Using Probabilistic Models for Data Management in Acquisitional Environments

Amol Deshpande[*]
University of Maryland
amol@cs.umd.edu

Carlos Guestrin[*]
CMU
guestrin@cs.cmu.edu

Samuel R. Madden
MIT
madden@csail.mit.edu

> There exists a black kingdom which the eyes of man avoid because its landscape fails signally to flatter them. This darkness, which he imagines he can dispense with in describing the light, is error with its unknown characteristics... Error is certainty's constant companion. Error is the corollary of evidence. And anything said about truth may equally well be said about error: the delusion will be no greater.
>
> (*Preface to a Modern Mythology*, Louis Aragon, French Poet, 1926.)

Abstract

Traditional database systems, particularly those focused on capturing and managing data from the real world, are poorly equipped to deal with the noise, loss, and uncertainty in data. We discuss a suite of techniques based on probabilistic models that are designed to allow database to tolerate noise and loss. These techniques are based on exploiting correlations to predict missing values and identify outliers. Interestingly, correlations also provide a way to give approximate answers to users at a significantly lower cost and enable a range of new types of queries over the correlation structure itself. We illustrate a host of applications for our new techniques and queries, ranging from sensor networks to network monitoring to data stream management. We also present a unified architecture for integrating such models into database systems, focusing in particular on *acquisitional systems* where the cost of capturing data (*e.g.*, from sensors) is itself a significant part of the query processing cost.

1 Introduction

The vision of ubiquitous computing promises to spread information technology throughout our lives. Though this vision can be compelling, it also threatens to overwhelm us with a flood of information, much of which is spurious, irrelevant, or misleading. Thus, the challenge of realizing this vision is separating the relevant, timely, and useful information out of this flood of data. The data management community has made significant progress towards achieving this goal – by providing tools that load and clean the data, languages and systems that can query the data (*e.g.*,[52, 36, 10, 38]), and algorithms that mine the data for patterns and relationships that are of interest [33].

These efforts have largely been focused on mitigating data complexity once it has been captured and stored inside of a traditional computing infrastructure. In contrast, we are focusing on techniques designed to take an active role in managing this wealth of data by managing when, where, and with what frequency data is acquired from distributed information systems. There are many modern systems where the capability of local nodes to generate data far outstrips the resources available to transmit or store that data. Nodes in a sensor network, for example, typically have processors that run at several megahertz, with data collection hardware capable of collecting many kilosamples per second, but radios that only transmit kilobytes per second aggregate across all of the nodes in the network. Worse yet, these nodes are battery powered, and, when sampling at maximum rates, only have sufficient energy to last for a few days [46]. Similarly, routers on the Internet can produce huge amounts of network monitoring traffic, so much so that the links which that traffic is transmitted across can be easily saturated. Administrators of large networks typically apply simple techniques (like random sampling) to choose which statistics to collect [40]. Streaming database systems have much the same problem, where the need to shed load [10] and drop or aggregate historical data [52] has been noted.

In addition to the challenges presented by limited resources, data from real world environments is often noisy, lossy, and hard to interpret. This noise and uncertainty can be misleading, particularly when the user is summarizing and aggregating data using a high-level language like SQL. For example, the California Department of Transportation maintains a database of current road speeds from about 10,000 traffic sensors on California highways [9]. On a recent visit to their website, 60% of sensors were missing data. Such loss could cause users' queries to pick congested routes if sensors on those routes happen to be offline. If the query system could instead *infer* that missing speeds along certain routes are likely to be slow based on past behavior or speeds from online sensors, query results would be much more likely to reflect reality.

Besides failures, real-world networks often produce data

---

[*] Work done while the authors were visiting Intel Research Berkeley.

Proceedings of the 2005 CIDR Conference

that is simply wrong. For example, in a sensor network deployment on Great Duck Island (off the coast of Maine) [63, 48], researchers noted that about 40% of the sensors produced erratic temperature and humidity readings at some point; though such readings sometimes precipitated node failure, in other cases nodes otherwise continued to function normally. If the data acquisition system could detect and filter such outliers, it could inform a user of the failure and conserve bandwidth being used to transmit bad readings.

We address all of these problems by building a *model* of the world as data is collected from it. This model allows us to capture the correlations and statistical relationships between attributes collected by devices. We focus on *probabilistic* models, where the value of each attribute (*e.g.*, temperature, light) is a probability distribution that reflects the most likely value of that attribute, possibly depending on the values of other attributes (their *dependents*), such as the time of day or behavior of another node in the network. Such dependencies, or *correlations*, can be exploited to efficiently answer queries and enable new query types that explore the relationships between attributes. Models are built by periodically *observing* values of one or more attributes (*e.g.*, by acquiring a reading from a sensor) and using those observations to adjust the probability distributions of the observed attributes and their dependees. Models offer three distinct benefits:

1. They make querying **more efficient**. By exploiting correlations between attributes, it is often possible to use observations of a small number of attributes to provide approximations of the values of a large number of attributes. For example, if several temperature sensors in a building read approximately the same temperature day after day, a good (though perhaps not 100% accurate) guess after observing one sensor would be that all of the other sensors have about the same value.
2. They allow the database system to provide **probabilistic guarantees on the correctness of answers.** Unlike existing database systems, which provide the illusion of precise answers, even when data is missing or nodes are faulty, probabilistic models provide probabilistic guarantees on answers, telling the user the probability that a particular attribute value differs by more than some ϵ from the reported value based on past observations or known values of other, correlated attributes.
3. They allow the database system to answer **new types of queries**. For example, a model can detect certain very unlikely values (again, by observing past correlations with other sensors) and flag them as potential *outliers*. Similarly, a model can reveal relationships between devices that indicate, for example, that a particular sensor is redundant or that a pair of network links are in no way independent of each other. Finally, a model can often *predict* the value of a particular attribute as some point of time in the past or future.

In this paper, we briefly summarize one model, called BBQ [22] which we have studied in detail to provide efficient query answers in sensor networks. We then show how our ideas can be generalized to provide the other advantages described above (*e.g.*, various kinds of probabilistic guarantees and support for new types of queries) in a variety of domains and applications beyond sensor networks. We argue that any resource limited environment can benefit from our techniques.

We also show how to adapt a range of techniques, based on ideas from the machine learning and data mining communities, that allow us to improve the predictive power of models, represent correlations more compactly, and select and train models that are most appropriate for the data being modeled. Though such techniques sometimes are directly transferable from these other domains, they often require significant re-tooling to deal with limited resources, data acquisition issues, and to enable integration into a SQL-based database system.

2 Background

In this section, we summarize the basics of probabilistic models and show how they can be used to answer queries. We also summarize our previous work on the BBQ system, which is an example of a probabilistic model tuned to efficiently collect data from a sensor network.

