particular, we provide an algorithm

that combines the GP model of a UDF and Monte Carlo (MC) sampling to compute output distributions. We perform an in-depth analysis of the algorithm and derive new theoretical results for quantifying the approximation of the output, including bounding the errors of both approximation of the UDF and sampling from input distributions. These error bounds can be used to tune our model to meet accuracy requirements. To the best of our knowledge, our work is the first to quantify output distributions of Gaussian processes.

3. An optimized online algorithm (in Section 6.6): We further propose an *online* algorithm to compute approximate output distributions that satisfy user accuracy requirements. Our algorithm employs a suite of optimizations of the GP learning and inference modules to improve performance and accuracy. Specifically, we propose *local inference* to increase inference speed while maintaining high accuracy, *online tuning* to refine function modeling and adapt to input data, and an *online retraining* strategy to minimize the training overhead. Existing work in machine learning [41, 69, 70, 74] does not provide a sufficient solution to such high-performance online training and inference while meeting user-specified accuracy requirements.

4. Performance evaluation (in Section 6.7): We conduct a thorough evaluation of the proposed techniques using both synthetic functions with controlled properties and real functions and data from the astrophysics application. The results show that our GP techniques can adapt to various function complexities, data characteristics and user accuracy requirements. Compared with Monte Carlo (MC) sampling, the GP approach starts to outperform when function evaluation takes longer than 1ms for functions of low dimensionality, such as up to 2, or when function evaluation takes longer than 100ms for high-dimensional functions such as 10 dimensions. This result applies to real-world expensive functions as we illustrate with the real functions from the astrophysics domain. For the functions we tested, the GP approach can offer up to two orders of magnitude speedup when compared to MC sampling.

6.2 An Evaluation Framework

Since the UDFs are given as *black boxes* and have no explicit formula, computing the output of the UDFs can be done only through function evaluation. Therefore, computing the exact distribution requires function evaluation at all possible values of the input, which is impossible when the input is continuous. Instead, we seek approximation algorithms to compute the output distribution when the input is uncertain. In this section, we present our approximation framework including accuracy metrics and objectives.

We adopt two distance metrics between random variables from the statistics literature [40]: the discrepancy and Kolmogorov–Smirnov (KS) measures. We choose these metrics because they are a natural fit of range queries, hence allowing easy interpretation of the output.

Definition 6.2.1. Discrepancy measure. *The discrepancy measure, $\mathcal{D}$, between two random variables Y and Y' is defined as:*

$\mathcal{D}(Y, Y') = \sup_{a,b:a\leq b} |\Pr[Y \in [a, b]] - \Pr[Y' \in [a, b]]|$.

Definition 6.2.2. KS measure. *The KS measure (or distance) between two random variables Y and Y' is defined as:*

$KS(Y, Y') = \sup_y |\Pr[Y \leq y] - \Pr[Y' \leq y]|$.

The values of these measures are in $[0, 1]$. It is straightforward to show that $KS(Y, Y') \leq \mathcal{D}(Y, Y') \leq 2KS(Y, Y')$. Both measures can be computed directly from the cumulative distribution function (CDF) of random variables. While the KS distance captures the maximum probability difference for any one-sided interval, in the form of $[-\infty, c]$ or $[c, \infty]$, the discrepancy measure captures the maximum probability difference for any two-sided interval $[a, b]$.

In practice, users may be interested only in intervals whose length is larger than a minimum length λ, an application-specific error level that is tolerable for the quantity under computation. This suggests a relaxed variant of the discrepancy measure:

Definition 6.2.3. λ-discrepancy. *Given the minimum interval length λ, the discrepancy measure $\mathcal{D}_\lambda$ between two random variables Y and Y' is:*

$\mathcal{D}_\lambda(Y, Y') = \sup_{a,b:b-a\geq\lambda} |\Pr[Y \in [a,b]] - \Pr[Y' \in [a,b]]|$.

This measure can be interpreted as: for all intervals of length at least λ, the probability of an interval under Y' does not differ from that under Y by more than $\mathcal{D}_\lambda$. These distance metrics can be used to indicate how well one random variable Y' approximates another random variable Y. We next state the our approximation objective, (ϵ, δ)-approximation, using the discrepancy metric; similar definitions hold for the λ-discrepancy and the KS metric.

Definition 6.2.4. (ϵ, δ)-approximation. *Let Y and Y' be two random variables. Then Y' is an (ϵ, δ)-approximation of Y iff with probability $(1-\delta)$, $\mathcal{D}(Y, Y') \leq \epsilon$.*

For query Q1, (ϵ, δ)-approximation requires that with probability $(1-\delta)$, the approximate distribution of `GalAge(G.redshift)` does not differ from the true one more than ϵ in discrepancy. For Q2, there is a selection predicate in the WHERE clause, which truncates the distribution of `Distance(G1.pos, G2.pos)` to the region $[l, u]$, and hence yields a tuple existence probability (TEP). Then, (ϵ, δ)-approximation requires that with probability $(1-\delta)$, (i) the approximate distribution of `Distance(G1.pos, G2.pos)` differs from the true distribution at most ϵ in discrepancy measure, and (ii) the result TEP differs from the true TEP by at most ϵ.

6.3 Monte Carlo Approach

In this section, we present a simple, standard technique to compute the query results based on Monte Carlo simulation (which is also introduced in Section 5.5.2). However, as we will see, the Monte Carlo approach can require evaluating the UDF many times, which is inefficient for slow UDFs. This inefficiency is the motivation for our new approach in Sections 6.4–6.6.

6.3.1 Computing the Output Distribution

In Section 5.5.2, we use Monte Carlo (MC) simulation to compute the output distribution of aggregates on uncertain input. This idea can be directly used to compute $Y = f(\mathbf{X})$. The idea is simple: draw the samples from the input distribution, do function evaluation to get the output samples. The algorithm is:

Algorithm 2 Monte Carlo simulation

1: Draw m samples $\mathbf{x}_1 \dots \mathbf{x}_m \sim p(\mathbf{x})$.
2: Compute the output samples, $y_1 = f(\mathbf{x}_1)$, ..., $y_m = f(\mathbf{x}_m)$.
3: Return the empirical CDF of the output samples, namely Y', $\Pr(Y' \leq y) = \frac{1}{m}\sum_{i\in[m]} \mathbb{1}_{[y_i,\infty)}(y)$, where $\mathbb{1}(\cdot)$ is the indicator function.

We have shown that if $m = \ln(2\delta^{-1})/(2\epsilon^2)$, then the output Y' is an (ϵ, δ)-approximation of Y in terms of KS measure, and $(2\epsilon, \delta)$-approximate in terms of discrepancy measure. Thus the number of samples required to reach the accuracy requirement ϵ is proportional to $1/\epsilon^2$, which is large for small ϵ. For example, if we use the discrepancy measure and set $\epsilon = 0.02$, $\delta = 0.05$, then m required is more than 18000.

6.3.2 Filtering with Selection Predicates

In many applications, users are interested in the event that the output is in certain intervals. This can be expressed with a selection predicate, e.g., $f(\mathbf{X}) \in [a, b]$, in the SQL query as shown in query Q2 above. If the probability $\rho = \Pr[f(\mathbf{X}) \in [a, b]]$ is smaller than a threshold θ specified by the users, it corresponds to an event that the users are not interested in and can be discarded. For high performance, we would like to quickly check whether ρ is small enough to be filtered. If so, we can save the cost from computing the full resulting distribution $f(\mathbf{X})$.

While drawing the samples as in Algorithm 2, we derive a confidence interval for ρ to decide whether to filter. By definition we have $\rho = \int \mathbb{1}(a \leq f(\mathbf{x}) \leq b)p(\mathbf{x})d\mathbf{x}$. Let $h(\mathbf{x}) = \mathbb{1}(a \leq f(\mathbf{x}) \leq b)$ and $\tilde{m}$ be the number of samples drawn so far ($\tilde{m} \leq m$). And let $\{h_i, i = 1 \dots \tilde{m}\}$, be the samples evaluated on $h(\mathbf{x})$. Then, h_i are iid, Bernoulli

samples, and ρ can be estimated by $\tilde{\rho}$, computed from the samples, $\tilde{\rho} = \frac{\sum_{i=1}^{\tilde{m}} h_i}{\tilde{m}}$. The following result, which is a direct application of *Hoeffding's inequality* in statistics, gives a confidence interval for ρ.

Remark 6.3.1. $\rho \in [\tilde{\rho} - \tilde{\epsilon}, \tilde{\rho} + \tilde{\epsilon}]$, *where* $\tilde{\epsilon} = \sqrt{\frac{1}{2\tilde{m}} \ln \frac{2}{1-\delta}}$, *with probability* $(1 - \delta)$.

If the user specifies a threshold θ to filter low-probability events, and $\tilde{\rho} + \tilde{\epsilon} \leq \theta$, then we can filter this output tuple. A modification to handle filtering low-probability events is to use the one-sided version of the Hoeffding's inequality, which states that $\Pr(\rho - \tilde{\rho} > \epsilon) \leq e^{-2m\epsilon^2}$, to get a tighter upper bound for ρ. This approach is appealingly simple, but as before requires $O(1/\epsilon^2)$ calls to the UDF in the worst case, which is inefficient for slow UDFs.

6.4 Emulating UDFs with Gaussian Processes

In the next three sections, we present an approach that aims to be more efficient than MC sampling by requiring many fewer calls to the UDF. The main idea is that every time we call the UDF, we gain information about the function. Once we have called the UDF enough times, we ought to be able to approximate it by *interpolating* between the known values to predict the UDF at unknown values. We call this predictor an *emulator* $\hat{f}$, which can be used in place of the original UDF f, and is much less expensive for many UDFs.

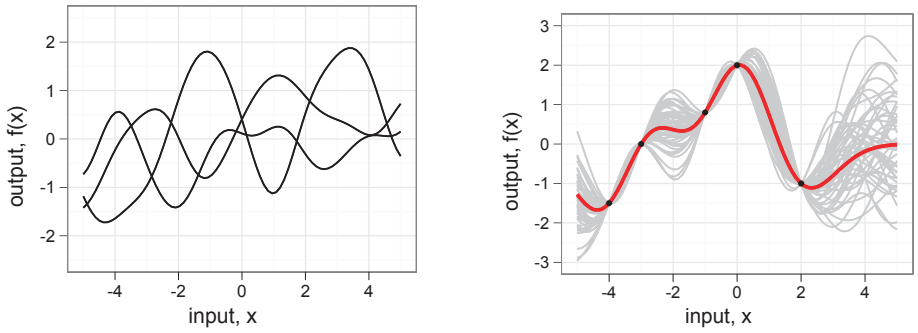
We briefly mention how to build the emulator using a statistical learning approach. The idea is that, if we have a set of function input-output pairs, we can use it as training data to estimate f. In principle, we could build the emulator using any regression procedure from statistics or machine learning, but picking a simple method like linear regression would work poorly on a UDF that did not meet the strong assumptions of that method. Instead, we build the emulator using a learning approach called *Gaussian processes (GPs)*. GPs have two key advantages. First, GPs are

flexible methods that can represent a wide range of functions and do not make strong assumptions about the form of f. Second, GPs produce not only a prediction $\hat{f}(\mathbf{x})$ for any point $\mathbf{x}$ but also a *probabilistic confidence* that provides "error bars" on the prediction. This is vital because we can use this to adapt the training data to meet the user-specified error tolerance. Building an emulator using a GP is a standard technique in the statistics literature (see [70] for an overview).

In this section, we provide background on the basic approach to building emulators. In Section 6.5, we present new results to quantify the uncertainty of outputs of UDFs. After that, we propose an online algorithm to compute UDFs with a suite of optimizations to address accuracy and performance requirements in our setting in Section 6.6.

6.4.1 Intuition for GPs

We give a quick introduction to the use of GPs as emulators, closely following the textbook [76]. A GP is a distribution over functions; whenever we sample from a GP, we get an entire function for f whose output is the real line. Figure 6.1(a) illustrates this in one dimension. It shows three samples from a GP, where each is a function $\mathbb{R} \rightarrow \mathbb{R}$. Specifically, if we pick any input $\mathbf{x}$, then $f(\mathbf{x})$ is a scalar random variable. This lets us get confidence estimates, because once we have a scalar random variable, we can get a confidence interval in the standard way, e.g., $mean \pm 2 * standard\_deviation$. To use this idea for regression, notice that since f is random, we can also define conditional distributions over f, in particular, conditional distribution of f given a set of training points. This new distribution over functions is called the *posterior distribution*, and it is this distribution that lets us predict the function values at new inputs.



(a) Prior (b) Posterior

Figure 6.1. Example of GP regression. (a) prior functions, (b) posterior functions conditioning on training data

6.4.2 Definition of GPs

Just as the multivariate Gaussian is an analytically tractable distribution over vectors, the Gaussian process is an analytically tractable distribution over functions. Just as a multivariate Gaussian is defined by a mean and covariance matrix, a GP is defined by a *mean function* and a *covariance function*. The mean function $m(\mathbf{x})$ gives the average value $\mathbb{E}[f(\mathbf{x})]$ for all inputs $\mathbf{x}$, where the expectation is taken over the random function f. The covariance function $k(\mathbf{x}, \mathbf{x}')$ returns the covariance between the function values at two input points, i.e., $k(\mathbf{x}, \mathbf{x}') = \text{Cov}(f(\mathbf{x}), f(\mathbf{x}'))$.

A GP is a distribution over functions with a special property: if we fix any vector of inputs $(\mathbf{x}_1, \ldots, \mathbf{x}_n)$, the output vector $\mathbf{f} = (f(\mathbf{x}_1), f(\mathbf{x}_2), \ldots, f(\mathbf{x}_n))$ has a multivariate Gaussian distribution. Specifically, $\mathbf{f} \sim \mathcal{N}(\mathbf{m}, K)$, where the vector $\mathbf{m} = (m(\mathbf{x}_1) \ldots m(\mathbf{x}_n))$ contains the mean function evaluated at all the inputs and K is a matrix of covariances $K_{ij} = k(\mathbf{x}_i, \mathbf{x}_j)$ between all the input pairs.

The covariance function has a vital role. Recall that the idea was to approximate f by interpolating between its values at nearby points. To do this we need a way to determine which points are "nearby". This is the role of the covariance function. If two points are far away, then their function values should be only weakly related, i.e., their covariance should be near 0. On the other hand, if two points are nearby,

then their covariance should be large in magnitude. We accomplish this by using a covariance function that depends on the distance between the input points.

In this work, we use standard choices for the mean and covariance functions. We choose the mean function $m(\mathbf{x}) = 0$, which is a standard choice when we have no prior information about the UDF. For the covariance, we use the *squared exponential* function, which in its simplest form is $k(\mathbf{x}, \mathbf{x}') = \sigma_f^2 e^{-\frac{1}{2l^2}\|\mathbf{x}-\mathbf{x}'\|^2}$, where $\|\cdot\|$ is Euclidean distance, and σ_f^2 and l are parameters of the covariance function. The signal variance σ_f^2 primarily determines the variance of the function value at individual points, i.e., if $\mathbf{x} = \mathbf{x}'$. More important is the lengthscale l, which determines how rapidly the covariance decays as $\mathbf{x}$ and $\mathbf{x}'$ move farther apart. One way to interpret the lengthscale is to imagine "typical" random functions sampled from the GP. If l is small, the covariance decays rapidly, so samples from the result GP will have many small bumps; if l is large, then these samples will tend to be smoother.

The key assumption made by GP modeling is that at any point x, the function value $f(x)$ can be accurately predicted using the function values at nearby points. GPs are flexible to model different types of functions by using an appropriate covariance function [76]. For instance, for smooth functions, squared-exponential covariance functions work well; for less smooth functions, Matern covariance functions work well (where smoothness is defined by "mean-squared differentiability"); further, neural network covariance functions are known to work for step functions. The hard cases that make the key assumption likely to be violated are functions that are extremely "spiky" and have many peaks. For example, in an extreme case, the function outputs are boolean or integral-valued, such as the parity function; then the GP approach may need many more training points to capture the change in the function value at the peaks, and the inference can be less accurate.

In this following, we focus on the common squared-exponential functions, which are shown experimentally to work well for the UDFs in our applications (see Section

6.7.4). In general, the user can choose a suitable covariance function based on the well-defined properties of UDFs, and plug it into our framework.