2.1 Probabilistic models

We denote a model as a *probability density function* (pdf), $p(X_1, X_2, \ldots, X_n)$, assigning a probability for each possible assignment to the attributes $X_1, \ldots, X_n$, where each X_i is an attribute at a particular sensor (*e.g.*, temperature on sensor number 5, bandwidth on link A-B). This model can also incorporate *hidden variables* (*i.e.*, variables that are not directly observable) that indicate, for example, whether a sensor is giving faulty values or a node is subject to a denial of service attack. Such models can be learned from historical data using standard algorithms (*e.g.*, [50]).

Answering queries probabilistically based on a pdf is conceptually straightforward. Suppose, for example, that a query asks for an approximation to the value of a set of attributes to within $\pm\epsilon$ of the true value of each attribute, with confidence (*i.e.*, probability of being correct) at least $1 - \delta$. Using standard probability theory, we can use this pdf to compute the expected value, μ_i, of each attribute in the query. These will be our reported values. We can then use the pdf again to compute the probability that X_i is within ϵ from the mean, $P(X_i \in [\mu_i - \epsilon, \mu_i + \epsilon])$. If all of these probabilities meet or exceed user specified confidence threshold, then the requested readings can be directly reported as the means μ_i. If the model's confidence is too low, then we require additional readings before answering the query.

Choosing which readings to observe at this point is an optimization problem: the goal is to pick the best set of attributes to observe, minimizing the cost of observation required to bring the model's confidence up to the user specified threshold for all of the query predicates.

We can use the same technique to compute the expected sum or average of several attributes (*e.g.*, temperature on k different sensors) by exploiting linearity of expectation, which says $E(A_1 + \ldots + A_k) = E(A_1) + \ldots + E(A_k)$ and using the standard expression for the variance(σ) of a sum to compute our ϵ, δ bound, *i.e.*, $\sigma(A_1 + \ldots + A_k) = \sum_{i=1}^{k} \sigma(A_i) + \sum_{i=1}^{k} \sum_{j=1}^{k} cov(A_i, A_j)$. We can also compute a confidence that a particular boolean predicate (*e.g.*, temp > 25) is true by integrating over area of the pdf representing the region where the predicate is satisfied.

2.2 Example: Gaussians

In this section, we describe the time-varying multivariate Gaussians as a type of model. This is the basic model used in BBQ [21], and we summarize it here to provide a concrete example of one kind of model. A multivariate Gaussian (hereafter, just Gaussian) is the natural extension of the familiar unidimensional normal probability density function (pdf), known as the “bell curve”. Just as with its 1-dimensional counterpart, a Gaussian pdf over d attributes, $X_1, \ldots, X_d$ can be expressed as a function of two parameters: a length-d vector of means, μ, and a $d \times d$ matrix of covariances, Σ. Figure 1(A) shows a three-dimensional rendering of a Gaussian over two attributes, X_1 and X_2; the z axis represents the *joint density* that $X_2 = x$ and $X_1 = y$. Figure 1(B) shows a contour plot representation of the same Gaussian, where each circle represents a probability density contour (corresponding to the height of the plot in (A)).

Intuitively, μ is the point at the center of this probability distribution, and Σ represents the spread of the distribution. The ith element along the diagonal of Σ is simply the variance of X_i. Each off-diagonal element $\Sigma[i, j], i \neq j$ represents the covariance between attributes X_i and X_j. Covariance is a measure of correlation between a pair of attributes. A high absolute covariance means that the attributes are strongly correlated: knowledge of one closely constrains the value of the other. The Gaussians shown in Figure 1(A) and (B) have a high covariance between X_1 and X_2. Notice that the contours are elliptical such that knowledge of one variable constrains the value of the other to a narrow probability band.

We can use historical data to construct the initial representation of this pdf p. This historical data is typically collected as a part of a short observation phase using data extraction tools (in the case of our sensornet deployments, we have typically used a simple selection query in TinyDB [47]). Once our initial p is constructed, we can answer queries using the model, updating it as new observations are obtained from the sensor network, and as time passes. We explain the details of how updates are done in Section 2.2.2, but illustrate it graphically with our 2-dimensional Gaussian in Figures 1(B) - 1(D). Suppose that we have an initial Gaussian shown in Figure 1(B) and we choose to observe the variable X_1; given the resulting single value of $X_1 = x$, the points along the line $\{(x, X_2) \mid \forall X_2 \in [-\infty, \infty]\}$ conveniently form an (unnormalized) one-dimensional Gaussian. After re-normalizing these points (to make the area under the curve equal 1.0), we can derive a new pdf representing $p(X_2 \mid X_1 = x)$, which is shown in 1(C). Note that the mean of X_2 given the value of X_1 is not the same as the prior mean of X_2 in 1(B). Then, after some time has passed, our belief about X_1's value will be “spread out”, and we will again have a Gaussian over two attributes, although the mean and variance may have shifted from their initial values, as in Figure 1(D).

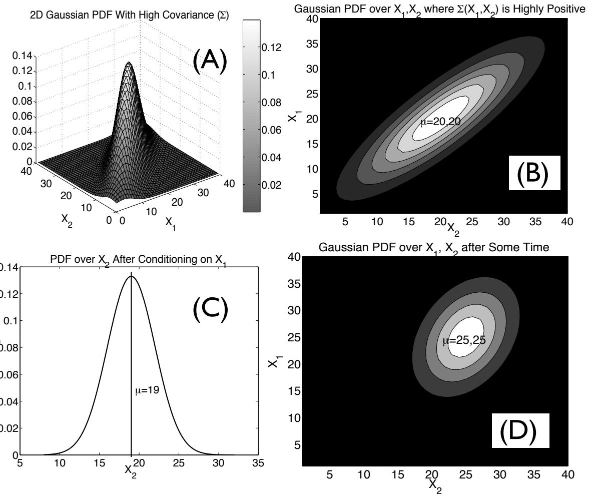


Figure 1: *Example of Gaussians: (a) 3D plot of a 2D Gaussian with high covariance; (b) the same Gaussian viewed as a contour plot; (c) the resulting Gaussian over X_2 after a particular value of X_1 has been observed; finally, (d) shows how, as uncertainty about X_1 increases from the time we last observed it, we again have a 2D Gaussian with a lower variance and shifted mean.*

Of course, this is but one example of many different types of models that could be used. Our basic approach can be generalized to various different models that may be more suitable in different environments and for different classes of queries. We will revisit this issue in Section 5. In the next two sections we look briefly at some of the technical details involved in creating and maintaining the Gaussian model used in BBQ.

2.2.1 Learning the model

Typically, probabilistic models are learned from some set of training data. In BBQ, this training data consisted of readings from all of the monitored attributes over some period of time. For example, with a Gaussian model, initial means and covariances can be computed from training data using standard statistical algorithms. Thus, for the specific model used in BBQ, we need to capture training data for some period of time before we can begin predicting values or exploiting correlations to avoid unneeded acquisitions. We are exploring techniques for interleaving model construction and query processing when possible, as described in Section 6 below.