6.4.3 Inference for New Input Points

Now we describe how to use a GP to predict the function outputs at new inputs. Denote the training data by $X^* = \{\mathbf{x}_i^* | i = 1, \ldots, n\}$ for the inputs and $\mathbf{f}^* = \{f_i^* | i = 1, \ldots, n\}$ for the function values. In this section, we assume that we are told a fixed set of m test inputs $X = (\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_m)$ at which we wish to predict the function values. Denote the unknown function values at the test points by $\mathbf{f} = (f_1, f_2, ..., f_m)$. The vector $(\mathbf{f}^*, \mathbf{f})$ is a random vector because each $f_{i:i=1...m}$ is random, and by the definition of a GP, this vector simply has a multivariate Gaussian distribution. This distribution is:

$$\begin{bmatrix} \mathbf{f}^* \\ \mathbf{f} \end{bmatrix} \sim \mathcal{N}\left(\mathbf{0}, \begin{bmatrix} K(X^*, X^*) & K(X^*, X) \\ K(X, X^*) & K(X, X) \end{bmatrix}\right), \tag{6.1}$$

where we have written the covariance as matrix with four blocks. The block $K(X^*, X)$ is an $n \times m$ matrix of the covariances between all training and test points, i.e., $K(X^*, X)_{ij} = k(\mathbf{x}_i^*, \mathbf{x}_j)$. Similar notions are for $K(X^*, X^*)$, $K(X, X)$, and $K(X, X^*)$.

Now that we have a joint distribution, we can predict the unknown test outputs $\mathbf{f}$ by computing the conditional distribution of $\mathbf{f}$ given the training data and test inputs. Applying the standard formula for the conditional of a multivariate Gaussian yields

$$\begin{aligned} &\mathbf{f} \,|\, X, X^*, \mathbf{f}^* \sim \mathcal{N}(\mathbf{m}, \Sigma), \text{where} \\ &\quad \mathbf{m} = K(X, X^*)K(X^*, X^*)^{-1}\mathbf{f}^* \\ &\quad \Sigma = K(X, X) - K(X, X^*)K(X^*, X^*)^{-1}K(X^*, X) \end{aligned} \tag{6.2}$$

To interpret $\mathbf{m}$ intuitively, imagine that $m = 1$, i.e., we wish to predict only one output. Then $K(X, X^*)K(X^*, X^*)^{-1}$ is an n-dimensional vector, and the mean $\mathbf{m}(\mathbf{x})$ is the dot product of this vector with the training values $\mathbf{f}^*$. So $\mathbf{m}(\mathbf{x})$ is simply a

weighted average of the function values at the training points. A similar intuition holds when there is more than one test point, $m > 1$.

Figure 6.1(b) illustrates the resulting GP after conditioning on training data. As observed, the posterior functions pass through these training data points marked by the black dots. From the sampled functions, we can see that the further a point is from the training points, the larger the variance is.

We now consider the complexity of this inference step. Note that once the training data is collected, the inverse covariance matrix $K(X^*, X^*)^{-1}$ can be computed once, with a cost of $O(n^3)$. Then given a test point $\mathbf{x}$ (or X has size 1), inference involves computing $K(X, X^*)$ and multiplying matrices, which has a cost of $O(n^2)$. The space complexity is also $O(n^2)$, for storing these matrices.

6.4.4 Learning the Hyperparameters

Typically, the covariance functions have some free parameters, which are called *hyperparameters*, such as the lengthscale l of the squared-exponential covariance function. The hyperparameters determine how quickly the confidence estimates expand as we consider test points that are farther from the training data. For example, in Figure 6.1(b), as we decrease the lengthscale, we will increase the spread of the function, meaning that we have less confidence in our predictions.

We can learn the hyperparameters using the training data (see Chapter 5, [76]). We adopt maximum likelihood estimation, which is a standard technique for this problem. Let $\boldsymbol{\theta}$ be the vector of hyperparameters. The log likelihood function is $\mathcal{L}(\boldsymbol{\theta}) := \log p(\mathbf{f}^*|X^*, \boldsymbol{\theta}) = \mathcal{N}(X^*; \mathbf{m}, \Sigma)$, where here we use $\mathcal{N}$ to refer to the density of the Gaussian distribution, and $\mathbf{m}$ and Σ are defined in Eq. (6.2). Maximum likelihood estimation solves for the value of $\boldsymbol{\theta}$ that maximizes $\mathcal{L}(\boldsymbol{\theta})$. We perform the maximization using the standard method of gradient descent. The complexity of this is $O(n^3)$, where n is the number of training points, due to the cost of inverting the

f, $\hat{f}$, $\tilde{f}$	true function, mean function of the GP, and a sample function of the GP, respectively.
f_L, f_S	upper and lower envelope functions of $\tilde{f}$ (with high probability)
Y, $\hat{Y}$, $\tilde{Y}$	output corresponding to f, $\hat{f}$, $\tilde{f}$, respectively.
Y_L, Y_S	output corresponding to f_L, f_S, respectively.
$\hat{Y}'$	estimate of $\hat{Y}$ using MC sampling. (Similarly for Y'_L and Y'_S)
$\tilde{\rho}$, $\hat{\rho}$	probability of $\tilde{Y}$ and $\hat{Y}$, in a given interval $[a, b]$.
ρ_U, ρ_L	upper and lower bounds of ρ (with high probability).
$\tilde{\rho}'$, $\hat{\rho}'$, ρ'_U, ρ'_L	MC estimates of $\tilde{\rho}$, $\hat{\rho}$, ρ_U and ρ_L respectively.
n	number of training points.
m	number of MC samples.

Table 6.1. Main notation used in GP techniques.

matrix $K(X^*, X^*)^{-1}$. Gradient descent requires many steps to compute the optimal $\boldsymbol{\theta}$; thus, retraining often has a high cost for a large number of training points. Note that when the training data X^* changes, $\boldsymbol{\theta}$ that maximizes the log likelihood may also change. Therefore, one would need to maximize the log likelihood function to update the hyperparameters. In Section 6.6.3, we will discuss retraining strategies that aim to reduce this computation cost.

6.5 Uncertainty in Query Results

So far in our discussions of GPs, we have assumed that all the input values are known in advance. However, our work aims to compute UDFs on uncertain input. In this section, we describe how to compute an output distribution using a GP emulator when the input is uncertain. We then derive theoretical results to bound the errors of the output using our accuracy metrics.

6.5.1 Computing the Output Distribution

In this section we describe how to approximate the UDF output $Y = f(\mathbf{X})$ given uncertain input $\mathbf{X}$. When we approximate f by the GP emulator $\hat{f}$, we have a new approximate output $\hat{Y} = \hat{f}(\mathbf{X})$, having CDF, $\Pr[\hat{Y} \leq y] = \int \mathbb{1}[\hat{f}(\mathbf{x}) \leq y] p(\mathbf{x}) d\mathbf{x}$. This

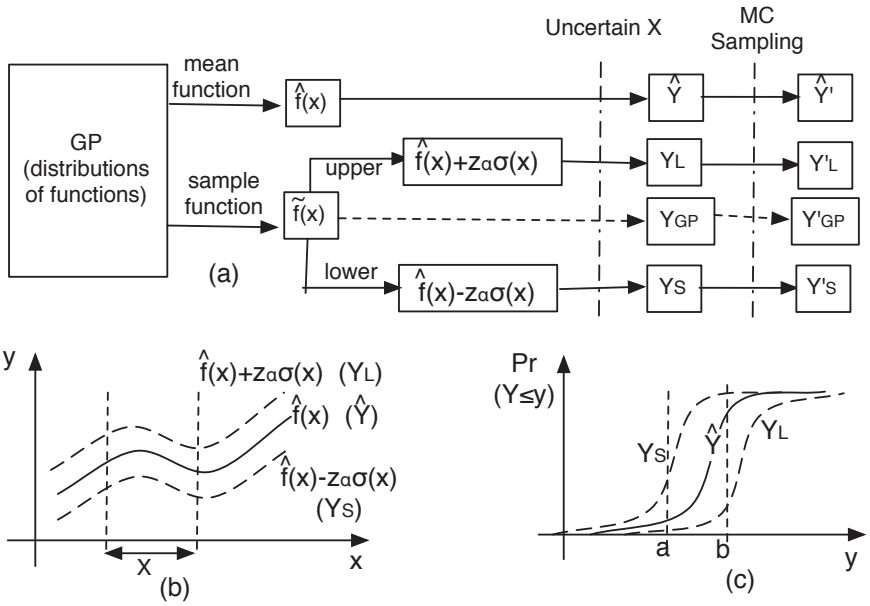


Figure 6.2. GP inference for uncertain input. (a) Computation steps (b) Approximate function with bounding envelope (c) Computing probability for interval [a, b] from CDFs

integral cannot be computed analytically. Instead, a simple, offline algorithm is to use Monte Carlo integration by repeatedly sampling input values from $p(\mathbf{x})$. This is shown in Algorithm 3. This algorithm is very similar to Algorithm 2, except that we call the emulator $\hat{f}$ rather than the UDF f. For computationally expensive UDFs, this is a cheaper operation.

Algorithm 3 Offline algorithm using Gaussian processes

1: Collect n training data points, $\{(\mathbf{x}_i^*, y_i^*), i = 1..n\}$ by evaluating $y_i^* = f(\mathbf{x}^*)$
2: Learning a GP via training using the n training data points, to get $\mathcal{GP} \sim (\hat{f}(\cdot), k(\cdot,\cdot))$.
3: For an input distribution, $\mathbf{X} \sim p(\mathbf{x})$:
4: Draw m samples, $\mathbf{x}_1, ..., \mathbf{x}_m$ from the distribution $p(\mathbf{x})$.
5: Predict function values at the samples via GP inference to get $\{\hat{f}(\mathbf{x}_i), \sigma^2(\mathbf{x}_i), i = 1..m\}$
6: Construct the empirical CDF of $\hat{Y}$ from the samples, namely $\hat{Y}'$, $\Pr(\hat{Y}' \leq y) = \frac{1}{m}\sum_{i\in[m]} \mathbb{1}_{[\hat{f}_i,\infty)}(y)$, and return it.

In addition to returning the CDF of $\hat{Y}'$, we also need to return an confidence of how close $\hat{Y}'$ is to the true answer Y. Ideally we would do this by returning the discrepancy metric, $\mathcal{D}(\hat{Y}', Y)$. But it is difficult to evaluate $\mathcal{D}(\hat{Y}', Y)$ without many calls to the UDF f, which would defeat the purpose of using emulators. So instead we ask a different question, which is easier to analyze. The GP defines a posterior distribution over functions, and we are using the posterior mean as the best emulator. The question we ask is *how different would the query output be if we emulated the UDF using a random function from the GP, rather than the posterior mean?* If this difference is small, this means the GP's posterior distribution over functions is very concentrated. In other words, the uncertainty in the GP modeling is small, and we do not need more training data.

To make this precise, let $\tilde{f}$ be a sample from the GP posterior distribution over functions, and define $\tilde{Y} = \tilde{f}(\mathbf{X})$. That is, $\tilde{Y}$ represents the query ouput if we select the emulator randomly from the GP posterior distribution. The confidence estimate that we will return will be an upper bound on $\mathcal{D}(\hat{Y}', \tilde{Y})$.

An important point to note is that there are two sources of errors here. The first is the *error due to Monte Carlo sampling of the input* and the second is the *error due to the GP modeling*. In the analysis that follows, we bound each of two sources of error individually and then combine them to get a single error bound.

6.5.2 Error Bounds Using Discrepancy Measure

Now we derive a bound on the discrepancy $\mathcal{D}(\hat{Y}', \tilde{Y})$. The main idea is that we will compute a *high probability envelope* over the GP prediction. That is, for any probability α, we can find two functions f_L and f_S such that $f_S \leq \tilde{f} \leq f_L$ with probability at least $1 - \alpha$. Once we have a high probability envelope on $\tilde{f}$, then we also have a high probability envelope of $\tilde{Y}$, and we can use this to bound the discrepancy. See Figure 6.2 for an illustration of this intuition.

Bounding error for one interval. To start, assume that we have already computed a high probability envelope. Since the discrepancy involves a supremum over intervals, we start by presenting upper and lower bounds on $\tilde{\rho} := \Pr[\tilde{Y} \in [a,b] \mid \tilde{f}]$ for a single fixed interval $[a,b]$. Now, $\tilde{\rho}$ is random because $\tilde{f}$ is; for every different function $\tilde{f}$ we get from the GP posterior, we get a different $\tilde{\rho}$.

For any envelope (f_S, f_L), e.g., having the form $\hat{f}(\mathbf{x}) \pm z\sigma(\mathbf{x})$ as shown in Figure 6.2, define $Y_S = f_S(\mathbf{X})$ and $Y_L = f_L(\mathbf{X})$. We bound $\tilde{\rho}$ (with high probability) using Y_S and Y_L. For any two functions g and h, and any random vector $\mathbf{X}$, it is always true that $g \leq h$ implies that $\Pr[g(\mathbf{X}) \leq a] \geq \Pr[h(\mathbf{X}) \leq a]$ for all a. Putting this together with $f_S \leq \tilde{f} \leq f_L$, we have that :

$$\tilde{\rho} = \Pr[\tilde{f}(\mathbf{X}) \leq b] - \Pr[\tilde{f}(\mathbf{X}) \leq a] \leq \Pr[f_S(\mathbf{X}) \leq b] - \Pr[f_L(\mathbf{X}) \leq a]$$

In other words, this gives the upper bound:

$$\tilde{\rho} \leq \rho_U := \Pr[Y_S \leq b] - \Pr[Y_L \leq a] \tag{6.3}$$

Similarly, we can derive the lower bound:

$$\tilde{\rho} \geq \rho_L := \max(0, \Pr[Y_L \leq b] - \Pr[Y_S \leq a]) \tag{6.4}$$

This is summarized in the following result.

Proposition 6.5.1. *Suppose that f_S and f_L are two functions such that $f_S \leq \tilde{f} \leq f_L$ with probability $(1-\alpha)$. Then $\rho_L \leq \tilde{\rho} \leq \rho_U$, with probability $(1-\alpha)$, where ρ_U and ρ_L are as in Equations 6.3 and 6.4.*

Bounding λ-discrepancy. Now that we have an the error bound for one individual intervals, we can use this to bound the λ-discrepancy $\mathcal{D}_\lambda(\tilde{Y}, \hat{Y})$. Using the bounds of $\tilde{\rho}$, we can write this discrepancy as

$$\mathcal{D}_\lambda(\tilde{Y}, \hat{Y}) = \sup_{[a,b]} |\tilde{\rho} - \hat{\rho}| \leq \sup_{[a,b]} \max\{|\rho_L - \hat{\rho}|, |\rho_U - \hat{\rho}|\},$$

Algorithm 4 Compute λ-discrepancy error bound from the output samples

1: Construct the empirical CDFs of the variables $\hat{Y}'$, Y'_S and Y'_L from the output samples.
2: Precompute $\max_{b \geq b_0}(\Pr[\hat{Y}' \leq b] - \Pr[Y'_L \leq b])$ and $\max_{b \geq b_0}(\Pr[Y'_S \leq b] - \Pr[\hat{Y}' \leq b])$ $\forall b_0$, by enumerating the values of $\hat{Y}'$ from the largest to the smallest.
3: Consider values for a, s.t. $[a, a+\lambda]$ lies in the support of $\hat{Y}'$. a is enumerated from small to large using the CDF of $\hat{Y}'$.
4: For a given a:
(a) Get $\Pr[\hat{Y}' \leq a]$, $\Pr[Y'_S \leq a]$, and $\Pr[Y'_L \leq a]$.
(b) Get $\max_{b \geq a+\lambda}(\Pr[Y'_S \leq b] - \Pr[\hat{Y}' \leq b])$.
Find smallest b_1 s.t. $\Pr[Y'_L \leq b_1] \leq \Pr[Y'_S \leq a]$, and then get $\max_{b \geq b_1}(\Pr[\hat{Y}' \leq b] - \Pr[Y'_L \leq b])$. This is done by using the precomputed values in Step 2.
(c) Compute $\max(\rho'_U - \hat{\rho}', \hat{\rho}' - \rho'_L)$. This is the error bound for intervals starting with this a.
5: Increase a, repeat step 4, and store the maximum error.
6: Return the maximum error for all a, which is ϵ_{GP}.

where in the inequality we apply the result from Proposition 6.5.1. This is progress, but we cannot compute ρ_L, ρ_U, or $\hat{\rho}$ exactly because they require integrating over the input $\mathbf{X}$. So we will use Monte Carlo integration once again. We compute Y'_L and Y'_S, as MC estimates of Y_L and Y_S respectively, from the samples in Algorithm 3. We also define (but do not compute) $\tilde{Y}'$, the random variable resulting from MC approximation of $\tilde{Y}$ with the same samples. An identical argument to that of Proposition 6.5.1 shows that

$$\mathcal{D}_\lambda(\tilde{Y}', \hat{Y}') = \sup_{[a,b]} |\tilde{\rho}' - \hat{\rho}'| \leq \sup_{[a,b]} \max\{|\rho'_L - \hat{\rho}'|, |\rho'_U - \hat{\rho}'|\} := \epsilon_{GP},$$

where adding a prime means to use Monte Carlo approximation, e.g., $\hat{\rho}' = \Pr[\hat{Y}' \in [a, b]]$, and so on.

Now we give an algorithm to compute ϵ_{GP}. The easiest idea would be to simply enumerate all possible intervals. Because $\hat{Y}'$ and $\tilde{Y}'$ are empirical cdfs over m samples, there are only $O(m^2)$ possible values for ρ'_U, ρ'_L, and $\hat{\rho}'$. But this is still inefficient for large numbers of samples m, as we observed empirically, and the point of a Monte Carlo simulation is to make m big.

Instead, we present a more efficient algorithm to compute this error bound, show in Algorithm 4. The high-level idea is to (i) pre-compute the maximum differences between the mean function and each envelope function considering decreasing values of b (Step 2), then (ii) enumerate the values of a increasingly and use the pre-computed values to bound $\tilde{\rho}'$ for intervals starting with a (Steps 3-5). Specifically, pre-computing $\max_{b \geq b_0}(\Pr[Y'_S \leq b] - \Pr[\hat{Y}' \leq b])$ and $\max_{b \geq b_0}(\Pr[\hat{Y}' \leq b] - \Pr[Y'_L \leq b])$ involves making a backward pass through the values of the empirical CDFs from the largest to the smallest. Then we make a forward pass from the smallest value to the largest when considering the value of a, the start of an interval. Given a value of a, we can use the precomputed values for all intervals ending at b, where $b - a \geq \lambda$.

Constructing the CDFs involves sorting the output samples, hence costs $O(m \log m)$. The main task of the above algorithm is to take a pass through the m points in the empirical CDF of $\hat{Y}'$. For a given value of a, finding the smallest interval end b_1 s.t. $\Pr[Y'_L \leq b] \leq \Pr[Y'_S \leq a]$ can be done by a binary search, taking $O(\log m)$ time. Thus, the total cost is $O(m \log m)$.

Combining effects of different sources of errors. What we return to the users is the distribution of $\hat{Y}'$, from which $\hat{\rho}'$ can be computed for any interval. As discussed, there are two sources of errors in $\hat{\rho}'$: the GP prediction error and the error from Monte Carlo sampling. The sampling error arises from having $\hat{Y}'$, Y'_L, and Y'_S to approximate $\hat{Y}$, Y_L, and Y_S, respectively. The GP error is from using the mean function in estimating ρ. We can combine these into a single error bound on the discrepancy

$$D_\lambda(\hat{Y}', Y_{GP}) \leq D_\lambda(\hat{Y}', Y'_{GP}) + D_\lambda(\hat{Y}'_{GP}, Y_{GP}).$$

This follows from the triangle inequality that D_λ satisfies because it is a metric. Now, in the last section we showed that $D_\lambda(\hat{Y}', Y'_{GP}) \leq \epsilon_{GP}$. Furthermore, $D_\lambda(\hat{Y}'_{GP}, Y_{GP})$ is just the error due to a standard Monte Carlo approximation, which as discussed in

Section 6.3.1 can be bounded with high probability by a number we call ϵ_{MC}. Also the two sources of error are independent. This yields the main error bound of this work on UDF, which we state as follows.

Theorem 6.5.1. *If MC sampling is $(\epsilon_{MC}, \delta_{MC})$-approximate and GP prediction is $(\epsilon_{GP}, \delta_{GP})$-approximate, then the output has an error bound of $(\epsilon_{MC} + \epsilon_{GP})$ with probability $(1 - \delta_{MC})(1 - \delta_{GP})$.*

Computing simultaneous confidence bands. Now we describe how to choose a high probability envelope, i.e., a pair (f_S, f_L) that contains $\tilde{f}$ with probability $1 - \alpha$. We will use a band of the form $f_S(\mathbf{x}) = \hat{f}(\mathbf{x}) - z_\alpha \sigma(\mathbf{x})$ and $f_L(\mathbf{x}) = \hat{f}(\mathbf{x}) + z_\alpha \sigma(\mathbf{x})$. The problem is to choose z_α. An intuitive choice would be to choose z_α based on the quantiles of the univariate Gaussian, e.g., choose $z_\alpha = 2$ for a 95% confidence band. This would give us a *point-wise* confidence band, i.e., at any point $\mathbf{x}$, we would have $f_S(\mathbf{x}) \leq \tilde{f}(\mathbf{x}) \leq f_L(\mathbf{x})$. But we need something stronger. Rather, we want (f_S, f_L) such that the probability that $f_S(\mathbf{x}) \leq \tilde{f} \leq f_L(\mathbf{x})$ at all inputs $\mathbf{x}$ *simultaneously* is at least $1 - \alpha$. An envelope with this property is called a *simultaneous confidence band*.

We will still use a band of the form $(\tilde{f}(\mathbf{x}) \in [\hat{f}(\mathbf{x}) - z_\alpha \sigma(\mathbf{x}), \hat{f}(\mathbf{x}) + z_\alpha \sigma(\mathbf{x})])$ but we will need to choose a z_α large enough to get an simultaneous confidence band. Say we set z_α to some value z. The confidence band is satisfied if $Z(\mathbf{x}) := |\frac{\tilde{f}(\mathbf{x}) - \hat{f}(\mathbf{x})}{\sigma(\mathbf{x})}| \leq z$ for any $\mathbf{x}$. Therefore, if the probability $\sup_{\mathbf{x} \in \mathbf{X}} Z(\mathbf{x}) \geq z$ is small, the confidence band is unlikely to be violated. We adopt an approximation of this probability due to [1], as follows:

$$\Pr[\sup_{\mathbf{x} \in \mathbf{X}} Z(\mathbf{x}) \geq z] \approx \mathbb{E}[\varphi(A_z(\mathbf{X})], \tag{6.5}$$

where the set $A_z(\mathbf{X}) := \{\mathbf{x} \in \mathbf{X} : Z(\mathbf{x}) \geq z\}$ is the set of all inputs where the confidence band is violated, and $\varphi(A)$ is the Euler characteristic of the set A. Also, [1] provides a numerical method to approximate Eq. (6.5) that works well for small α, i.e., when we require a high probability that the confidence band is correct, which

is precisely the case of interest. The main computation is to evaluate the following quantity, called the *Lipschitz-Killing curvature*:

$$\mathcal{L}(\mathbf{X}) = \int_{\mathbf{X}} [\det(\Lambda(\mathbf{x}))]^{1/2} d\mathbf{x}, \tag{6.6}$$

where $\Lambda(\mathbf{x})$ is a matrix whose elements are

$$\lambda_{ij}(\mathbf{x}) = \mathbb{E}\left\{ \frac{\partial Z(\mathbf{x})}{\partial \mathbf{x}_i} \frac{\partial Z(\mathbf{x})}{\partial \mathbf{x}_j} \right\} = \frac{\partial^2 k(\mathbf{x}', \mathbf{x}'')}{\partial \mathbf{x}'_i \partial \mathbf{x}''_j}|_{\mathbf{x}'=\mathbf{x}''=\mathbf{x}},$$

and i and j index particular dimensions of the input tuples. We compute the integral in Eq. (6.6) using Monte Carlo integration, The main computational expense is that the approximation requires computing second derivatives of the covariance function, but we have still found it to be feasible in practice. Once we computed the approximation to Eq. (6.5), we compute the confidence band by setting z_α to be the solution of the equation $\Pr[\sup_{\mathbf{x}\in\mathbf{X}} Z(\mathbf{x}) \geq z_\alpha] \approx \mathbb{E}[\varphi(A_z(\mathbf{X})] = \alpha$.

6.5.3 Error Bounds for KS Measure

The above analysis can be applied if the KS distance is used as the accuracy metric in a similar way. The main result is as follows.

Proposition 6.5.2. *Consider the mean function $\hat{f}(\mathbf{x})$ and the envelope $\hat{f}(\mathbf{x}) \pm z\sigma(\mathbf{x})$. Let $\tilde{f}(\mathbf{x})$ be a function in the envelope. Given uncertain input $\mathbf{X}$, let $\hat{Y} = \hat{f}(\mathbf{X})$ and $\tilde{Y} = \tilde{f}(\mathbf{X})$. Then $KS(\tilde{Y}, \hat{Y})$ is largest when $\tilde{f}(\mathbf{x})$ is at either the boundary of the envelope.*

Proof. Recall that $KS(\tilde{Y}, \hat{Y}) = \sup_y |\Pr[\tilde{Y} \leq y] - \Pr[\hat{Y} \leq y]$. Let y_m correspond to the supremum in the formula of KS. Wlog, let $KS = \int (\mathbb{1}[\hat{f}(\mathbf{x}) \leq y_m] - \mathbb{1}[\tilde{f}(\mathbf{x}) \leq y_m])p(\mathbf{x})d\mathbf{x} > 0$. That is, for some $\mathbf{x}$, $\hat{f}(\mathbf{x}) \leq y_m < \tilde{f}(\mathbf{x})$. Now suppose there exists some $\mathbf{x}'$ s.t. $\tilde{f}(\mathbf{x}') < \hat{f}(\mathbf{x}')$, the KS distance would increase if $\hat{f}(\mathbf{x}') \leq \tilde{f}(\mathbf{x}')$. This

means, KS becomes larger when $\tilde{f}(\mathbf{x}) \geq \hat{f}(\mathbf{x})$ for all $\mathbf{x}$; or, $\tilde{f}(\mathbf{x})$ lies above $\hat{f}(\mathbf{x})$ for all $\mathbf{x}$. Also, it is intuitive to see that among the functions that lie above $\hat{f}(\mathbf{x})$, $\hat{f}(\mathbf{x}) + z\sigma(\mathbf{x})$ yields the largest KS error, since it maximizes $\mathbb{1}[\hat{f}(\mathbf{x}) \leq y] - \mathbb{1}[\tilde{f}(\mathbf{x}) \leq y], \forall y$. (Similarly, we can show that if $KS = \int(\mathbb{1}[\tilde{f}(\mathbf{x}) \leq y_m] - \mathbb{1}[\hat{f}(\mathbf{x}) \leq y_m])p(\mathbf{x})d\mathbf{x} > 0$, KS is maximized if $\tilde{f}(\mathbf{x})$ lies below $\hat{f}(\mathbf{x})$ for all $\mathbf{x}$.) □

As a result, let Y_S and Y_L be the output computed using the upper and lower boundaries $\hat{f}(\mathbf{x}) \pm z\sigma(\mathbf{x})$ respectively. Then, the KS error bound is $\max(KS(\hat{Y}, Y_S), KS(\hat{Y}, Y_L))$

If we use Monte Carlo sampling, we can obtain the empirical variables $\hat{Y}'$, Y_S', and Y_L' as before. We then can compute $\max(KS(\hat{Y}', Y_S'), KS(\hat{Y}', Y_L'))$, denoted ϵ_{GP}.

We also analyze the combining effects of the two sources of errors, MC sampling and GP modeling, as for the discrepancy measure. We obtain a similar result: the total error bound is the sum of the two error bounds, ϵ_{MC} and ϵ_{GP}. The proof of this result is outlined as follows.

$$
\begin{aligned}
KS(\hat{Y}', Y) &= \sup_y |\Pr[\hat{Y}' \leq y] - \Pr[Y \leq y]| \\
&\leq \sup_y(\max(|\Pr[\hat{Y}' \leq y] - \Pr[Y_L \leq y]|, |\Pr[\hat{Y}' \leq y] - \Pr[Y_S \leq y]|)) \\
&\leq \sup_y(\max(|\Pr[\hat{Y}' \leq y] - \Pr[Y_L' \leq y]| + \epsilon_{KS}, |\Pr[\hat{Y}' \leq y] - \Pr[Y_S' \leq y]| + \epsilon_{KS}) \\
&= \epsilon_{KS} + \sup_y(\max(|\Pr[\hat{Y}' \leq y] - \Pr[Y_L' \leq y]|, |\Pr[\hat{Y}' \leq y] - \Pr[Y_S' \leq y]|)) \\
&= \epsilon_{KS} + \epsilon_{GP}
\end{aligned}
$$

6.6 An Optimized Online Algorithm

In Section 6.5.1, we present a basic algorithm to compute output distributions using Gaussian processes to model our UDFs (see Algorithm 3). However, this algorithm does not satisfy our design constraints as follows. This is an offline algorithm since the training data is fixed and learning is performed once, before inference. Given a user-specified accuracy requirement, it is hard to know how many training points are needed beforehand. If we use a large number of training points, the accuracy is

higher, but the performance suffers. Besides, using standard inference has an inference cost of $O(n^2)$ per sample point, which is high for large numbers of training points n. In this section, we aim to provide an online algorithm that is robust to UDFs and input distributions in terms of meeting accuracy requirements. We further optimize it for high performance.

6.6.1 Local Inference

We first propose a technique to reduce the cost of inference while maintaining good accuracy. The key observation is that the covariance between two points $\mathbf{x}_i$ and $\mathbf{x}_j$ is small when the distance between them is large. For example, for the squared exponential covariance function, the covariance decreases exponentially in the squared distance, $k(\mathbf{x}_i, \mathbf{x}_j) = \sigma_f^2 \exp\{-\frac{||\mathbf{x}_i - \mathbf{x}_j||^2}{l^2}\}$. In other words, the far training points have only small weights in the weighted average, hence can be omitted. This observation suggests a technique which we call *local inference*. (We refer to the standard inference technique described before as *global inference*.) The steps of local inference are described as follows.

Algorithm 5 Local inference

Input: Input distribution $p(\mathbf{x})$. Training data: $\{(\mathbf{x}_i^*, y_i^*), i = 1 \dots n\}$, stored in an R-tree.

1: Draw m samples from the input distribution $p(\mathbf{x})$ and construct a bounding box for the samples.
2: Retrieve a set of training points, called X_L^*, from the R-tree that have distance to this bounding box less than some maximum distance. (This controls by the local inference threshold Γ discussed below.)
3: Run inference using X_L^* to get the function values at the samples. Return the CDF constructed from the inferred values.

Figure 6.3 illustrates the execution of local inference, where a subset of relevant training points is chosen for inference based on the input distribution. The darker rectangle is the bounding box of the input samples, and the lighter rectangle is the bounding box of the training points selected for local inference.

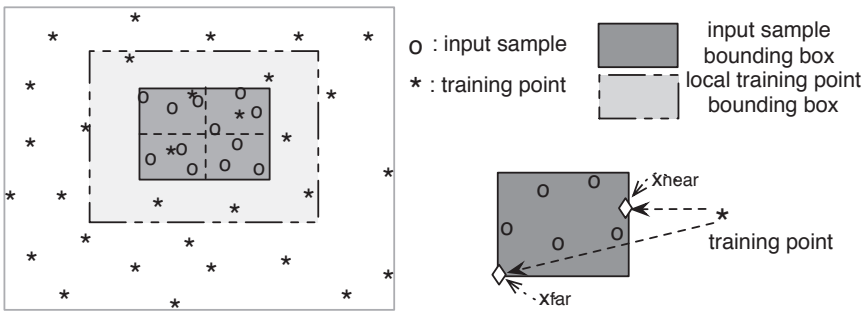


Figure 6.3. Choosing a subset of training points for local inference

Choosing the training points for local inference given a threshold. The threshold Γ is chosen so that the approximation error in $\hat{f}(\mathbf{x}_j)$, for all samples $\mathbf{x}_j$, is small. That is, $\hat{f}(\mathbf{x}_j)$ when computed using either global inference or local inference does not differ much. We can update the vector $K(X^*, X^*)^{-1}\mathbf{y}^*$, called $\boldsymbol{\alpha}$, once the training data changes, and store it for later inference. Then, computing $\hat{f}(\mathbf{x}_j) = K(\mathbf{x}_j, X^*)K(X^*, X^*)^{-1}\mathbf{y}^* = K(\mathbf{x}_j, X^*)\boldsymbol{\alpha}$ involves a vector dot product. Note that the cost of computing the mean is $O(n)$; the high cost of inference comes from computing the variance $\sigma^2(\mathbf{x}_j)$, which is $O(n^2)$ (see Section 6.4.3 for more details).