2.2.2 Updating the model

Thus far, the model we have described represents *spatial* correlation in a network deployment. However, many real-world systems include attributes that evolve over time. For example, in a sensor network deployment in our lab, we noted that the temperatures have both temporal and spatial correlations [19]. Thus, the temperature values observed earlier in time should help us estimate the temperature later in time. A *dynamic probabilistic model* can represent such temporal correlations by describing the evolution of this system over time, telling us how to compute $p(X_1^{t+1}, \ldots, X_n^{t+1} \mid \mathbf{o}^{1\ldots t})$ from $p(X_1^t, \ldots, X_n^t \mid \mathbf{o}^{1\ldots t})$, where $\mathbf{o}^{1\ldots t}$ is the set of observations made over the network up to time t.

One common dynamic model is a *Markovian* model, where given the value of *all* attributes at time t, the value of the attributes at time $t+1$ are independent of those for any time earlier than t. This assumption leads to a simple model for a dynamic system where the dynamics are summarized by a conditional density called the *transition model*, $p(X_1^{t+1}, \ldots, X_n^{t+1} \mid X_1^t, \ldots, X_n^t)$. Using a transition model, we can compute $p(X_1^{t+1}, \ldots, X_n^{t+1} \mid \mathbf{o}^{1\ldots t})$ using the standard probabilistic technique of *marginalization* by integrating the transition model over the attribute values at time t.

This approach assumes the transition model is the same for all times t. Often, this is not the case – for example, in an outdoor environment, in the mornings temperatures tend to increase, while at night they tend to decrease. This suggests that the transition model should be different at different times of the day. One way to address this problem is by learning a different transition model $p^i(\mathbf{X}^{t+1} \mid \mathbf{X}^t)$ for each hour i of the day. At a particular time t, we simply use the transition model $mod(t, 24)$. This idea can, of course, be generalized to other cyclic variations.

Once we have obtained $p(X_1^{t+1}, \ldots, X_n^{t+1} \mid \mathbf{o}^{1\ldots t})$, the prior pdf for time $t+1$, we can again incorporate the measurements $\mathbf{o}^{t+1}$ made at time $t+1$ obtaining $p(X_1^{t+1}, \ldots, X_n^{t+1} \mid \mathbf{o}^{1\ldots t+1})$, the posterior distribution at time $t+1$ given all measurements made up to time $t+1$. This process is then repeated for time $t+2$, and so on. The pdf for the initial time $t = 0$, $p(X_1^0, \ldots, X_n^0)$, is initialized with the prior distribution for attributes $X_1, \ldots, X_n$.

2.3 Architecture

Given this basic structure for models, we show how they fit into a probabilistic query answering architecture. Parts of this architecture were laid out in our work on BBQ [22], though we have extended the architecture here to support several new kinds of queries as described in Section 4 below. One of our specific goals is for our architecture to be model-agnostic, *i.e.*, as long as a new model conforms to a basic interface, it requires no changes to the query processor and can reuse the code that interfaces with and acquires particular tuples.

Figure 2 illustrates our basic architecture through an example of a probabilistic model running over a sensor network. For other environments that involve data acquisition (as we note in Section 3 below), this basic architecture applies unchanged, with the main difference being the data acquisition mechanism. In non-acquisitional environments, models can still play an important role, as we note in Section 3.

Users submit queries to the database as in a traditional database, though we allow some unusual types of queries (see Section 4). One such class of queries is standard SQL queries augmented with error tolerances and target confidence bounds that specify how much uncertainty the user is willing to tolerate; such bounds will be intuitive to many scientific and technical users, as they are the same as the confidence bounds used for reporting results in most scientific fields (c.f., the graph shown in the upper right of Figure 2), though we are also exploring techniques, such as visualization, to allow the layperson to interpret query results.

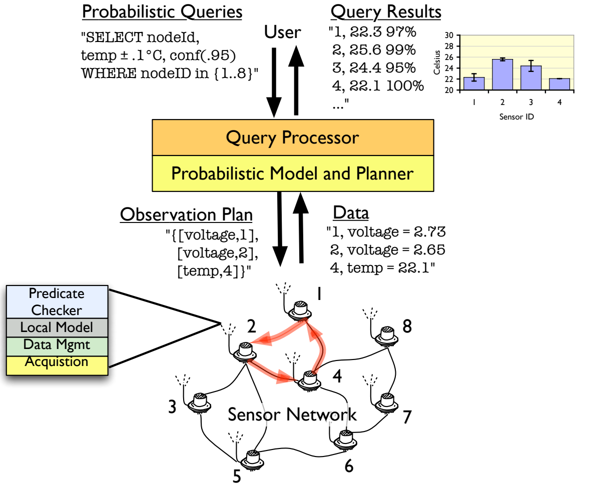


Figure 2: *Our architecture for model-based querying, shown as an example running on top of a sensor network.*

In this example, the user is interested in estimates of the value of sensor readings for nodes numbered 1 through 8, within .1 degrees C of the actual temperature reading with 95% confidence. After consulting the model, the system realizes that the model is not sufficiently accurate to answer the query with the specified confidence, and it decides that the most efficient way to achieve that confidence level is to read battery voltage from sensors 1 and 2 and temperature from sensor 4. Based on knowledge of the sensor network topology, it generates an *observation plan* that specifies how to acquire those samples (*e.g.*, which route to use to visit the relevant sensors), and sends the plan into the network, where the appropriate readings are collected. These readings are used to update the model, which can then be used to generate query answers with specified confidence intervals.

Notice that the model in this example chooses to observe the voltage at some nodes despite the fact that the user's query was over temperature. This happens for two reasons:

1. **Correlations in value:** Battery voltage and temperature often vary together, since batteries are somewhat higher voltage at warmer temperatures. For many types of batteries (such at the lithium-ion cells used in many mote deployments), this effect is quite pronounced (*e.g.*, we observe about 1% variation per degree on motes). Local variations in voltage are much more likely to be due to temperature fluctuations than decreased capacity, since if battery voltage drops at all as a battery's storage dwindles, it will vary over a much longer time scale.
2. **Cost differential:** Depending on the specific type of temperature sensor used, it may be much cheaper to

sample the voltage than to read the temperature. For example, on sensor boards from Crossbow Corporation for Berkeley Motes [15], the temperature sensor requires orders of magnitude more energy to sample than simply reading battery voltage. A primary goal of our work is to use models to help decide which sensors are significant and worth acquiring, given differential data acquisitional costs and the user's data demands (as specified in queries).

Thus, one of the key properties of many probabilistic models is that they can capture correlations between different attributes.

In general, the software that runs on each of the nodes in the network (shown in the small box on the bottom-left of Figure 2) includes some code to facilitate model-based query execution. The *predicate checker* is in charge of applying probabilistic predicates to determine if a particular query answer is worth transmitting – this is needed to help execute continuous queries that are looking for outliers or other exceptional conditions. It executes against a local image of the model which captures the state and behavior of the local node and its relationship to other nodes. The *data management layer* is in charge of managing typed tuples of data, which it builds up by calling down into the *acquisition layer*. Note that, in the example described here, the predicate checker and local model are not needed, because (in this case) the model is stored centrally. In general, centralized models make query planning easier since they have access to state in a single location, but are more expensive (in terms of communication or energy), because they must collect that state to a single location rather storing it locally at the nodes.