If we use a subset of training points, we approximate $\hat{f}(\mathbf{x}_j)$ with $\hat{f}_L(\mathbf{x}_j) = K(\mathbf{x}_j, X_L^*)\boldsymbol{\alpha}_L$. ($\boldsymbol{\alpha}_L$ is the same as $\boldsymbol{\alpha}$ except that the entries in $\boldsymbol{\alpha}$ that do not correspond to a selected training point are set to 0). Then the approximate error γ_j is equal to:

$$\begin{aligned} \gamma_j &\approx K(\mathbf{x}_j, X^*)\boldsymbol{\alpha} - K(\mathbf{x}_j, X_L^*)\boldsymbol{\alpha}_L \\ &= K(\mathbf{x}_j, X_{\bar{L}}^*)\boldsymbol{\alpha}_{\bar{L}} = \sum_{l \in \bar{L}} k(\mathbf{x}_j, \mathbf{x}_l^*)\alpha_l, \end{aligned}$$

where $X_{\bar{L}}^*$ is the set of training points excluded from local inference. Ultimately, we want to compute $\gamma = \max_j |\gamma_j|$, which is the maximum error over all the samples. The cost of computing γ by considering every j is $O(mn)$, as $j = 1...m$, which is high for large m.

We next present a more efficient way to compute an upper bound for γ. We use a bounding box for all the samples $\mathbf{x}_j$ as constructed during local inference. For any training point with index l, let $\mathbf{x}_{near}$ be the closest point from the bounding box to $\mathbf{x}_l^*$ and $\mathbf{x}_{far}$ be the furthest point from the bounding box to $\mathbf{x}_l^*$ (see Figure 6.3 for an example of these points). Then for any sample j we have:

$$k(\mathbf{x}_{far}, \mathbf{x}_l^*) \leq k(\mathbf{x}_j, \mathbf{x}_l^*) \leq k(\mathbf{x}_{near}, \mathbf{x}_l^*)$$

Therefore, if $\alpha_l \geq 0$,

$$k(\mathbf{x}_{far}, \mathbf{x}_l^*)\alpha_l \leq k(\mathbf{x}_j, \mathbf{x}_l^*)\alpha_l \leq k(\mathbf{x}_{near}, \mathbf{x}_l^*)\alpha_l$$

If $\alpha_l < 0$, we have a similar inequality with opposite signs.

Using these inequalities, we can obtain an upper bound γ_{upper} and lower bound γ_{lower} for $\gamma_j, \forall j$. Then,

$$\gamma = \max_j |\gamma_j| \leq \max(|\gamma_{upper}|, |\gamma_{lower}|)$$

Computing this takes time proportional to the number of excluded training points, which is $O(n)$. For each of these points, we need to consider the sample bounding box, which incurs a constant cost when the dimension of the function is fixed. After computing γ, we compare it with the threshold Γ. If $\gamma > \Gamma$, we expand the bounding box for selected training points and recompute γ until we have $\gamma \leq \Gamma$. Note that Γ should be set to be small compared with the domain of Y, i.e., the error incurred for every test point is small. In Section 6.7, we show how to set Γ to obtain good performance.

Complexity for local inference. If l is the number of selected training points, the cost of inference is $O(l^3 + ml^2 + n)$. $O(l^3)$ is to compute the inverse matrix $K(X_L^*, X_L^*)^{-1}$ used in computing the predicted variance; $O(ml^2)$ is to compute the output variance; and $O(n)$ is to compute γ when deciding the set of local training points. Among the components, $O(ml^2)$ is usually the dominant cost (especially

when the accuracy requirement is high). This is an improvement compared to global inference, which has a cost of $O(mn^2)$, because usually l is smaller than n.

Optimizations. We discuss some insights that help improve performance of local inference. To make the bound γ tighter, we can divide the sample bounding box into smaller non-overlapping boxes as shown in Figure 6.3. Then for each box, we compute the corresponding γ. The cost to compute γ is higher by a constant factor, but the bounds are tighter since $\mathbf{x}_{near}$ and $\mathbf{x}_{far}$ are closer to any $\mathbf{x}_j^*$, so overall fewer training points may be needed to satisfy the threshold Γ.

Comparing to existing work. We discuss two lines of existing work related to our idea of GP local inference. [69] suggests predividing the function domain into fixed local regions corresponding to local models. It then runs inference using the local models and combines the results by weighting them. This is different from our technique since all training points are used, hence can be inefficient for large training datasets. Besides, this work does not include an error bound analysis. The work [74] has a similar idea to ours by using sparse covariance matrices, which zeroes out low covariances. However, it does not address approximation errors either.

6.6.2 Online Tuning

In our work, we seek an online algorithm for GPs: we start with no training points and collects them over time so that the function model gets more accurate. We can examine each input distribution on-the-fly to see whether more training points are needed given the accuracy requirement. This contrasts with the offline approach where the training data must be obtained before inference.

To develop an online algorithm, we need to make two decisions. The first decision is *how many* training points to add. This is the job of the error bounds from Section 6.5, that is, we add training points until the upper bound on the error is less than the user's tolerance. The second decision is *where* the training points should be,

specifically, what input location $\mathbf{x}_{n+1}$ to use for the next training point. A standard idea is to add new training points where the function evaluation is highly uncertain, i.e., $\sigma(\mathbf{x})$ is large. We adopt a simple heuristic for this. Recall that we use a number of Monte Carlo samples $\mathbf{x}_j$ to evaluate the error bounds. We simply cache these samples throughout the algorithm, and when we need more training points, we choose the sample $\mathbf{x}_j$ that has the largest prediction variance $\sigma^2(\mathbf{x}_j)$, compute its true function value $f(\mathbf{x}_j)$, and add it to the training data set. After that, we run inference, compute the error bound again, and repeat until the error bound is small enough. We have experimentally found that this simple heuristic works well.

A complication is that when we add a new training point, the inverse covariance matrix gets bigger $K(X^*, X^*)^{-1}$, so it needs to be recomputed. Recomputing it from scratch would be expensive, i.e., $O(n^3)$. Fortunately, we can update it incrementally using the standard formula for inverting a block matrix as follows.

$$K_{n+1} = \begin{bmatrix} K_n & \mathbf{m} \\ \mathbf{m}^T & k(\mathbf{x}', \mathbf{x}') \end{bmatrix}, \; K_{n+1}^{-1} = \begin{bmatrix} K_n^{-1} + \frac{1}{\mu}\mathbf{g}\mathbf{g}^T & \mathbf{g} \\ \mathbf{g}^T & \mu \end{bmatrix},$$

where $\mathbf{m} = [k(\mathbf{x}_1, \mathbf{x}')...k(\mathbf{x}_n, \mathbf{x}')]$, $\mathbf{g} = -\mu K_n^{-1}\mathbf{m}$, and $\mu = (k(\mathbf{x}', \mathbf{x}') - \mathbf{m}^T K_n^{-1}\mathbf{m})^{-1}$

6.6.3 Online Retraining

In our work, the training data is obtained on the fly. Since different inputs correspond to different regions of the function, we may need to tune the GP model to best fit the up-to-date training data, i.e., to *retrain*. A key question is when we should perform retraining (the technique is mentioned in Section 6.4.4). It is preferable that retraining is done infrequently due to its high cost of $O(n^3)$ in the number of training points n and multiple iterations required. The problem of retraining is less commonly addressed in existing work for GPs (whereas it is better addressed for other models in machine learning).

Since retraining involves maximizing the likelihood function $\mathcal{L}(\boldsymbol{\theta})$, we will make this decision by examining the likelihood function. Recall also that the numerical optimizer, e.g., gradient descent, requires multiple iterations to find the optimum. The heuristic is: Run training only if the optimizer is able to make a big step during its very first iteration. Given the current hyperparameters $\boldsymbol{\theta}$, we run the optimizer for one step to get a new setting $\boldsymbol{\theta'}$, and continue with training only if $||\boldsymbol{\theta'} - \boldsymbol{\theta}||$ is larger than a pre-set threshold Δ_θ.

In practice, we have found that gradient descent does not work well with this heuristic, because it does not move far enough during each iteration. Instead, we use a more sophisticated heuristic based on a numerical optimizer, called Newton's method, which uses both the first and the second derivatives of $\mathcal{L}(\boldsymbol{\theta})$. Some algebra shows that second derivatives of $\mathcal{L}(\boldsymbol{\theta})$ are

$$\mathcal{L}''(\theta_j) = \frac{1}{2}\,\mathrm{tr}[(\frac{\partial K^{-1}}{\partial \theta_j}\mathbf{y}^*\mathbf{y}^{*T}K^{-1} + K^{-1}\mathbf{y}^*\mathbf{y}^{*T}\frac{\partial K^{-1}}{\partial \theta_j} - \frac{\partial K^{-1}}{\partial \theta_j})\frac{\partial K}{\partial \theta_j} + (K^{-1}\mathbf{y}^*\mathbf{y}^{*T}K^{-1} - K^{-1})\frac{\partial^2 K}{\partial \theta_j^2}]$$

where $\mathrm{tr}[\cdot]$ denotes the trace of a matrix. Using Newton's method, we have $\theta'_j - \theta_j \approx -\frac{\mathcal{L}'(\theta_j)}{\mathcal{L}''(\theta_j)}$. Then we can compute $\Delta_{\boldsymbol{\theta}} = ||\boldsymbol{\theta'} - \boldsymbol{\theta}||$. If $\Delta_{\boldsymbol{\theta}}$ is smaller than some threshold, then retraining is not triggered. Otherwise, we will perform retraining. As we add more training points, we can update $\partial K/\partial \theta_j$ and $\partial^2 K/\partial \theta_j^2$ incrementally, similarly to K^{-1} in Section 6.6.2.

6.6.4 A Complete Online Algorithm

We now put together the techniques discussed above to form a complete algorithm to compute UDFs on uncertain data using Gaussian processes. The main idea is, starting with no training data, given an input distribution, we use online tuning in Section 6.6.2 to obtain more training data, and run inference to compute the output distribution. Local inference in Section 6.6.1 is used for improved performance. When

Algorithm 6 OLGAPRO: Compute output distribution using Gaussian process with optimizations

Input: Input tuple $\mathbf{X} \sim p(\mathbf{x})$. Training data: $\mathcal{T} = \{(\mathbf{x}_i^*, y_i^*), i = 1..n\}$; hyperparameters of the GP: $\boldsymbol{\theta}$. Accuracy requirement for the discrepancy measure: (ϵ, δ).

1: Draw m samples for $\mathbf{X}$, $\{\mathbf{x}_j, j = 1..m\}$, where m depends on the sampling error bound $\epsilon_{MC} < \epsilon$.
2: Compute the bounding box for these samples. Retrieve a subset of training points for local inference given the threshold Γ (see Section 6.6.1). Denote this set of training point $\mathcal{T}_\Gamma$.
3: **repeat**
4: Run local inference using $\mathcal{T}_\Gamma$ to get the output samples $\{(\hat{f}(\mathbf{x}_j), \sigma^2(\mathbf{x}_j)), j = 1..m\}$.
5: Compute the discrepancy error bound $\mathcal{D}_{upper}$ using these samples (see Section 6.5.2).
6: If $\mathcal{D}_{upper} > \epsilon_{GP}$, add a new training point at the sample with largest variance, i.e., $(\mathbf{x}_{n+1}^*, f(\mathbf{x}_{n+1}^*))$ (see Section 6.6.2), and insert $(\mathbf{x}_{n+1}^*, f(\mathbf{x}_{n+1}^*))$ into the training data index. Set $n = n + 1$.
7: **until** $\mathcal{D}_{upper} < \epsilon_{GP}$
8: **if** one or more training points are added **then**
9: Compute the log likelihood $\mathcal{L}(\boldsymbol{\theta}) = \log p(\mathbf{y}^*|X^*, \boldsymbol{\theta})$ and its first and second derivatives, and estimate δ_θ (see Section 6.6.3).
10: **if** $\delta_\theta \leq \Delta_\theta$ **then**
11: Retrain to get the new hyperparameters $\boldsymbol{\theta'}$. Set $\boldsymbol{\theta} = \boldsymbol{\theta'}$.
12: Rerun inference.
13: **end if**
14: **end if**
15: Return the distribution of Y, computed from samples $\{\hat{f}(\mathbf{x}_j)\}$.

some training points are added, we use our retraining strategy to decide whether to relearn the GP model by updating its hyperparameters.

Our algorithm, which we name OLGAPRO, standing for *ONline GAussian PROcess*, is shown as Algorithm 6. The objective is to compute the output distribution that meets the user-specified accuracy requirement under the assumption of GP modeling. Basically, the main steps of the algorithm involves: (a) Compute the output distribution by sampling the input and inferring with the Gaussian process (Steps 1-4). (b) Compute the error bound (Steps 5-7). If the error bound is larger than the specified accuracy requirement, use online tuning to add a new training point.

Repeat this until the error bound is acceptable. (c) If one or more training points have been added, use the retraining strategy to decide whether retraining is needed and perform it if so (Steps 8-12).

Parameter setting. We further consider the parameters used in the algorithm. The choice of Γ for local inference in step 2 is discussed in Section 6.6.1. The allocation of two sources of error, ϵ_{MC} and ϵ_{GP}, is according to Theorem 6.5.1, $\epsilon = \epsilon_{MC} + \epsilon_{GP}$. Then our algorithm automatically chooses the number of samples m to meet the accuracy requirement ϵ_{MC} (see Section 6.3 for the formula). For retraining, setting the threshold Δ_θ, mentioned in Section 6.6.3, smaller triggers retraining more often but potentially makes the model more accurate, while setting it high can give inaccurate results. In Section 6.7, we experimentally show how to set these parameters efficiently.

Complexity. The complexity of local inference is $O(l^3+ml^2+n)$ as shown in Section 6.6.1. Computing the error bound takes $O(m \log m)$ (see Section 6.5.2). And retraining takes $O(n^3)$. The number of samples m is $O(1/\epsilon_{MC}^2)$ and the number of training points n depends on ϵ_{GP} and the UDF itself. The unit cost is basic math operations, in contrast to complex function evaluations as in Monte Carlo simulation. This is because when the system converges, i.e., having enough training points, we seldomly need to add more training points, so the cost of function evaluation can be avoided. Also, at convergence, since few training points are added, we can avoid the high cost of retraining, i.e., the required computation involves inference and computing error bounds.

6.6.5 Hybrid Solution

We next discuss a hybrid solution that combines our two approaches: direct MC sampling, and GP modeling and inference. The need for a hybrid solution rises since functions can vary on their complexity and evaluation time. Therefore, when given a black-box function, we explore these properties and choose the better solution.

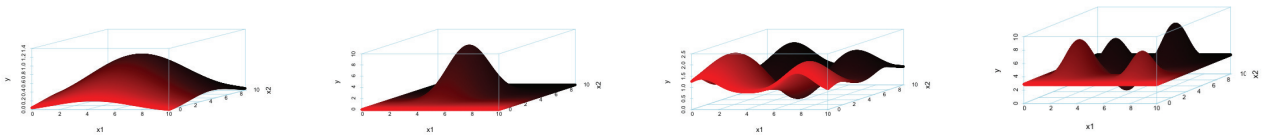
We now outline the steps of our hybrid solution. (1) When given a black box UDF, we draw some samples in the input domain, evaluate their function values and measure the time taken for function evaluation. (2) We use the samples above as training points and learn a GP to model the function. (3) Run the OLGAPRO algorithm to compute the output distribution and measure the running time. (4) Repeat the above steps for later inputs. This step helps determine when the GP converges (i.e., how many training points are required) and the time taken to run the GP. (5) We then compare the running time of the two approaches, choose the more efficient one, and use it for future inputs. In Section 6.7, we conduct experiments to find out when each approach can be applied for different UDFs.

6.6.6 Online Filtering

Similarly to filtering using Monte Carlo simulation in the presence of a selection predicate (as discussed in Section 6.3.2), we now consider how to do online filtering when sampling with a Gaussian process. Again, we consider selection with the predicate $a \leq \tilde{f}(\mathbf{x}) \leq b$. Let $(\hat{f}(\mathbf{x}), \sigma^2(\mathbf{x}))$ be the estimation at any input point $\mathbf{x}$. With the GP approximation, the tuple existence probability $\tilde{\rho}$ is approximated with $\hat{\rho} = \Pr[\hat{f}(\mathbf{x}) \in [a, b]]$. This is exactly the quantity that we bounded in Section 6.5.2, where we showed that $\tilde{\rho} \leq \rho_U$. So in this case, we filter tuples whose estimate of ρ_U is less than our threshold. Again, since ρ_U is computed from the samples, we can check this online for filtering decision as in Section 6.3.2.