There are thus four major steps to query processing in our architecture:

1. Using the model, the query optimizer generates an observation plan which will allow it to answer the query to within the specified bounds at a minimal cost.
2. The plan is executed by the network, collecting data from relevant nodes (and possibly filtering out some results by consulting an in-network version of the model).
3. The model is updated with results collected from the network.
4. Using basic probability computations (Section 2.1), the query answer and confidence bounds are computed.

We note that he user in Figure 2 could have requested 100% confidence and no error tolerance, in which case the model would have required us to interrogate every sensor. Conversely, the user could have requested very wide confidence bounds, in which case the model may have been able to answer the query without acquiring any additional data from the network.

Given this basic introduction to our architecture, we now turn our attention to some of the ways in which our techniques can be applied.

3 Applications

Systems that exploit statistical modeling techniques and optimize the utilization of a network of resource constrained devices, such as BBQ, could have significant impact in a number of areas, as outlined by some case studies described in this section. Although our architecture is targeted primarily at acquisitional environments, some of the systems we discuss do not fall into this category (e.g., database cost estimation) and can still benefit from our core probabilistic modeling technology.

3.1 Sensor applications

We begin with several sensor-network applications:

Building control: Sensor networks have a number of applications in control and automation in buildings. For example, rather than monitoring temperatures at just a few points in a building, as is done in most HVAC systems today, the sensor network can monitor temperatures throughout the building, and regulate more effectively the power generation and output of heating and air conditioning systems [44]. Battery powered sensors are desirable because they can be deployed much more cheaply in existing building infrastructures. However, for batteries to be cost effective, they must last a fairly long time. Our modeling techniques make it possible to capture information that could be used in such a building control environment (with bounds on the error and probability of exceeding that error) while visiting a small number of nodes, thus, significantly extending the lifetime of the network.

Sensor failure detection: In long-term environmental sensing deployments, sensors are known to be failure prone [63]; in many cases these failures are "Byzantine" – that is, nodes do not stop, but rather simply produce erroneous output. Such failures may show up as outlier values, or, more generally, generate sensor readings that follow unexpected patterns. Thus our outlier detection queries should be able to detect them. Here, probabilistic models and statistical techniques provide the basis for detecting such unexpected patterns. Using fault injection techniques, and by studying failures from previous deployments, we can build alerting tools that can detect failed and misbehaving sensors.

Highway traffic monitoring and optimization: As we noted in the introduction, traffic sensor data (as currently made available by the California Department of Transportation [9]), consists of data from hundreds or thousands of sensors (typically, these are metal loops embedded in the freeway that use inductance to record as cars pass over them). Based on our studies over several days of the data from these web sites, it appears as though such sensors are often offline – Figure 3 shows a screenshot from a CalTrans Java applet (`http://www.dot.ca.gov/traffic/`) looking at the San Francisco area during rush hour on 8/4/04. In this case, the larger circles to the left and right of the roadways represent the speeds on the two sides of the freeway; green and yellow circles (the lightest in color, when viewed in grayscale) represent speeds above 15 MPH, whereas red (dark) circles represent slower speeds. Gray (intermediate darkness) circles with black dots in the middle represent offline sensors. Notice that a few sensors at the west edge of the Bay Bridge indicate traffic there is slow, but that many sensors around it are offline. It is not clear if travellers

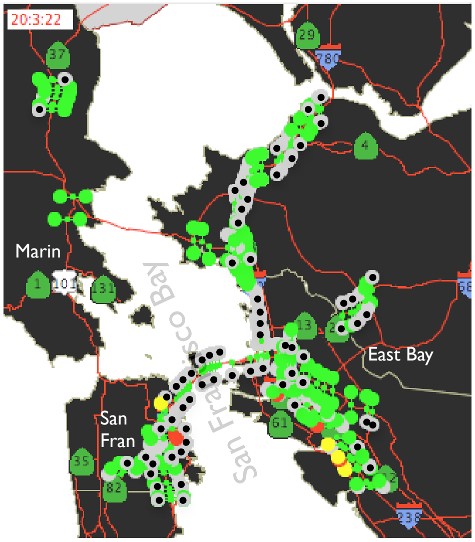


Figure 3: *Screenshot from the California Department of Transportation road sensor website in the Bay Area. Green dots represent roads where the traffic is travelling faster than 45 MPH; yellow represents traffic moving 15-45 MPH, and red represents traffic moving at speeds less than 15 MPH. Gray circles with black dots (added for clarity) represent offline sensors.*

should avoid the Bridge, or if this is a localized anomaly that will not cause long delays. Feeding such data to a route planning algorithm is likely to cause it to do very strange things if it tries to apply linear interpolation or other simple techniques to guess traffic speeds. In contrast, a probabilistic model can use data from times when the sensors were online, combined with the data from a few of the nodes, to infer the missing speeds.

Structural and factory health monitoring: A popular application for sensor networks is *preventative maintenance* [37], where structures and industrial equipment are monitored for early signs of failure. A widely used technique for failure detection involves measuring changes in the phase between vibration signals from groups of sensors – the intuition being that if two parts of a piece of equipment are solidly connected, they will vibrate in-phase, but if they suddenly become out-of-phase with each other, that is a sign that something is wrong. Probabilistic models provide a convenient way to determine the components that are expected to vibrate in-phase with each other, and outlier detection techniques like those used for sensor failure detection can identify low-probability changes in the phase structure, indicating the possibility of impending failure.

Intrusion detection and tampering: As a part of an involvement in MIT's new Center for Information Security and Privacy (CISP) [49], we are investigating techniques for intrusion detection in wireless and sensor networks. In sensor networks, there are a range of physical attacks that involve tampering with devices or sensors. Examples include intruders seeking to hide information about their presence or trying to cause a control or regulatory system to misbehave (*e.g.*, people often 'hack' computers in their cars to increase performance, possibly decreasing safety and increasing emissions). Outlier and influence queries have potential application in detecting this sort of tampering.

3.2 Non-sensor applications

There are also a wide range of non-sensor applications that can benefit from our probabilistic model-based approach.

Network monitoring: Network monitoring, even in wired networks, has the potential to consume a significant proportion of available bandwidth. For example, on a typical edge gateway in a large university, per-flow statistics are collected to identify users and applications that are potential security concerns or who are over-utilizing the network. Such statistics constitutes tens of MB/sec of data, and, even on a well-provisioned inter-university network, collecting a complete set of such statistics exhausts the CPU and bandwidth capabilities of edge routers [40]. Current practice is to randomly sample a subset of flows and store just the sample. Similarly, in wireless networks, the collection of time-varying link quality and congestion information can impose a significant overhead, especially in dynamic networks where such information may change rapidly, requiring frequent link-sampling. We can use probabilistic modeling techniques to estimate and track loss rates, congestion information and security concerns (*e.g.*, types of flows that are likely to use unusual amounts of bandwidth or are otherwise outliers), exploiting correlations to avoid acquiring data that can be inferred from a well-chosen subset of available readings.