6.7 Performance Evaluation

In this section, we evaluate the performance of our proposed techniques using both synthetic functions and data with controlled properties, and real workloads from the astrophysics domain.



(a) Funct1 (b) Funct2 (c) Funct3 (d) Funct4

Figure 6.4. A family of functions of different smoothness and shape used in evaluation.

6.7.1 Experimental Setup

We first use synthetic functions so that we can vary different parameters to test the performance and sensitivity of our algorithms. We now describe the settings of these functions, workload data and parameters considered.

A. Functions. We generate functions (UDFs) of different shapes in terms of bumpiness and spikiness. A simple method is to use Gaussian mixtures [39] to simulate various function shapes (which should not be confused with the input and output distributions of the UDF and by no means favors our GP approach). We vary the number of Gaussian components, which dictates the number of peaks of a function. The means of the components determine the domain of the function, and their covariance matrix determines the stretch and bumpiness of the function. We denote the dimensionality of the functions d, which is the number of input variables of the function. We observe that in real applications, many functions have low dimensionality—the ones we found in astrophysics have $d = 1$ or $d = 2$. For evaluation purposes, we vary d in a wide range of [1,10]. Besides the shape, a function is characterized by the evaluation time, denoted as T and varied in the range 1μs to 1s.

B. Input Data. By default, we consider uncertain data following Gaussian distributions, i.e., the input vector has distribution characterized by $\mathcal{N}(\boldsymbol{\mu}_I, \Sigma_I)$. μ_I is drawn from the given support of the function $[L, U]$. Σ_I determines the spread of the input distributions. For simplicity, we assume the input variables of a function

are independent, but supporting correlated inputs is not harder—we just need to sample from the joint distributions. We also consider other distributions including exponential and Gamma. We note that handling other types of distributions is similar due to the same reason (the difference is the cost of sampling).

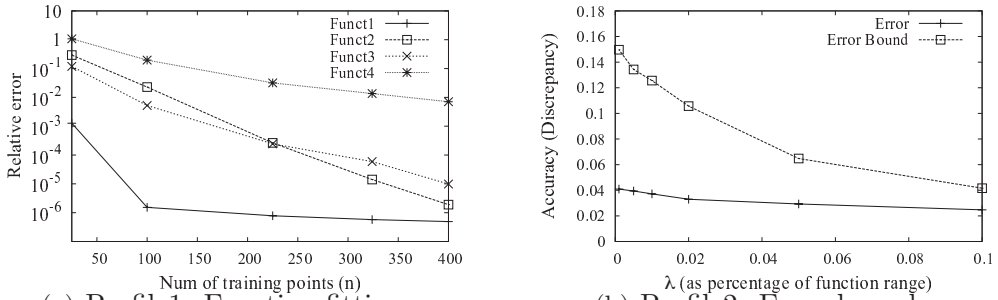
C. Accuracy Requirement. We use the discrepancy measure as the accuracy metric in our experiments. The user specifies the accuracy requirement (ϵ, δ) and the minimum interval length λ. λ is set to be a small percentage (e.g., 1%) of the range of the function. This requirement means that with probability $(1-\delta)$, for all intervals of length at least λ, the probability of an interval computed from the approximate output distribution does not differ from that computed from the true distribution by more than ϵ. For the GP approach, the error bound ϵ is allocated to two sources of errors, GP error bound ϵ_{GP} and sampling error bound ϵ_{MC}, where $\epsilon = \epsilon_{GP} + \epsilon_{MC}$. We also distribute δ so that $1 - \delta = (1 - \delta_{GP})(1 - \delta_{MC})$.

Our default setting is as follows. The domain of function $[L, U] = [0, 10]$, input standard deviation $\sigma_I = 0.5$, function evaluation time $T = 1ms$, accuracy requirement $(\epsilon = 0.1, \delta = 0.05)$, and minimum interval length $\lambda = 1\%\mathbf{x}funct\_range$. The reported results are averaged from 500 output distributions or when the algorithm converges, whichever larger. The convergence is achieved when the running time stabilizes, i.e., the algorithm is less likely to add more training points or call retraining.

6.7.2 Evaluating GP Techniques

We first evaluate the individual techniques employed in our Gaussian process algorithm, OLGAPRO. This is aimed to understand and set various internal parameters of our algorithm.

Profile 1: Accuracy of function fitting. We first choose four representative two-dimensional functions of different shape and bumpiness (see Figure 6.4). These functions are the four combinations between (i) one or five components, (ii) large or



(a) Profile1: Function fitting (b) Profile2: Error bound

Figure 6.5. Experimental results for profiling of the GP approach

small variance of Gaussian components. We will refer to these functions as $\mathcal{F}_1$, $\mathcal{F}_2$, $\mathcal{F}_3$, and $\mathcal{F}_4$. First, we check the use of GPs to model these functions. We vary the number of training points n and use basic global inference to infer the function values at test points. Figure 6.5(a) shows the relative errors for GP inference, i.e., $|\frac{\hat{f}(\mathbf{x})-f(\mathbf{x})}{f(\mathbf{x})}|$, when evaluating at a large number of test points. We observe that the simplest function $\mathcal{F}_1$ with one peak and being flat needs a small number of training points, e.g., 30, to give a good approximation. On the other hand, the most bumpy and spiky function $\mathcal{F}_4$ with five peaks requires the largest number of training points, $n > 300$, to start giving good approximations. The other two functions are in between. This confirms that the GP approach can model functions of different shape well, however the number of training points needed varies with the functions. In the later experiments, we will show that our algorithm OLGAPRO can robustly determine the number of training points needed online.

Profile 2: Behavior of error bound. We next test the behavior of our discrepancy error bound, which is described in Section 6.5.2 and shown how to compute in Algorithm 4. We compute the error bounds and measure the actual errors. Figure 6.5(b) shows the result for the function $\mathcal{F}_4$. As observed, it confirms that the error bounds are actual upper bounds, which indicates the validity of GP modeling. More interestingly, it shows how tight the bounds are (about 2 to 4 times larger than the

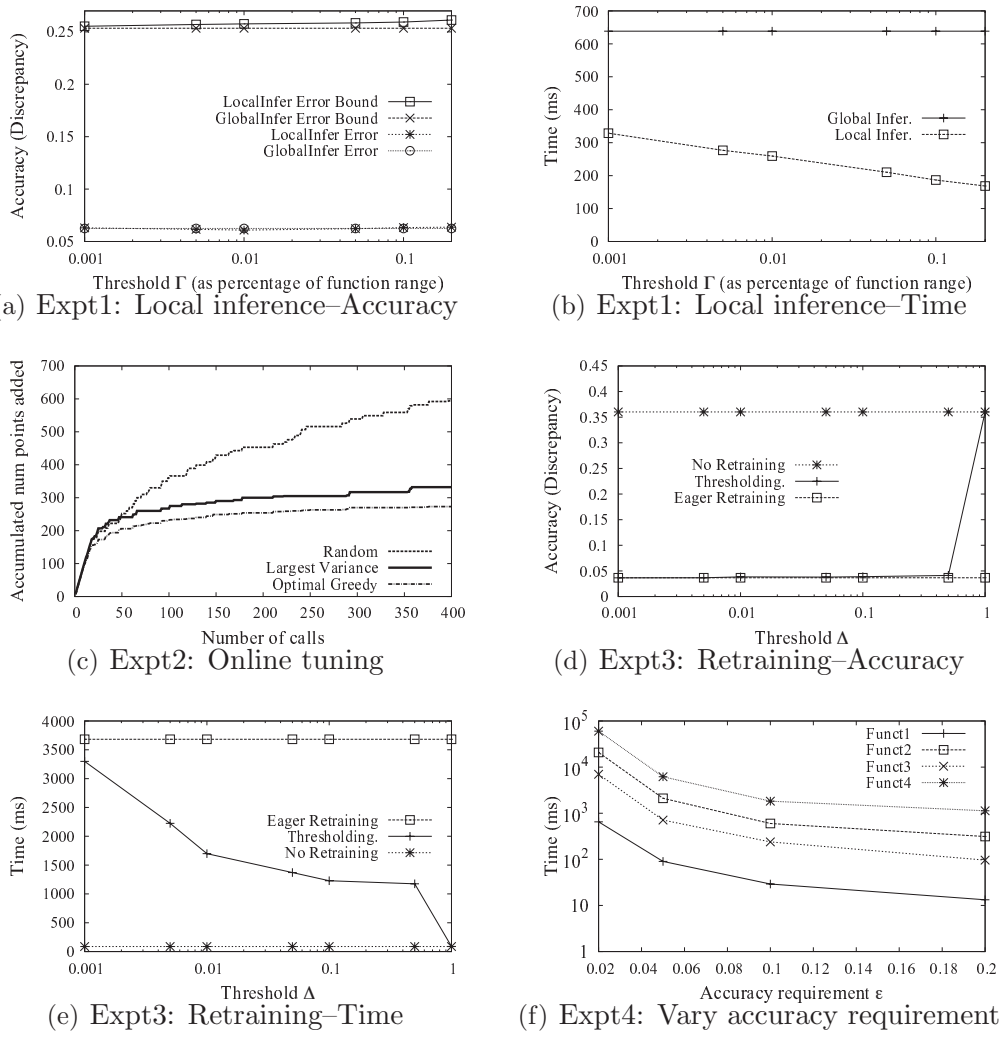
actual errors). As λ gets smaller, more intervals are considered when computing the discrepancy measure. As a result, the errors and error bounds, which are the suprema computed from a larger set of intervals, get larger. We run tests with other functions and observe the same trends.

In the following experiments, we use a stringent requirement by setting λ to be 1% of the function range.

Profile 3: Allocation of two sources of error. Our GP approach has two sources of errors, GP modeling and Monte Carlo sampling as stated in Theorem 6.5.1. Given the user-specified error bound ϵ, we want to allocate $\epsilon = \epsilon_{GP} + \epsilon_{MC}$. We vary the ratio of two error bounds to find a setting that optimizes performance. Overall, we observe setting ϵ_{MC} to be 60% to 80% of ϵ gives best performance. The bias towards a higher percentage of ϵ_{MC} can be explained as, the number of samples m is proportional to $1/\epsilon_{MC}^2$, whereas the number of training points n needed is less sensitive to ϵ_{GP}. For default we will use $r = 0.7$ in our experiments.

In the next three experiments, we evaluate the three key techniques employed in our GP algorithm. The default function is $\mathcal{F}_4$.

Expt 1: Local inference. We first consider our local inference technique as shown in Section 6.6.1. We compare the accuracy and running time of local inference with those of global inference. For now, we fix the number of training points to compare the performance of the two inference techniques. We vary the threshold Γ of local inference from 0.1% to 20% of the function range. Recall that setting Γ small corresponds to using more training points and hence similar to global inference. Our goal is to choose a setting of Γ so that local inference has similar accuracy as global inference while being faster. Figures 6.6(a) and 6.6(b) show the accuracy and running time of local inference and global inference for $\mathcal{F}_4$. We see that for most of values Γ tested, local inference is as accurate while offering a speedup from 2 to 4 times. We repeat this experiment for other functions and observe that for less bumpy functions,



(a) Expt1: Local inference–Accuracy

(b) Expt1: Local inference–Time

(c) Expt2: Online tuning

(d) Expt3: Retraining–Accuracy

(e) Expt3: Retraining–Time

(f) Expt4: Vary accuracy requirement

Figure 6.6. Experimental results for evaluating the GP approach using synthetic data and functions

the speedup for local inference is less pronounced, but the accuracy is always comparable. This is because for smooth functions, far training points still have a high weight in inference. In general, we set Γ about (0.05x function_range), which results in good accuracy and improved running time.

Expt 2: Online tuning. In Section 6.6.2, we proposed to add training points on-the-fly to meet the accuracy requirement. We now evaluate our heuristics for adding new training points by choosing samples with the largest variance. We compare it

with two following heuristics: Given an input distribution, a simple one is to choose a sample of the input at random. Another heuristics is what we call "optimal greedy". It considers all samples, and for each one, simulates adding it to compute the new error bound, and then picks the sample that has the most error bound reduction. This is only hypothetical since it is prohibitively expensive to simulate adding every sample. For the purpose of this experiment, we assume that there are 400 samples for the input, so that we can use "optimal greedy". We start with just 25 training points and add more over time as necessary. Figure 6.6(c) shows the accumulated number of training points added over time (for performance, we restrict that for every input, we do not add more than 10 training points). As can be seen, our technique using the largest variance requires fewer training points, hence runs faster, than randomly adding points. Also, it is close to our optimal greedy standard while being much faster to be run online.

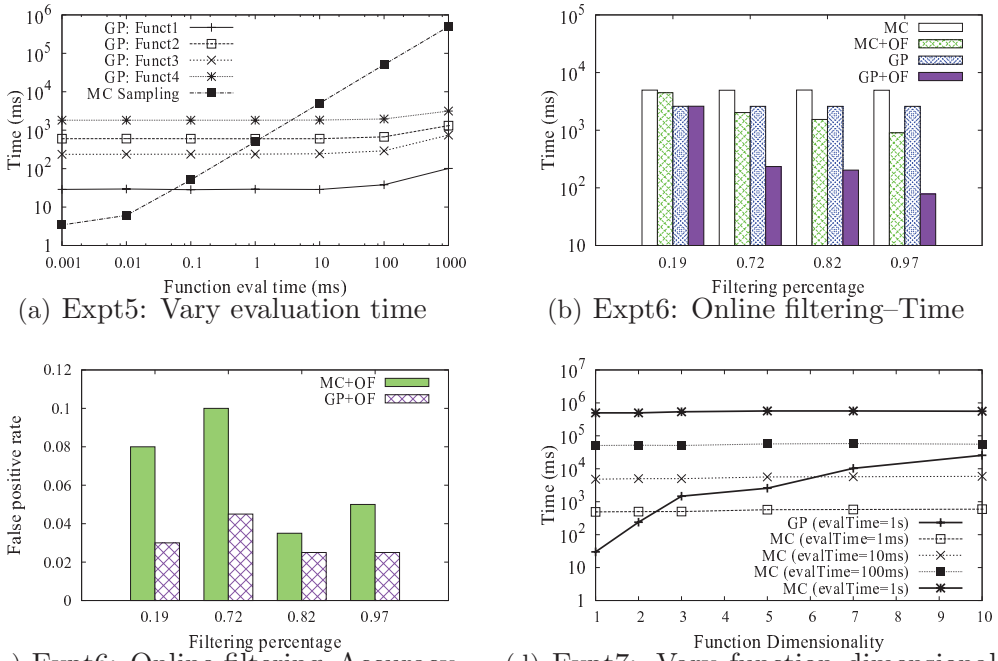
Expt 3: Retraining strategy. We now examine the performance of our retraining strategy (see Section 6.6.3). We vary our threshold Δ for retraining and compare this strategy with two other strategies: eager training every time one or more training points are added, and no training. Again, we start with a small number of training points and add more according to online tuning. Figures 6.6(d) and 6.6(e) show the accuracy and running time respectively. As expected, setting Δ smaller means retraining more often and is similar to eager retraining, while larger Δ means less retraining. We see that setting Δ less than 0.5 gives best performance, as fewer retraining calls are needed while the hyperparameters are still good estimates, hence high accuracy. We repeat this experiment with other functions and see that conservatively setting $\Delta = 0.05$ gives good performance in general. This is because, this constant Δ is small compared to the GP hyperparameters for this set of functions. In practice, Δ can be chosen in reference with the hyperparameter values.

6.7.3 GP versus Monte Carlo Approach

We next examine the performance of our complete online algorithm, OLGAPRO, as described in Algorithm 6. The internal parameters are set as mentioned above. We also compare this algorithm with the Monte Carlo approach in Section 6.3.1.

Expt 4: Varying user-specified ϵ. We run the GP algorithm for all four functions $\mathcal{F}_1$ to $\mathcal{F}_4$. We vary ϵ in the range of [0.02, 0.2]. Figure 6.6(f) shows the running time for the four functions. (We verify that the accuracy requirement ϵ is always satisfied, and omit the plot here due to space constraints.) As ϵ gets smaller, the running time increases. This is due to the fact that the number of samples is proportional to $1/\epsilon_{MC}^2$. Besides, small ϵ_{GP} requires more training points, hence higher cost for inference. This experiment also verifies the effect of the function complexity on the performance. A flat function like $\mathcal{F}_1$ needs much fewer training points than a bumpy, spiky function like $\mathcal{F}_4$, thus running time is about two orders of magnitude different. We also repeat this experiment for other input distributions including Gamma and exponential distributions, and observe very similar results. Overall, our algorithm can robustly adapt to the function complexity and the accuracy requirement.