Database summaries: Capturing the joint data distribution of multi-dimensional data sets through compact and accurate *synopses* is a fundamental problem arising in a variety of practical scenarios, including *query optimization*, *query profiling*, and *approximate query answering*. Cost-based query optimizers employ such synopses to obtain accurate estimates of intermediate result sizes that are, in turn, needed to evaluate the quality of different execution plans. Similarly, query profilers and approximate query processors require compact data synopses in order to provide users with fast, useful feedback on their original query [11, 59]. Such query feedback (typically, in the form of an *approximate answer*) allows OLAP and data-mining users to identify the truly interesting regions of a data set and, thus, focus their explorations quickly and effectively, without consuming inordinate amounts of valuable system resources. Further, users can make informed decisions on whether they would like to invest more time and resources to fully executing their queries.

The idea of using probabilistic modeling techniques to build synopses has already been explored [18, 27]. As this previous work shows, using probabilistic models to capture and exploit the correlations in the data can lead to significantly more compact summaries. The techniques we have developed can be directly applied in this context as well; in particular we are interested in answering more complex queries as well as in providing probabilistic guarantees to the user.

Load shedding in streams: Load-shedding is cited as a requirement in many stream-based query processors [52, 10]. The Aurora [10] project proposes *semantic load shedding*, where input tuples that correspond to particular output values are considered more important than other tuples (and are thus not shed). The authors of Aurora propose a scheme where the query plan is “reversed” to determine such input-output mappings, but for general query plans, such an approach is infeasible, since operations like joins and aggregates are not readily invertible. As a more tractable alternative, we can use probabilistic models to determine the relationship between inputs and outputs, keeping input tuples that have a high probability of mapping to valued outputs. These probabilistic relationships may include correlations between different fields in the input tuples, so that, for example, the model may determine that intermediate join tuples have a low probability of producing a high-value output, even though the base tuples of the join both had a high value prior to the join.

Monitoring distributed streams: Recently there has also been an increasing interest in distributed data streams, *i.e.*, data streams that originate and are processed in a distributed fashion [23, 13]. Though similar to sensor networks in many aspects, the optimization goal in such systems is network latency, not the battery life of the sensors. The IrisNet project [23] proposes use of caching to reduce the latencies incurred in query answering. We believe a model-based approach can lead to both better answer quality, and a reduction in latencies, especially in applications such as the motivating *parking space finder* application of IrisNet.

4 New queries

These applications require a range of new queries that non-probabilistic database systems are ill-equipped to answer. In this section, we summarize the range of new queries that we are working to support.

Probabilistic, approximate queries: The most basic class of queries that we anticipate users to ask are probabilistic and approximate variations of traditional SQL queries. Examples of such queries include queries asking for *temperature* at a certain location in a building, or *average speed* along a segment of a highway. We can support such queries by using additional predicates in SQL expressions that specify the confidence that the user wants in the answer, or the error she is willing to tolerate. This class of queries covers traditional *exact* queries which can be asked by setting the confidence required to 100%.

Our initial effort in BBQ provided support for this type of query; Figure 4 shows one advantage of approximate queries: improved performance. In this case, we ran three range queries over temperature readings from a 11 node sensor deployment in the Berkeley Botanical Garden. We trained our model for 20 days and ran test queries over a 10 day period. We used pre-collected data so we could verify the accuracy of our approximate query answers during this test period. If we had asked an exact query, we would have been required to observe the value of every sensor at each point in time; using our Gaussian-based probabilistic model with queries specifying 95% confidence, we were required to observe the values of only a small fraction of the sensors. The truth values of the predicates on the unobserved sensors could be accurately predicted by exploiting cross-sensor correlations (in all cases, we had less than the 5% allowed errors when we compared the predicted predicate values to the actual values from the test data). Notice that different predicates require observation of different numbers of sensors at different times of day – this is because of the natural temperature distributions in the garden that our model is able to exploit. For example, during the day, the temperatures are typically significantly higher than the top end of the range specified in first predicate (16-17 degrees). Because of this, during the day, very few observations need to be made to ascertain that the predicate is false with sufficient confidence.

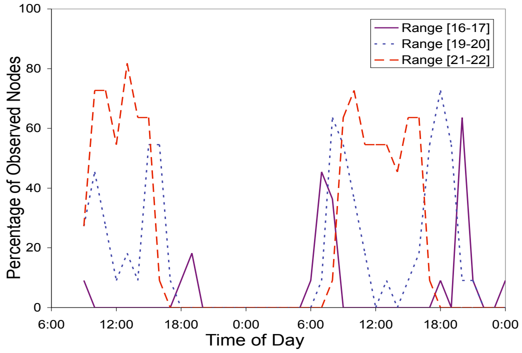


Figure 4: *Percentage of observed sensors versus time of day for a 36 hour period over 11 sensors deployed in the Berkeley Botanical Garden for three different range predicate queries. In this case, we set* $\delta = 5\%$ *(95% confidence).*

Outlier queries: Outliers are essentially events of low probability, and use of probabilistic models provides an excellent mechanism to detect outliers. To detect outliers, the user could ask the system to report whenever any attribute value occurs that has a low user-specified probability of occurrence, or that differs from its expected value by more than some threshold. As an example, a user might register a continuous query that reports any time the bandwidth on their wireless network is more than three times the expected value for the current time of day. Note that, outlier detection will typically require continuous observation of the underlying attributes at the motes, and the main advantage of using models in this case would be to save communication cost (though knowledge of how often, and under what circumstances outliers are expected to occur may be used to reduce the observation costs as well).

Prediction queries: These queries estimate the value of an attribute or predicate either (1) at a location where there are currently no available devices, or (2) at some time in near future, with the best precision (and report that precision and confidence in the estimate). For example, a user might ask: “What will the temperature in Room 938 be in 10 minutes?” Similarly, users might post “what-if” queries, to discover how a change in one attribute might affect other attributes – for example, in a system monitoring application, a user might ask how increasing bandwidth on a given link would increase CPU utilization on a given processor.

Queries over hidden variables: In many scenarios, there may be interesting variables that cannot be directly observed, but that can be reasoned about. For example, in a sensor network that has sensors for monitoring *temperature*, *pressure* and *light*, but not for monitoring *rain*, we can never "observe" rain directly, but it may be possible to infer whether it is raining or not based on the values of the observable variables. There has been much work on hypothesizing about hidden variables (in the above example, we knew beforehand about existence of an unobservable variable; there may be cases where we have first infer that a hidden variable exists), and learning structures containing them [25]. We expect to be able to leverage these existing techniques in our work.

Influence queries: Use of probabilistic models also opens up avenues for asking sophisticated analysis queries. One such class of queries are *influence queries*, where the user might want to know which attributes are most closely correlated with the value of a particular attribute. Such queries can be used to help infer causality or determine when sensors in an area are redundant. For example, a user might ask the question: "What percentage of the traffic on link i is predicted by the traffic on links j and k over all time?"

5 Challenges

Given these applications and queries, we now discuss some of the challenges they present, followed by a set of techniques we are exploring to address these challenges.

Model selection: Choosing the best model for the given query workload and environment is a key issue. The choice of model affects many aspects of our approach:

- *Accuracy of the answers:* Recall that we provide probabilistic answers to the user, and the confidence in the answers provided relies on the assumption that the underlying data follows the model with sufficient accuracy. If this is not the case, the answers provided by our models could be erroneous.
- *Ability to answer certain types of queries:* Some models are more naturally suited to answer certain types of queries. For example, outlier detection requires the system to continuously sample sensors and check them against the model to see if they have a low probability of occurring. To do this efficiently, we may require a model that is distributed across the nodes in a system.
- *Algorithmic aspects of querying:* The techniques to query the model efficiently are highly dependent on the choice of the model. The space and time requirements of different models can vary by orders of magnitude.

Selecting a suitable model for the data is one of the critical challenges in the deployment of model-based systems.

Transparency in model selection and usage: Although different models may be better suited for different environments and for different classes of queries, developing a completely new system for each different model may be a waste of time and development effort. Ideally, using a new model should involve little to no effort on the part of user. Given a large variety of models that may be applicable in various different scenarios, this may turn out to be a tremendous challenge.

Data acquisition: Irrespective of the model selected, when and how to acquire data is one of the key issues that needs to be addressed. In most of the scenarios that we envision a model-based system being deployed, the cost of acquiring and/or transferring data is the dominant cost. For example, in sensor networks, both the cost of sampling data, and the cost of communicating it to the basestation are high – for example, in a recent analytical study, we estimated that 98% of energy in a typical sensor network data collection scenario was consumed sampling sensors or communicating. In distributed system where the data is being generated all over the world, minimizing the *latency* in answering the query could be the optimization goal. There are two aspects to this problem:

- *When* to acquire data: This is partly dependent on the model and the user query. To maintain the required confidence in the answers it provides, the model could ask for more samples of the underlying data. In many cases, we expect that the same confidence may be achieved in many different ways, *i.e.*, by sampling different sets of attributes of the data. Because of this, the question of *when* to acquire data will typically be tightly integrated with the question of *how* it is acquired.
- *How* to acquire data: Most of the environments we have discussed so far exhibit highly non-uniform cost structures. For example, in sensor networks, the costs of sampling different attributes can be wildly different. Also, the multi-hop nature of communication in sensor networks means that sampling sensors closer to the base station is cheaper than sampling far away sensors.

Issues surrounding when and how data is collected are amongst of the most interesting algorithmic challenges in the development and deployment of model-based systems.

Training and retraining: In general, a probabilistic model is only as good at prediction as the data used to train it. For models to perform accurate predictions they must be trained in the kind of environment where they will be used. That does not mean, however, that well-trained models cannot deal with changing relationships over time; for example, the model we used in BBQ[21] uses different correlation data depending on time of day. Extending it to handle seasonal variations, for example, is a straightforward extension of the techniques we use for handling variations across hours of the day. Typically in probabilistic modeling, we pick a class of models, and use learning techniques to pick the best model in the class. The problem of selecting the right model class has been widely studied (*e.g.*, [50]), but can be difficult in some applications.

In Section 6, we outline a more general Bayesian approach that integrates querying of the data with learning.

Data model and query language: Our initial efforts have focused on building simple Gaussian models and demonstrating that they can answer certain classes of queries [21]. However, we do not have an integrated acquisition-oriented database system that includes notions of uncertainty or modeling, and it is not clear how such models can be integrated in a general way into the existing relational data model and query languages. One possible data representation is to attach a probability distribution to each data point – several proposals for probabilistic data models of this type have been made in the literature [42, 3, 24, 27], and we may be able to adapt this existing work. None of these approaches, however, focus on the data acquisition or model-learning issues. Instead, they concentrate representing user-specified uncertainty in the database; the approaches do not help us address our principal challenge of acquiring appropriate data to answer user queries at the desired confidences.

As an initial step towards supporting uncertainty in our query language, we currently represent ϵ, δ bounds as additional query predicates in SQL expressions, as in the query shown in Figure 2 above, but there are a number of outstanding questions about how a system deals with readings with differing levels of uncertainty. For example, how can readings with different uncertainties from different sub-queries be composed into a final query result? What should we do if our models cannot answer queries to a given confidence level? Are there other representations besides confidence that we should consider (*e.g.*, absolute or relative deltas)?

Exposing uncertainty to the user: One issue with exposing uncertainty to the user is that it requires him or her to understand the basics of probability. Although this may be acceptable for scientific users who are used to statistical tests of significance and other confidence metrics, for the layperson, such notions will be quite confusing. One possibility is to convert probabilistic answers to definite answers when result confidence is above a give threshold, suppressing the uncertainty report. Though at first this appears to be no better than what traditional uncertainty-unaware database do, it is in fact substantially better as the user is never exposed to answers that do not have a high probability of being true. For example, in the case of the California Freeway sensors given in the introduction, users wouldn't receive average speeds for segments where the uncertainty was low, preventing them from inadvertently using congested routes.

The other possibility we are exploring is to avoid these concerns through the use of visualization tools. Figure 5 shows a visualization mockup of uncertainty in readings from a single sensor. It shows a stream of temperature readings from a single sensor; the small circles represent points of time when readings are actually captured. The contours represent different levels of uncertainty in readings. The darkest, narrowest band corresponds to a band of confidence about the most probable value of the sensor – in this case, there is a 90% chance the true value of the sensor is in this band. The lighter, outermost band captures the range of readings where the true value lies with 99% probability. This visualization could be built using a probabilistic model such as our Gaussian model and provides an intuitive representation of uncertainty.

Figure 6 shows a second visualization of uncertainty information for a query that collects readings from a set of geographically distributed sensors (in this case, sensors are shown on a portion of Nantucket Island). In this case, colors represent temperature estimates; large solid circles represent the locations of sensors. White circles are inactive sensors that were not involved in data collection. Densely colored regions represent areas where there is high certainty on the reading, with sparsely colored regions having low certainty. These certainties can be derived, for example, from a probabilistic model that represents correlations between the active sensors and the inactive sensors. Such a visualization allows users to quickly determine where more sensors may be needed, and to understand how well sensors are monitoring an area of interest.

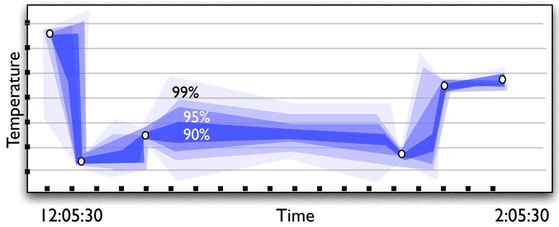


Figure 5: *Mockup visualization of a display used to visualize uncertainty in a stream of readings coming from a sensor.*

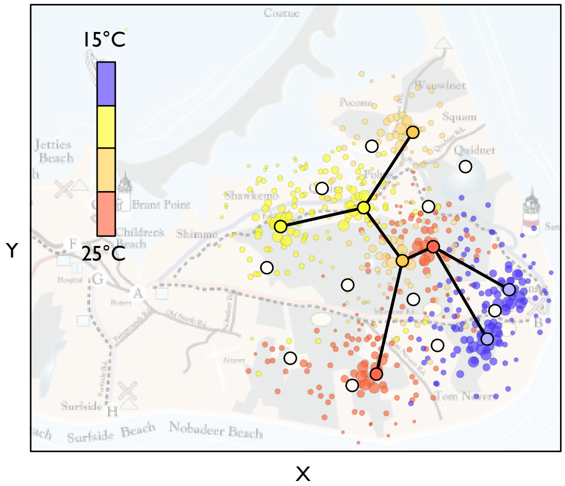


Figure 6: *Mockup visualization of uncertainty display of a set of temperature sensors (circles outlined in black) on Nantucket Island. Confidence in readings is represented by colored-dot density.*

We are planning to build support for both types of uncertainty visualization into our system.