Expt 5: Varying the evaluation time T. We now compare our GP algorithm with the Monte Carlo approach. We fix $\epsilon = 0.1$ and vary the function evaluation time T from 1μs to 1s. Figure 6.7(a) shows the running time of the two approaches for all four functions. Note that the running time for MC sampling is similar for all functions, hence we just show one line. As can be observed, the GP approach starts to outperform the sampling approach when function evaluation takes longer than 0.1ms for simple functions like $\mathcal{F}1$, and up to 10ms for complex functions like $\mathcal{F}4$. Also we note that our GP approach is almost insensitive to function evaluation time. This is because we only need to acquire more training points, hence involving function evaluation, during the early phase before the system converges. After that,



(a) Expt5: Vary evaluation time

(b) Expt6: Online filtering–Time

(c) Expt6: Online filtering–Accuracy

(d) Expt7: Vary function dimensionality

Figure 6.7. Experimental results for comparing the GP and MC approaches using synthetic data and functions

function evaluation is only infrequent. This demonstrates the applicability of the GP approach for long running functions.

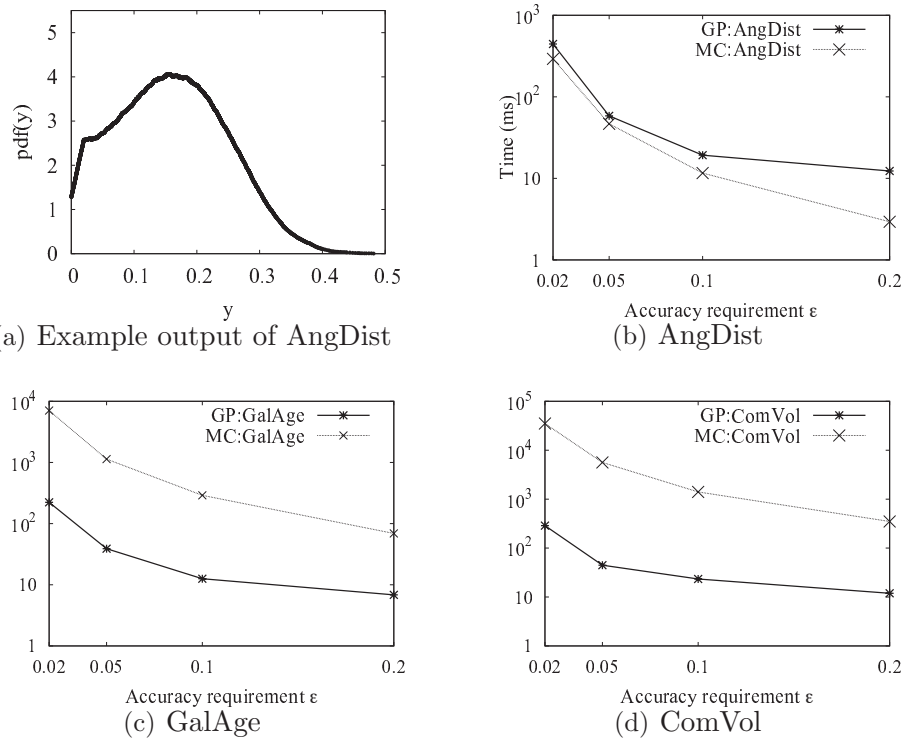
This result also shows the need for the hybrid solution. Since the function complexity is not known beforehand, it is hard to know how many training points are enough. Therefore, the hybrid solution can be performed to find out which approach is preferred. For example, for simple functions we can adopt the GP approach even when evaluation time is about 0.1ms, but for complex functions, we start to apply it when function evaluation is longer. The hybrid solution computes the function evaluation time and evaluates its complexity to automatically make this decision.

Expt 6: Optimization for selection predicates. We next examine the performance of using online filtering when there is a selection predicate. As shown in Sections 6.3.2 and 6.6.6, this can be applied for both direct MC sampling or sampling

with a GP. We vary the selection predicate, which in turn affects the rate that the output is filtered. We decide to filter an output whose tuple existence probability is less than 0.1. Figure 6.7(b) shows the running time. As seen, when the filtering rate is high, online filtering helps reduce the running time for both approaches: the running time reduces by a factor of 5 and 30 times for MC and GP respectively. We observe that the GP approach has a higher speedup because besides processing fewer samples, it results in a GP model with fewer training points, hence smaller inference cost. Figure 6.7(c) shows the false positive rates, i.e., output tuples should be filtered but are not during the sampling process. We observe that this rate is low, always less than 10%. The false negative rates are zero or negligible (less than 0.5%).

Expt 7: Varying function dimensionality d. We conduct one experiment on varying the dimension of the function d from 1 to 10. Figure 6.7(d) shows the running time of these functions for both GP and MC approaches. Since the running time using GP is insensitive to function evaluation time, we show only one line for $T = 1s$ for clarity. We observe that with GPs, high-dimensional functions incur high cost, which is due to the fact that more training points are needed to capture a larger region. This is also known as the "curse of dimensionality". We observe that even with a high dimension of 10, the GP approach still outperforms MC when function evaluation time reaches 0.1s.

The results indicate that the hybrid approach is feasible by encoding general rules based on the known dimensionality and observed evaluation time. When the function is really fast, i.e., $T \leq 0.01ms$, MC sampling is a preferred solution. For most functions we see in our applications, which have low dimensionality, we use GP approach for better performance if functions have $T \geq 1ms$. For very high-dimensional functions, we use MC approach. (We consider exploring improvements for GPs for this case in future work.)



(a) Example output of AngDist (b) AngDist

(c) GalAge (d) ComVol

Figure 6.8. Results for real astrophysics functions and SDSS data

6.7.4 Case Study: UDFs in Astrophysics

In this experiment, we consider the application of our techniques in the astrophysics domain using real functions and data. We examined functions to compute various astrophysical quantities, available in a library package at [6] and found eight functions computing a scalar value, which can all be incorporated into our framework; our algorithms treat them as black-box UDFs. Most of these functions are one and two-dimensional, usually have simple shapes but can be slow-running due to complex numerical computation. We chose three representative functions, as shown below, that vary in evaluation time across a range (while the others have similar fast evaluation times). (See Section 6.1 for the queries using these functions.)

functName	dim	evalTime (ms)
AngDist	2	0.00298
GalAge	1	0.29072
ComoveVol	2	1.82085

We use a real dataset from the Sloan digital sky survey (SDSS) project [92] and extract the necessary attributes for these functions, which are uncertain and characterized by Gaussian distributions.

We vary the accuracy requirement ϵ from 0.02 to 0.2. We verify that output distributions are non-Gaussian; an example output distribution of *AngDist* is shown in Figure 6.8(a). We compare the performance of Monte Carlo simulation with our algorithm OLGAPRO. The results are shown in Figure 6.8. (We omit the accuracy plot due to space constraints, but verify that the average errors are less than 0.5ϵ.) Overall, we observe that these functions are generally smooth and non-spiky, hence do not need many training points to model. The 1D function *GalAge* only requires around 10 training points, while the two 2D functions, *AngDist* and $ComoveVol$, require less than 40 points. OLGAPRO adds most of these training points for the first few input tuples; after that, the running time becomes stable. *AngDist* is a quite fast function and OLGAPRO is somewhat slower than MC sampling. For the other two functions, *GalAge* and $ComoveVol$, whose evaluation time is about 0.3 and 2 ms respectively, our OLGAPRO is faster than the MC sampling by one to two orders of magnitude. These results are consistent with those using synthetic functions shown above and demonstrate the applicability of our techniques for the real workloads.

6.8 Related Work

We discuss several broad areas of research related to our work on UDFs.

Work on UDFs with deterministic input. Existing work, e.g.,[18], considers queries invoking UDFs that are expensive to execute and proposes reordering the exe-

cution of the predicates. Another work [26] considers functions that are iterative and computed by numerical methods, and whose accuracy varies with computation cost. It proposes to adaptively execute these functions depending on given queries. However, this line of work considers only deterministic input, hence inherently different from this thesis work.

Gaussian process regression with uncertain input. Regression using GPs has been well studied in the machine learning literature (see [76] for a survey of existing work). However, most of the work considers deterministic input and presents offline solutions, while we consider uncertain input in an online setting. One line of work from the statistics literature most related to CLARO is [70], which uses GPs as emulators to computer code that is slow to run. It briefly mentions using sampling to handle uncertain input, but does not quantify the approximation, which we tackle in this work by deriving error bounds. Further, we present an online algorithm with different optimizations, which has not been done before. The prior work [41] computes only the mean and variance of the output. Since the UDF output in most cases is not Gaussian, this approach does not fully characterize the output distribution. In addition, the work [41] mainly considers input data that follows Gaussian distributions, while our work can support input data of any distributions.

Optimizations of Gaussian processes. Existing lines of work [69, 74] propose optimizations for inference with GPs; however, they work in an offline manner and are not suitable for our online settings due to the lack of online tuning to obtain training data on the fly and retraining strategies to reduce the training overhead. Regarding inference only, the paper [69] suggests pre-dividing the function domain into fixed local regions corresponding to local models, then runs inference using the local models and combines the results by weighting them. This is different from our local inference technique since all training points are used, and hence can be inefficient for large training datasets. The work [74] has a more similar idea to our

local inference by using sparse covariance matrices, which zeroes out low covariances, to reduce the number of training points under inference. However, it does not quantify the approximation errors, while we can control these errors using the threshold Γ as discussed in Section 6.6.1.

CHAPTER 7

CONCLUSION AND FUTURE WORK

This chapter summarizes the contributions made in this thesis work and states some research directions for future work.

7.1 Thesis Summary

We have presented CLARO, a complete system for uncertain data processing. CLARO focuses on supporting continuous-valued uncertain data, which is prevalent in scientific and sensing applications, but has been under-addressed in the existing work. It provides an end-to-end-solution to capture data uncertainty from raw data collection to query processing to final result generation. More specifically, CLARO offers two key functionalities, (*i*) raw data capture and transformation and (*ii*) probabilistic query processing. The proposed techniques aim for both accuracy and efficiency to support high data volume and online processing, which we have validated using both synthetic and real-world data and workloads. Our main contributions are summarized as follows.

- *Capture and transformation of raw data.* We proposed a probabilistic approach based on graphical modeling and inference algorithm to transform raw data into queriable tuples with quantified uncertainty. Specifically, we demonstrated this approach for the RFID data to translate noisy, raw data streams from mobile RFID readers into clean, rich event streams with location information. The experimental results show that our approach offers 49% error reduction over a state-of-the-art RFID cleaning approach [50] while scaling to read rates of over 1500 readings/sec.

Compared to the standard inference technique, our solution offers up to 7 orders-of-magnitude improvement in scalability.

- *Data models and formal semantics.* Given that uncertain data in our applications is continuous-valued, we proposed a flexible data model based on Gaussian mixture distributions to capture those continuous attributes. Furthermore, observing that processing can yield discrete distributions, we extended our data model to the mixed-type data model to capture tuple existence uncertainty that arises from conditioning operations. We employed measure theory to define the formal semantics, which lays the foundation for relational processing under this data model.
- *Relational processing under the mixed-type data model.* We examined relational processing under the mixed-type data model, with the focus on aggregation operations. We devised efficient solutions to compute the distributions of different aggregates, which are either exact or approximate with bounded errors. We then presented a technique for query planning for complex queries that meets query accuracy requirements. The experimental results show that when the existence of tuples is certain, the proposed techniques for aggregation outperform the state-of-the-art sampling approach [38] in both accuracy and throughput. When the existence of tuples is uncertain, our deterministic algorithm for max/min is faster than our proposed randomized algorithm by orders of magnitude; for sum/count, our deterministic algorithm works better given high accuracy requirements. We also evaluated query planning for complex queries using real data and workloads from the applications of object tracking and computational astrophysics. The results showed that CLARO can meet given accuracy requirements while achieving throughput of thousands of tuples per second or higher for most workloads tested. Also, when applied to the CASA tornado detection case study, the proposed techniques of CLARO are shown to produce detection results at stream speed with much improved quality.

- *Supporting user-defined functions on uncertain data.* We presented a complete solution to computing the outputs of user-defined functions (UDFs) using both Monte Carlo sampling and a learning approach based on Gaussian processes (GPs). For the GP approach, we presented new results to compute the output distribution and quantify their error bounds. We then proposed an online algorithm that can adapt to user accuracy requirements, together with a suite of novel optimizations for improved accuracy and performance. Our evaluation using both real-world and synthetic functions shows that the proposed GP approach can outperform the Monte Carlo approach with up to two orders of magnitude improvement for many UDFs of low dimensions and relatively high evaluation time, e.g., 1ms and above.

7.2 Future Work

We have considered the design and development of a system for uncertain data processing. We now discuss some directions for future work based on this thesis work.

Handling correlation. In this work, we consider independent input tuples and the operations that produce independent output tuples. Handling correlation is an important, yet difficult, unsolved problems. We now consider the different sources of correlation and possible solutions to capture it. Correlation can arise in two ways: (i) input tuples are correlated, and (ii) even when the input tuples are independent, correlation can emerge in the immediate outputs computed from some same tuples.

For the first case, given a sequence of N random variables, model testing and identification tools [13] can be used to test the randomness and determine the order of correlation if it does exist. If the input tuples are correlated, e.g., forming a time series, exact derivation of the result distribution of sum can be difficult, because it requires the use of multivariate integration. Numerical methods such as adaptive quadrature or Monte Carlo integration can be used [61]. However, these approaches are often inefficient, and hence require optimizations or new algorithms for improved

performance. Approximation techniques may depend on specific correlation structures. One technique that pertains to our weather monitoring domain is the Central Limit Theorem for time series [14]. As stated in Section 3.4, given the observed velocity series, we can use the autocorrelation function to identify sub-series length to aggregate upon so that the sub-series can be modeled as an MA model. For a series that is from an MA model, the Central Limit Theorem states that the average velocity has an asymptotic normal distribution and can be easily computed. We plan to study other correlation structures.

For the second case, if a query involves multiple operators, the intermediate results can be correlated even if the input tuples to the first operator(s) are independent. For example, if a join is followed by an aggregation, the join may produce correlated results by matching a tuple from one input with multiple tuples from the other. Then characterizing result distributions of aggregation with correlated inputs requires the full joint distribution of input tuples and is hard to compute. A general solution to computing result distributions from correlated intermediate tuples is to use sampling and density estimation to obtain the result distribution. However, this can be slow for high-volume streams. Given our focus on selection, join, and aggregation and their uses in real-world applications, we aim to identify common patterns of correlation and explore different optimizations and approximations to obtain the result distributions. A useful related technique is to exploit *lineage* [9, 101] about how correlated tuples are produced. Such lineage helps determine the correlation structure among tuples and eliminates the need of computing full joint distributions for intermediate tuples. Given the correlation structure, the last query operator has the flexibility to optimize its computation. We will aim to study the use of lineage to capture correlation and efficient techniques to evaluate them.

Extension for complex user-defined functions. In Chapter 6, we have presented a general framework to support user-defined functions (UDFs). We now discuss some

directions to support various types of UDFs including multivariate output and high-dimensional input. For functions of multivariate output, some directions such as modeling the correlation among the outputs are briefly mentioned in [76]. For high-dimensional functions, it may be the case that the output is less sensitive to some or many input dimensions. We plan to explore existing techniques from the machine learning literature such as automatic relevance determination (ARD) to find out which dimensions are relevant to the output. Another important case we have observed in the weather monitoring case is that, the output may be classified to different events, only some of which are events of interest, e.g., there is a storm or tornado. The user would want to quickly identify if there is such an event of interest, and if so, which input corresponds to it. This way, the high dimensional input can be reduced to a subset that matters most to the application, hence saving computation cost. We aim to study techniques from the machine learning literature such as active learning to address this problem.