Continuous vs. snapshot queries: We plan to support both snapshot queries, *i.e.*, one-time queries about the current state of the system, and continuous queries, *i.e.*, queries which the user wants to know the answers of on a periodic basis. Depending on the optimization goals, these two classes of queries pose different optimization challenges. The challenge with snapshot queries is to balance "push" vs "pull". Pushing too much data towards the user can lead to wasted communication; on the other hand, having the system pull data for every snapshot query could lead to unreasonable latencies. For a continuous query, we must figure out which nodes should stay active, when to do sampling and how to communicate the data from those nodes to the base station. Though it is possible to optimize the data acquisition process heavily, dynamic sensornet topologies can

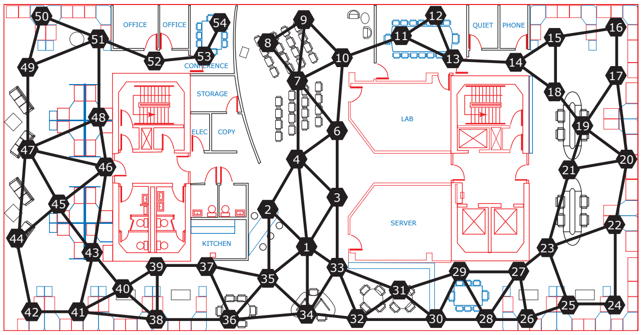


Figure 7: *Structure of graphical model learned for the temperature variables for a deployment in the Intel Berkeley Lab [56].*

complicate matters.

6 New techniques

In order to address the wide range of applications and new queries described in Sections 3 and 4, and to surmount the challenges in Section 5, it is insufficient to simply adapt existing methods in data bases, machine learning and distributed systems; we need new integrated approaches. This section outlines techniques that can address some of these issues as well as research directions that we are currently pursuing.

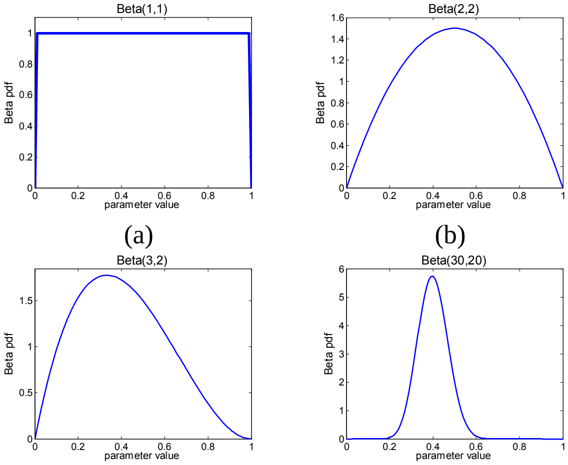
6.1 Representations for probability distributions

Probabilistic models are the centerpiece of our approach. In Section 2.1, we described very general probability distributions, $p(X_1, X_2, \dots, X_n)$. Choosing the appropriate representation for such distribution, allowing us to represent complex correlations compactly, to learn the parameters effectively, and to answer queries efficiently is one of the biggest challenges in this research. Probabilistic graphical models are a very appropriate choice to address these issues [57].

In a (probabilistic) graphical model, each node is associated with a random variable. Edges in the graph represent "direct correlation", or, more formally, conditional independencies in the probability distribution. Consider, for example, the sensor deployment shown in Figure 7, where the attributes are the temperatures in various locations in the Intel Berkeley Lab. The graphical model in Figure 7 assumes, for instance, that temperatures in the right side of the lab are independent of those in the left side, given temperatures in the center (*e.g.*, T_{20} and T_{47} are independent given T_{10}, T_{32}, and T_{33}).

The sparsity in graphical models is the key to efficient representation and probabilistic querying [14]. In discrete settings, for example, a naive representation of $p(X_1, X_2, \dots, X_n)$ is exponential in the number of attributes n, while a graphical model is linear in n and, in the worst case, exponential in the degree of each node. In addition to reducing space complexity, reducing the number of parameters can prevent overfitting when the model is learned from small data sets. Similarly, answering a query naively is exponential in n, while in a graphical model the complexity is linear in n and exponential in the tree-width of the graph.

In our setting, in addition to allowing us to answer queries efficiently, graphical models are associated with a wide range of learning algorithms [34]. These algorithms can be used both for learning a model from data, and to evaluate the current model, addressing many of the model selection issues discussed above.



(a) (b)

(c) (d)

Figure 8: *Bayesian approach for learning the parameter of a coin: (a) prior distribution, Beta(1,1); posterior distributions over the coin parameter after observing: (b) 1 head, 1 tail; (b) 2 heads, 1 tail; (b) 29 heads, 19 tails.*

Additionally, graphical models allow us to efficiently address hidden variables, both in terms of answering queries and of learning about hidden variables [25]. In the example in Figure 7, each node could be associated with a faulty sensor hidden variable [43]. When a node is faulty, the sensed value is, for example, independent of the true temperature. By exploiting correlations in the temperatures measured by the nodes and sparsity in the graphical model, we can efficiently answer outlier queries.

Finally, there is vast graphical models literature for addressing other types of queries and models. For example, these models can be extended to allow for efficient representation and inference in dynamical systems [8, 17], and to answer causal queries [58].

6.2 Integrating learning and querying

Thus far, we have focused on a two-phase approach: in the first phase, we learn the probabilistic model, and in the second, we use the model to answer queries. This is an artificial distinction, raising many questions, such as when should we stop learning and start answering queries. We can address this issue by applying a *Bayesian learning* approach [6].

In a Bayesian approach, we start with a *prior distribution* $p(\Theta)$ over the model parameters Θ. After observing some value for the attributes $\mathbf{x}$, we use Bayes rule to obtain a *posterior distribution* over the model parameters $p(\Theta \mid \mathbf{x})$:

$$p(\Theta \mid \mathbf{x}) \propto p(\mathbf{x} \mid \Theta) p(\Theta). \tag{1}$$

This process is repeated as new data is observed, updating the distribution over model parameters.