Parallel processing for big data. We have studied several optimizations for the proposed techniques to meet performance requirements such as stream speed processing and scalability to a large number of objects. However, as the volume of available data becomes higher and higher in many applications, further optimizations or different techniques may be required. Given that the recent popularity of the MapReduce framework has facilitated the implementation of parallel processing, we aim to support parallel processing for high performance scalable uncertain data management. This would involve revising the existing techniques or designing new ones to be amenable to parallel processing. For example, if a query plan has operators independent of each other, they can be executed in parallel. We now consider a more complex example from the CLARO system, inference with Gaussian processes used as emulators for UDFs. The GP model contains the hyperparameters, which specify the mean function and the covariance function, and the training dataset. The

domain of the input can be divided into different regions, each corresponding to a sub-model. This fits well with our local inference approach. An input distribution may correspond to more than one local model. Therefore, we can draw the samples from the input distribution, group the samples according to the regions of the local models, and call inference in the appropriate models. This can also address the problem of functions with high dimensionality since the training dataset, possibly large for high dimensions, is divided into subsets in local models. Besides, this can support input data of high volume, because different input tuples may correspond to different local models, and hence can be processed concurrently. Replicating local models can further help with high data volume. These ideas can be examined and evaluated in future work.

Supporting a complete hybrid system. In this thesis, we focus on sensing and scientific data that is naturally modeled by continuous random variables. Some processing operations such as selection, group-by, and subsequently, sum of Bernoulli variables, may generate discrete distributions as seen in Chapter 5. Many of our proposed processing algorithms such as the approximation algorithm for sum and count and the online algorithm to computing UDFs using a GP model already work for discrete random variables. A direction of future work is aimed to examine existing techniques or devise new ones to form a suite of processing techniques that together support both discrete and continuous uncertainty. A follow-up study can be to compare the performance of the available techniques for different input representations, either discrete or continuous, which may offer insights into the cases where each technique can be applied.

Another direction that can be considered for the complete system is to allow for tradeoffs between accuracy and performance. For instance, instead of computing full result distributions, we can aim to compute statistics of the output if the cost of the former is too high. Once we understand the relative performance between computing

the full distribution and the different statistics, we can incorporate this knowledge into designing the processing algorithms. One use of this idea is that the user can specify some computation budget and accuracy requirements, and then the system would find a solution that best addresses these two requirements.

APPENDIX

MATHEMATICAL PROOFS

This appendix includes the proofs of the propositions and theorems presented in Chapter 5.

Proof of Proposition 5.3.1. We first show that $\mathrm{KS}(F,G) \leq \mathrm{VD}(f,g)$, where the pdf's f and g have the corresponding CDF's F and G.

$$\begin{aligned}
2(\mathrm{KS}(F,G) - \mathrm{VD}(f,g)) =& 2|F(x) - G(x)| - \int_{-\infty}^{\infty} |f(x) - g(x)|dx \\
=& |F(x) - G(x)| + |(1 - F(x)) - (1 - G(x))| \\
& - \int_{-\infty}^{x} |f(x) - g(x)|dx - \int_{x}^{\infty} |f(x) - g(x)|dx
\end{aligned}$$

$$|F(x) - G(x)| = |\int_{-\infty}^{x} f(x)dx - \int_{-\infty}^{x} g(x)dx| \leq \int_{-\infty}^{x} |f(x) - g(x)|dx.$$

Similarly:

$$|(1 - F(x)) - (1 - G(x))| \leq \int_{x}^{\infty} |f(x) - g(x)|dx$$

Since the above inequalities hold for any x, we have $\mathrm{KS}(F,G) \leq \mathrm{VD}(f,g)$. Additionally, $\mathrm{KS}(F,G)$ can be arbitrarily smaller than $\mathrm{VD}(f,g)$. For instance, if f is the uniform distribution of the set of intervals

$$[0,1] \cup [2,3] \cup \ldots \cup [2k-2, 2k-1]$$

and g is the uniform distribution on

$$[1,2] \cup [3,4] \cup \ldots \cup [2k-1, 2k]$$

then $\mathrm{KS}(F,G) = 1/k$ whereas $\mathrm{VD}(f,g) = 1$. □

Proof of Theorem 5.4.1. We first consider the sum of two random variables, $S = X_1 + X_2$, X_1 and X_2 be Gaussian mixtures of m_1 and m_2 components. That is:

$$\begin{aligned} f_1(x) &= p_{11}N(\mu_{11},\sigma_{11}) + ... + p_{1m_1}N(\mu_{1m_1},\sigma_{1m_1}) \\ f_2(x) &= p_{21}N(\mu_{21},\sigma_{21}) + ... + p_{2m_2}N(\mu_{2m_2},\sigma_{2m_2}) \end{aligned}$$

The pdf of the sum S can be written as:

$$\begin{aligned} f_S(s) &= \int_{x_1}\int_{x_2:(x_1+x_2=s)} f_1(x_1)f_2(x_2)dx_2dx_1 \\ &= \int_{-\infty}^{+\infty} f_1(x)f_2(s-x)dx \\ f_1(x)f_2(s-x) &= [p_{11}N(\mu_{11},\sigma_{11}) + ... + p_{1m_1}N(\mu_{1m_1},\sigma_{1m_1})] \\ &\quad [p_{21}N(\mu_{21},\sigma_{21}) + ... + p_{2m_2}N(\mu_{2m_2},\sigma_{2m_2})] \\ &= \sum_{i=1,j=1}^{m_1,m_2} p_{1i}p_{2j}\frac{1}{2\pi\sigma_{1i}\sigma_{2j}}e^{-\frac{1}{2}(\frac{(x-\mu_{1i})^2}{\sigma_{11}^2}+\frac{(x-\mu_{2j})^2}{\sigma_{2j}^2})} \end{aligned}$$

Now consider the integral of one term of the sum:

$$A = \int_{-\infty}^{+\infty} p_{1i}p_{2j}\frac{1}{2\pi\sigma_{1i}\sigma_{2j}}e^{-\frac{1}{2}(\frac{(x-\mu_{1i})^2}{\sigma_{1i}^2}+\frac{(x-\mu_{2j})^2}{\sigma_{2j}^2})}$$

Let B be the term in the exponent, except for the constant:

$$\begin{aligned} B &= \frac{(x-\mu_{1i})^2}{\sigma_{1i}^2} + \frac{(x-\mu_{2j})^2}{\sigma_{2j}^2} \\ &= \frac{1}{\sigma_{11}^2\sigma_{21}^2}\left[(\sigma_{1i}^2+\sigma_{2j}^2)x^2 - 2\sigma_{2j}^2x\mu_{1i} + 2\sigma_{1i}^2x(\mu_{2j}-s) + \sigma_{2j}^2\mu_{1i}^2 + \sigma_{1i}^2(\mu_{2j}-s)^2\right] \\ &= \frac{\sigma_{1i}^2+\sigma_{2j}^2}{\sigma_{11}^2\sigma_{21}^2}\left[\left(x - \frac{\sigma_{2j}^2\mu_{1i}+\sigma_{1i}^2(s-\mu_{2j})}{\sigma_{1i}^2+\sigma_{2j}^2}\right)^2 - \left(\frac{\sigma_{2j}^2\mu_{1i}+\sigma_{1i}^2(s-\mu_{2j})}{\sigma_{1i}^2+\sigma_{2j}^2}\right)^2 + \frac{\sigma_{2j}^2\mu_{1i}^2+\sigma_{1i}^2(s-\mu_{2j})^2}{\sigma_{1i}^2+\sigma_{2j}^2}\right] \\ &= \frac{\sigma_{1i}^2+\sigma_{2j}^2}{\sigma_{1i}^2\sigma_{2j}^2}\left[\left(x - \frac{\sigma_{2j}^2\mu_{1i}+\sigma_{1i}^2(s-\mu_{2j})}{\sigma_{1i}^2+\sigma_{2j}^2}\right)^2 + \frac{\sigma_{1i}^2+\sigma_{2j}^2(s-\mu_{1i}-\mu_{2j})^2}{(\sigma_{1i}^2\sigma_{2j}^2)^2}\right] \end{aligned}$$

Substitute into A, we have:

$$\begin{aligned} A &= \int_{-\infty}^{+\infty} p_{1i}p_{2j}\frac{1}{2\pi\sigma_{1i}\sigma_{2j}} e^{\frac{1}{2}\frac{\sigma_{1i}^2+\sigma_{2j}^2}{\sigma_{1i}^2\sigma_{2j}^2}\left(x-\frac{\sigma_{2j}^2\mu_{1i}+\sigma_{1i}^2(s-\mu_{2j})}{\sigma_{1i}^2+\sigma_{2j}^2}\right)^2} e^{\frac{(s-\mu_{1i}-\mu_{2j})^2}{\sigma_{1i}^2+\sigma_{2j}^2}} dx \\ &= \frac{p_{1i}p_{2j}}{\sqrt{2\pi}\sqrt{\sigma_{1i}^2+\sigma_{2j}^2}} e^{\frac{(s-\mu_{1i}-\mu_{2j})^2}{\sigma_{1i}^2+\sigma_{2j}^2}} \int_{-\infty}^{+\infty} \frac{1}{\sqrt{2\pi}\frac{\sigma_{1i}\sigma_{2j}}{\sqrt{\sigma_{1i}^2+\sigma_{2j}^2}}} e^{\frac{1}{2}\frac{\sigma_{1i}^2+\sigma_{2j}^2}{\sigma_{1i}^2\sigma_{2j}^2}\left(x-\frac{\sigma_{2j}^2\mu_{1i}+\sigma_{1i}^2(s-\mu_{2j})}{\sigma_{1i}^2+\sigma_{2j}^2}\right)^2} dx \end{aligned}$$

The integral is equal to 1 since it is the integral of the pdf of a Gaussian distribution. Hence:

$$A = \frac{p_{1i}p_{2j}}{\sqrt{2\pi}\sqrt{\sigma_{1i}^2+\sigma_{2j}^2}} e^{\frac{(s-\mu_{1i}-\mu_{2j})^2}{\sigma_{1i}^2+\sigma_{2j}^2}}$$

This is one component of a Gaussian mixture with mean $(\mu_{1i}+\mu_{2j})$, variance $(\sigma_{1i}^2+\sigma_{2j}^2)$ and coefficient $p_{1i}p_{2j}$. Therefore, the theorem is proved for the case of $N = 2$.

The generalization to an arbitrary N is straightforward since we can do the sum for two distributions at a time by computing the partial sum and then summing it with the next distribution.

Proof of Lemma 5.5.1 Because a cumulative distribution is non-decreasing, for any $x < y < z$, $F_t^M(x) \leq F_t^M(y) \leq F_t^M(z)$. Consequently if for some α, β, γ, c_x/α and c_z/α are under-estimates for $F_t^M(x)$ and $F_t^M(z)$ such that

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

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

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

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

We proceed by induction on the generation. Clearly for $g = 0$, the result is true because c_1 and c_n are computed exactly. Consider an interval $[a, b]$ at step t, characterized by generation g and net shifting effect s, and assume that the following inequality holds for v in $\{a, b\}$ before updating with tuple t.

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

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

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

Proof of Lemma 5.5.2 Suppose $\mathcal{I} = \{I_1, I_2, \ldots, I_m\}$ where $I_i = [a_i, b_i]$. The lemma follows because $\epsilon \leq c_{b_1} \leq c_{a_1}(1+\epsilon')$, $c_{a_m} \leq c_{b_m} \leq 1$ and for all $i \in [m-1]$, $c_{a_{i+1}} \geq c_{a_i}\sqrt{1+\epsilon'}$. □

Proof of Lemma 5.5.3 We define the *width* of an interval $I = [a, b]$ to be $b-a+1$. Note that the generation 0 interval has width n and that every interval has width at least 1. The lemma follows from the fact that if a generation g interval I is subpartitioned into generation $g+1$ intervals $I_1, I_2, \ldots I_k$ then each $I_i, i \in [k]$, has a width of at most half of the width of I. □

Proof of Theorem 5.5.2 From Lemma 5.5.3, for any interval $[a, b]$, if we have compensated for the net shifting effect by $\bar{c}_a = c_a/(\sqrt{1+\epsilon'})^s$ and $\bar{c}_b = c_b/(\sqrt{1+\epsilon'})^s$, then we have:

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

Also, from the algorithm, we have: $\bar{c}_a \leq \bar{c}_b \leq (1+\epsilon')\bar{c}_a$. Therefore, as shown in our analysis in the proof of Lemma 5.5.1, the multiplicative error is $(1+\epsilon')^{g+1} \leq (1+\epsilon')^{\log \mathbb{U}+1}$. It can be shown using Taylor's theorem that $\epsilon' \leq \epsilon/((1+0.5\epsilon e^{\epsilon})(\log \mathbb{U}+1))$ suffices to ensure that the multiplicative error (and therefore the additive error since all quantities are less than 1) is less than ϵ.

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

Proof of Theorem 5.5.3 We first consider the error accumulated by repeatedly "rounding" $F(x)$, as defined in Equation 5.5 in Section 5.5.4, to construct $\text{LinCDF}_{P_t}(x)$. We first note that for any x,

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

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

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

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

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

Proof of Proposition 5.7.1. Let $G{=}(p, f)$ and $\tilde{G}{=}(\tilde{p}, \tilde{f})$ be two mixed-type distributions of attributes $\mathbf{X} = (X_1, X_2, ..., X_n)$. If $p = \tilde{p} = 1$, and X_i's are independent of each other and each X_i is bounded with a KSerror ϵ_i, then $\mathrm{KSM}(G, \tilde{G})$ can be written as:

$$\mathrm{KSM}(G, \tilde{G}) = \max_{\mathbf{o}} \sup_{\mathbf{x}} |\mathbb{P}_{\tilde{\mathbf{X}}} [\langle \mathbf{o}, \mathbf{x} \rangle] - \mathbb{P}_{\mathbf{X}} [\langle \mathbf{o}, \mathbf{x} \rangle]|$$

Consider a vector $\mathbf{x}$ and an ordering $\mathbf{o}$:

$$\begin{aligned}A &= |\mathbb{P}_{\tilde{\mathbf{X}}} [\langle \mathbf{o}, \mathbf{x} \rangle] - \mathbb{P}_{\mathbf{X}} [\langle \mathbf{o}, \mathbf{x} \rangle]| \\ &= |\prod_i \mathbb{P}\left[\tilde{X}_i\ o_i\ x_i\right] - \prod_i \mathbb{P}[X_i\ o_i\ x_i]| \\ &\leq |\prod_i (\mathbb{P}[X_i\ o_i\ x_i] + \epsilon_i) - \prod_i \mathbb{P}[X_i\ o_i\ x_i]| \\ &= |\epsilon_1 \prod_{i \neq 1} (\mathbb{P}[X_i\ o_i\ x_i] + \epsilon_i) \\ &\quad + \mathbb{P}[X_1\ o_1\ x_1] (\sum_{i \neq 1} \epsilon_i \prod_j (\mathbb{P}[X_j\ o_j\ x_j] + \epsilon_j))| \\ &\leq \sum_i \epsilon_i\end{aligned}$$

The third line above assumes $\mathbb{P}\left[\tilde{X}_i \; o_i \; x_i\right] \geq \mathbb{P}\left[X_i \; o_i \; x_i\right]$ w.l.o.g. The last line holds because $(\mathbb{P}\left[X_i \; o_i \; x_i\right] + \epsilon_i)$, being an upper bound for $\mathbb{P}\left[\tilde{X}_i \; o_i \; x_i\right]$, and can be replaced by 1 if it is greater than 1. $\square$

Proof of Theorem 5.7.2. We consider a tuple t having a mixed type distribution $(\tilde{p_t}, \tilde{F_t})$, which is an $(\epsilon, 0)$ approximation of the exact distribution (p_t, F_t). Let $\bar{t}$ denote the output tuple after applying a selection on t using a range condition. Again, the approximate distribution of $\bar{t}$ is denoted by $(\tilde{p_{\bar{t}}}, \tilde{F_{\bar{t}}})$, while the corresponding exact distribution is $(p_{\bar{t}}, F_{\bar{t}})$.

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

After selection,

$$\tilde{F_{\bar{t}}}(x) = \frac{\tilde{F_t}(x)}{\tilde{F_t}(u)} \quad \text{and} \quad F_{\bar{t}}(x) = \frac{F_t(x)}{F_t(u)}, \quad x \leq u$$

The first error component from the approximate CDF is:

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

The second error component from the approximate CDF is:

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

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

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

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

BIBLIOGRAPHY

[1] R. J. Adler. Some new random field tools for spatial analysis. In *Stochastic enviromental research and risk assessment*, 2008.

[2] P. Agrawal and J. Widom. Continuous uncertainty in trio. In *Proceedings of the 3rd International Workshop on Management of Uncertain Data (MUD)*, 2009.

[3] T. Ando. *Bayesian Model Selection and Statistical Modeling*. Chapman and Hall/CRC, 2012.

[4] L. Antova, T. Jansen, C. Koch, and D. Olteanu. Fast and simple relational processing of uncertain data. In *ICDE*, pages 983–992, 2008.

[5] L. Antova, C. Koch, and D. Olteanu. From complete to incomplete information and back. In *SIGMOD*, pages 713–724, New York, NY, USA, 2007. ACM.

[6] `http://idlastro.gsfc.nasa.gov`. IDL Astronomy Library.

[7] D. Barbará, H. Garcia-Molina, and D. Porter. The management of probabilistic data. *IEEE Trans. Knowl. Data Eng.*, 4(5):487–502, 1992.

[8] N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger. The R*-tree: an efficient and robust access method for points and rectangles. In *SIGMOD*, pages 322–331, New York, NY, USA, 1990. ACM.

[9] O. Benjelloun, A. D. Sarma, A. Y. Halevy, and J. Widom. Uldbs: Databases with uncertainty and lineage. In *VLDB*, pages 953–964, 2006.

[10] C. M. Bishop. *Pattern recognition and machine learning*. Springer-Verlag New York, Inc., 2009.

[11] X. Boyen and D. Koller. Tractable inference for complex stochastic processes. In *Proceedings of the 14th Conference on Uncertainty in Artificial Intelligence—UAI 1998*, pages 33–42. San Francisco: Morgan Kaufmann, 1998.

[12] H. C. Bravo and R. Ramakrishnan. Optimizing mpf queries: decision support and probabilistic inference. In *SIGMOD*, pages 701–712, New York, NY, USA, 2007. ACM.

[13] P. J. Brockwell and R. A. Davis. *Time Series: Theory and Methods*. Springer Series in Statistics, 1998.

[14] D. Carney, U. Çetintemel, M. Cherniack, C. Convey, S. Lee, G. Seidman, M. Stonebraker, N. Tatbul, and S. Zdonik. Monitoring streams: a new class of data management applications. In *VLDB*, pages 215–226. VLDB Endowment, 2002.

[15] `http://www.casa.umass.edu/`. Engineering Research Center for Collaborative Adaptive Sensing of the Atmosphere (CASA).

[16] G. Casella and R. Berger. *Statistical inference (Second ed.)*. Duxbury Press, Belmont, CA, 2001.

[17] R. Cavallo and M. Pittarelli. The theory of probabilistic databases. In P. M. Stocker, W. Kent, and P. Hammersley, editors, *VLDB*, pages 71–81. Morgan Kaufmann, 1987.

[18] S. Chaudhuri and K. Shim. Optimization of queries with user-defined predicates. In *ACM Transactions on Database Systems*, pages 87–98, 1996.

[19] S. S. Chawathe, V. Krishnamurthy, S. Ramachandran, and S. E. Sarma. Managing RFID data. In *VLDB*, pages 1189–1195, 2004.

[20] R. Cheng, D. V. Kalashnikov, and S. Prabhakar. Evaluating probabilistic queries over imprecise data. In *SIGMOD*, pages 551–562, 2003.

[21] R. Cocci, T. Tran, Y. Diao, and P. Shenoy. Efficient data interpretation and compression over rfid streams. In *ICDE*, 2008. Poster.

[22] G. Cormode and M. Garofalakis. Sketching probabilistic data streams. In *SIGMOD*, pages 281–292, New York, NY, USA, 2007. ACM.

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

[24] N. N. Dalvi and D. Suciu. Management of probabilistic data: foundations and challenges. In *PODS*, pages 1–12, 2007.

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

[26] M. Denny and M. J. Franklin. Adaptive execution of variable-accuracy functions. In *Proceedings of the 32nd international conference on Very large data bases*, VLDB '06, pages 547–558, 2006.

[27] A. Deshpande, C. Guestrin, and S. Madden. Using probabilistic models for data management in acquisitional environments. In *CIDR*, pages 317–328, 2005.

[28] A. Deshpande, C. Guestrin, S. Madden, J. M. Hellerstein, and W. Hong. Model-driven data acquisition in sensor networks. In *VLDB*, pages 588–599, 2004.

[29] A. Deshpande and S. Madden. MauveDB: supporting model-based user views in database systems. In *SIGMOD*, pages 73–84, 2006.

[30] Y. Diao, B. Li, A. Liu, L. Peng, C. Sutton, T. Tran, and M. Zink. Capturing data uncertainty in high-volume stream processing. In *CIDR*, 2009.

[31] A. Doucet, N. de Freitas, and N. Gordon, editors. *Sequential Monte Carlo in Practice*. Springer-Verlag, 2001.

[32] I. Ehrenberg, C. Floerkemeier, and S. Sarma. Inventory management with an RFID-equipped mobile robot. In *IEEE Conference on Automation Science and Engineering (CASE)*, pages 1020–1026, 2007.

[33] C. Floerkemeier and M. Lampe. Issues with RFID usage in ubiquitous computing applications. In *Pervasive*, pages 188–193, 2004.

[34] M. J. Franklin, S. R. Jeffery, S. Krishnamurthy, F. Reiss, S. Rizvi, E. W. 0002, O. Cooper, A. Edakkunni, and W. Hong. Design considerations for high fan-in systems: The HiFi approach. In *CIDR*, pages 290–304, 2005.

[35] S. Fruhwirth-Schnatter. *Finite Mixture and Markov Switching Models*. Springer, 2006.

[36] S. Garfinkel and B. Rosenberg, editors. *RFID: Applications, Security, and Privacy*. Addison-Wesley, 2005.

[37] M. N. Garofalakis, K. P. Brown, M. J. Franklin, J. M. Hellerstein, D. Z. Wang, E. Michelakis, L. Tancau, E. Wu, S. R. Jeffery, and R. Aipperspach. Probabilistic data management for pervasive computing: The data furnace project. *IEEE Data Eng. Bull.*, 29(1):57–63, 2006.

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

[39] M. Geoffrey and P. David. *Finite Mixture Models*. Wiley-Interscience, 2000.

[40] A. L. Gibbs and F. E. Su. On choosing and bounding probability metrics. *International Statistical Review*, 70:419–435, 2002.

[41] A. Girard, J. Q. Candela, R. Murray-Smith, and C. E. Rasmussen. Gaussian process priors with uncertain inputs - application to multiple-step ahead time series forecasting. In *NIPS*, 2003.

[42] H. Gonzalez, J. Han, X. Li, and D. Klabjan. Warehousing and analyzing massive RFID data sets. In *ICDE*, page 83, 2006.

[43] C. Guestrin, P. Bodi, R. Thibau, M. Paski, and S. Madden. Distributed regression: an efficient framework for modeling sensor network data. In *Proceedings of the third international symposium on Information processing in sensor networks (IPSN)*, pages 1–10, 2004.

[44] D. Hähnel, W. Burgard, D. Fox, K. Fishkin, and M. Philipose. Mapping and localization with RFID technology. In *Proc. of the IEEE International Conference on Robotics and Automation (ICRA)*, 2004.

[45] J. M. Hellerstein and J. F. Naughton. Query execution techniques for caching expensive methods. In *SIGMOD*, pages 423–434, New York, NY, USA, 1996.

[46] R. Jampani, F. Xu, M. Wu, L. L. Perez, C. M. Jermaine, and P. J. Haas. Mcdb: a monte carlo approach to managing uncertain data. In *SIGMOD*, pages 687–700, 2008.

[47] T. S. Jayram, S. Kale, and E. Vee. Efficient aggregation algorithms for probabilistic data. In *SODA*, pages 346–355, Philadelphia, PA, USA, 2007. Society for Industrial and Applied Mathematics.

[48] T. S. Jayram, A. McGregor, S. Muthukrishnan, and E. Vee. Estimating statistical aggregates on probabilistic data streams. In *PODS*, pages 243–252, New York, NY, USA, 2007. ACM.

[49] T. S. Jayram, A. McGregor, S. Muthukrishnan, and E. Vee. Estimating statistical aggregates on probabilistic data streams. *ACM Trans. Database Syst.*, 33(4), 2008.

[50] S. R. Jeffery, M. J. Franklin, and M. N. Garofalakis. An adaptive RFID middleware for supporting metaphysical data independence. *VLDB Journal*, 2007.

[51] M. I. Jordan, editor. *Learning in graphical models*. MIT Press, Cambridge, MA, USA, 1999.

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

[53] B. Kanagal and A. Deshpande. Efficient query evaluation over temporally correlated probabilistic streams. In *ICDE*, 2009.

[54] G. A. Kantor and S. Singh. Preliminary results in range-only localization and mapping. In *Proceedings of the IEEE Conference on Robotics and Automation (ICRA '02)*, volume 2, pages 1818 – 1823, May 2002.

[55] Kiva. `http://www.kiva.edu/`.

[56] C. Koch and D. Olteanu. Conditioning probabilistic databases. In *VLDB*, 2008.

[57] P. Kulkarni, P. J. Shenoy, and D. Ganesan. Approximate initialization of camera sensor networks. In *EWSN*, pages 67–82, 2007.

[58] J. F. Kurose, E. Lyons, D. McLaughlin, D. Pepyne, B. Philips, D. Westbrook, and M. Zink. An end-user-responsive sensor network architecture for hazardous weather detection, prediction and response. In *AINTEC*, pages 1–15, 2006.

[59] L. V. S. Lakshmanan, N. Leone, R. B. Ross, and V. S. Subrahmanian. Probview: A flexible probabilistic database system. *ACM Trans. Database Syst.*, 22(3):419–469, 1997.

[60] L. Liao, D. J. Patterson, D. Fox, and H. Kautz. Learning and inferring transportation routines. *Artificial Intelligence*, 171(5-6):311–331, 2007.

[61] J. S. Liu. *Monte Carlo Strategies in Scientific Computing.* Springer, October 2002.

[62] X. Liu, M. D. Corner, and P. Shenoy. Ferret: Rfid localization for pervasive multimedia. In *Proceedings of the 8th International Conference on Ubiquitous Computing*, 2006.

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

[64] S. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong. The design of an acquisitional query processor for sensor networks. In *SIGMOD*, pages 491–502, 2003.

[65] M. Montemerlo, S. Thrun, D. Koller, and B. Wegbreit. FastSLAM: A factored solution to the simultaneous localization and mapping problem. In *Proceedings of the AAAI National Conference on Artificial Intelligence*, Edmonton, Canada, 2002. AAAI.

[66] K. Murphy. *Dynamic Bayesian Networks: Representation, Inference and Learning.* Ph.d. thesis, University of California, Berkeley, July 2002.

[67] M. Mutsuzaki, M. Theobald, A. de Keijzer, J. Widom, P. Agrawal, O. Benjelloun, A. D. Sarma, R. Murthy, and T. Sugihara. Trio-One: Layering uncertainty and lineage on a conventional dbms (demo). In *CIDR*, pages 269–274, 2007.

[68] B. Ng, L. Peshkin, and A. Pfeffer. Factored particles for scalable monitoring. In *Conference on Uncertainty in Artificial Intelligence (UAI)*, 2002.

[69] D. T. Nguyen and J. Peters. Local gaussian process regression for real-time model-based robot control. In *International Conference on Intelligent Robots and Systems (IROS)*, 2008.

[70] A. O'Hagan. Bayesian analysis of computer code outputs: A tutorial. In *Reliability Engineering and System Safety*, 2006.

[71] D. J. Patterson, L. Liao, D. Fox, and H. A. Kautz. Inferring high-level behavior from low-level sensors. In *Ubicomp*, pages 73–89, 2003.

[72] L. Peng, Y. Diao, and A. Liu. Optimizing probabilistic query processing on continuous uncertain data. *PVLDB*, 4(11):1169–1180, 2011.

[73] Y. Qi, R. Jain, S. Singh, and S. Prabhakar. Threshold query optimization for uncertain data. In *SIGMOD*, pages 315–326, New York, NY, USA, 2010.

[74] A. Ranganathan and M. H. Yang. Online sparse matrix gaussian process regression and vision applications. In *ECCV*, 2008.

[75] J. Rao, S. Doraiswamy, H. Thakkar, and L. S. Colby. A deferred cleansing method for RFID data analytics. In *VLDB*, pages 175–186, 2006.

[76] C. E. Rasmussen and C. K. I. Williams. *Gaussian processes for machine learning*. MIT Press, 2009.

[77] C. Re, N. N. Dalvi, and D. Suciu. Efficient top-k query evaluation on probabilistic data. In *ICDE*, pages 886–895, 2007.

[78] C. Ré, J. Letchner, M. Balazinska, and D. Suciu. Event queries on correlated probabilistic streams. In *SIGMOD*, pages 715–728, 2008.

[79] C. Ré and D. Suciu. Approximate lineage for probabilistic databases. In *VLDB*, 2008.

[80] S. Rizvi, S. R. Jeffery, S. Krishnamurthy, M. J. Franklin, N. Burkhart, A. Edakkunni, and L. Liang. Events on the edge. In *SIGMOD*, pages 885–887, 2005.

[81] A. D. Sarma, O. Benjelloun, A. Y. Halevy, and J. Widom. Working models for uncertain data. In *ICDE*, page 7, 2006.

[82] A. D. Sarma, M. Theobald, and J. Widom. Exploiting lineage for confidence computation in uncertain and probabilistic databases. In *ICDE*, pages 1023–1032, 2008.

[83] T. Sauer. *Numerical Analysis*. Addison Wesley, 2005.

[84] D. Schulz, W. Burgard, D. Fox, and A. Cremens. People tracking with mobile robots using sample-based joint probabilistic data association filters. *International Journal of Robotics Research (IJRR)*, 22(2):99–116, 2003.

[85] `http://www.scidb.org`. SciDB Project.

[86] P. Sen, A. Deshpande, and L. Getoor. Exploiting shared correlations in probabilistic databases. In *VLDB*, 2008.

[87] A. Silberstein, R. Braynard, and J. Yang. Constraint chaining: on energy-efficient continuous monitoring in sensor networks. In *SIGMOD*, pages 157–168, 2006.

[88] S. Singh, C. Mayfield, R. Shah, S. Prabhakar, S. E. Hambrusch, J. Neville, and R. Cheng. Database support for probabilistic attributes and tuples. In *ICDE*, pages 1053–1061, 2008.

[89] `http://www.sdss.org`. Sloan Digital Sky Survey.

[90] A. Smith, H. Balakrishnan, M. Goraczko, and N. B. Priyantha. Tracking Moving Devices with the Cricket Location System. In *International Conference on Mobile Systems, Applications and Services (Mobisys 2004)*, Boston, MA, June 2004.

[91] D. Suciu, A. Connolly, and B. Howe. Embracing uncertainty in large-scale computational astrophysics. In *Proceedings of the 3rd International Workshop on Management of Uncertain Data (MUD)*, 2009.

[92] A. S. Szalay, P. Kunszt, A. Thakar, J. Gray, D. Slutz, and R. J. Brunner. Designing and mining multi-terabyte astronomy archives: The sloan digital sky survey. In *SIGMOD*, pages 451–462, 2000.

[93] A. Thiagarajan and S. Madden. Querying continuous functions in a database system. In *SIGMOD*, pages 791–804, 2008.

[94] T. Tran, C. Sutton, R. Cocci, Y. Nie, Y. Diao, and P. Shenoy. Probabilistic inference over rfid streams in mobile environments. In *ICDE*, 2009.

[95] T. T. L. Tran, A. McGregor, Y. Diao, L. Peng, and A. Liu. Conditioning and aggregating uncertain data streams: Going beyond expectations. In *Proceedings of VLDB*, 2010.

[96] T. T. L. Tran, L. Peng, Y. Diao, A. McGregor, and A. Liu. Claro: Modeling and processing uncertain data streams. *VLDB Journal*, 2011.

[97] T. T. L. Tran, L. Peng, B. Li, Y. Diao, and A. Liu. Pods: A new model and processing algorithms for uncertain data streams. In *SIGMOD*, 2010.

[98] D. Z. Wang, E. Michelakis, M. Garofalakis, and J. Hellerstein. Bayesstore: Managing large, uncertain data repositories with probabilistic graphical models. In *VLDB*, 2008.

[99] F. Wang and P. Liu. Temporal management of RFID data. In *VLDB*, pages 1128–1139, 2005.

[100] E. Welbourne, N. Khoussainova, J. Letchner, Y. Li, M. Balazinska, G. Borriello, and D. Suciu. Cascadia: a system for specifying, detecting, and managing rfid events. In *MobiSys*, pages 281–294, 2008.

[101] J. Widom. Trio: A system for integrated management of data, accuracy, and lineage. In *CIDR*, pages 262–276, 2005.

[102] J. Xie, J. Yang, Y. Chen, H. Wang, and P. S. Yu. A sampling-based approach to information recovery. In *ICDE*, pages 476–485, 2008.