Consider, for example, the task of learning the parameter of a biased coin; that is, the coin flips are independently distributed according to the usual binomial distribution with unknown parameter. Typically, for efficiency reasons, we choose a prior distribution that yields a closed form representation of the posterior in Equation (1); when such closed-form solutions are possible, the prior $p(\Theta)$ and the likelihood function $p(\mathbf{x} \mid \Theta)$ are said to be *conjugate*. In our example, Beta distributions are conjugate to the binomial distribution of the coin. Figure 8 illustrates the process of Bayesian learning for our coin example: We start with the Beta(1,1) in Figure 8(a); here, the distribution over possible coin parameters is almost uniform. Figures 8(b)-(d) illustrate the posterior distribution over the coin parameters after successive observations. As more coin flips are observed, the distribution becomes more peaked. Thus, when answering a query about the coin after a few flips, our answer will be uncertain, but after making a larger number of observations, the answer will have significantly lower variance.

These ideas can be integrated with our approach to avoid the need for a separate learning phase. Consider the Gaussian distributions used in the BBQ system. Initially, we may be very uncertain about the mean and covariance matrix of this distribution, which can be represented by a highly uncertain prior (the conjugate prior for the covariance matrix is the Wishart distribution). Thus, when faced with a query, we will need to observe the value of many sensors. However, using Bayesian updating, after we observe these sensor values, in addition to answering the query at hand, we become more certain about the mean and covariance matrix of the model. Eventually, we will be certain about the model parameters, and the number of sensors that we will need to observe will automatically decrease. This integrated approach achieves two goals: first, the learning phase is completely eliminated; second, using simple extensions, we can add dynamics to the parameter values, allowing the model to change over time.

6.3 Long-term query plans

Modeling the correlations between different attributes in the system and also, the correlations across time, enables the query planner to consider a much richer class of execution plans than previously possible.

One such interesting class of execution plans that we have explored in our previous work [20] are *conditional plans*. These plans exploit the correlations present in the data by introducing low-cost predicates in the query execution plan that are used to change the ordering of the more expensive predicates in the query plan.

As an example, consider a query containing two predicates `temp` $> 20^o$ `C`, and `light` $<$ `100 Lux` over a sensor network. Let the *apriori* selectivities of these two predicates be $\frac{1}{2}$ and $\frac{1}{2}$ respectively, and let the costs of acquiring the attributes be equal to 1 unit each. In that case, either of the two plans a traditional query processor might choose has expected cost equal to 1.5 units (Figure 9). However, we might observe that the selectivities of these two predicates vary considerably depending on whether the query is being evaluated during the day or at night. For instance, in Berkeley, during Summer, the predicate on `temp` is very likely to be false during the night, whereas the predicate on `light` is very likely to be false during the day.

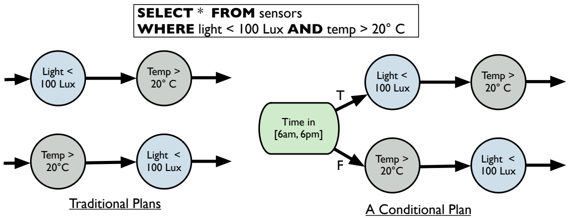


Figure 9: *A conditional query plan that uses different ordering of query predicates depending on the time of day.*

This observation can be utilized to construct a *conditional* plan as shown in the figure that checks the time of the day first, and evaluates the two query predicates in different order depending on the time. Assuming that the selectivity of the `temp` predicate is $\frac{1}{10}$ at night, and the selectivity of the `light` predicate is $\frac{1}{10}$ during day, the expected cost of this plan will be 1.1 units, a savings of almost 40%.

More generally, in continuous queries, additional cost savings can be obtained by exploiting similarities between queries. For example, if we know that the next query will require an attribute at a particular node i, and the current query plan observes values at nearby nodes, then it is probably better to visit node i as well in the current time step.

The optimal solution to such long-term planning problems can be formulated as a *Markov decision process* (MDP) [5, 60]. In an MDP, at each time step, we observe the current state of the system (in our setting, the current distribution and query), and choose an action (our observation plan); the next state is then chosen stochastically given the current state (our next query and distribution). Unfortunately, traditional approaches for solving MDPs are exponential in the number of attributes. Recently, new approximate approaches have been developed to solve very large MDPs by exploiting structure in problems represented by graphical models [7, 32]. Such approaches could be extended to address the long-term planning problem that arises in our setting.

6.4 In-network processing

Thus far, we have focused on algorithms where the probabilistic querying task occurs in a centralized fashion, and we seek to find efficient network traversal and data gathering techniques. However, in typical distributed systems, nodes also have computing capabilities. In such settings, we can obtain significant performance gains by pushing some of the processing into the network.

In some settings, we can reduce communication by aggregating information retrieved from the network [45, 35]. We could integrate these techniques with our models by conditioning on the value of the aggregate attributes rather than the sensor values. Such methods will, of course, increase our planning space: in addition to finding a path in the network for collecting the required sensor values, we must decide whether to aggregate values along the way.

More recently, a suite of efficient algorithms has been developed for robustly solving inference tasks in a distributed fashion [31, 56]. In these approaches, each node in the network obtains a local view of a global quantity. For example, each node computes the posterior probability over a subset of the attributes given the sensor measurements at all nodes [56]; or each node obtains a functional representation (*e.g.*, a curve fit) of the sensor (*e.g.*, temperature) field [31]. Given such distributed algorithms, we can push some of the probabilistic query processing into the network, allowing nodes to locally decide when to make observations and when to communicate. When integrated with a system like BBQ, these methods allow the user to connect to any node in the network, which can collaborate with the rest of the network to answer queries or detect faulty nodes.

7 Related work

There have been other model-based approaches for query answering that rely on a model-like abstraction [64, 55, 54, 39]. In most cases, the related work assumes a client-server relationship, where the model runs on the server and monitors the value of a number of attributes at the clients. These approaches maintain a bound or trajectory over each of the attribute values at the server, using that bound to predict the value. When the clients notice that their value no longer fits the model, or when the server has sufficient bandwidth or energy, it will directly observe the attribute values and update the model. Our approach differs from previous research in that it uses multidimensional probabilistic models that track the relationship between attributes in addition to the values of attributes themselves. These relationships, or correlations, allow the model to update its value estimates of many attributes when a single attribute is observed; models can be built across attributes that are directly observed (*e.g.*, light readings from sensors), as well as global attributes (*e.g.*, time of day), and derived attributes (*e.g.*, network loss rate).

There has been substantial work on approximate query processing in the database community, often using model-like *synopses* for query answering much as we rely on probabilistic models. For example, the AQUA project [30, 28, 29] proposes a number of sampling-based synopses that can provide approximate answers to a variety of queries using a fraction of the total data in a database. As in our approach, such answers typically include tight bounds on the correctness of answers. AQUA does not exploit correlations. A few recent papers [18, 27] propose exploiting data correlations through use of graphical model techniques for approximate query processing, but neither provide any guarantees on the answers returned. Recently, Considine *et al.* [41] and Gibbons *et al.* [53] have shown that sketch based approximation techniques can be applied in sensor networks to compute aggregates. Others have proposed approximation techniques for stream-query processing, *e.g.*, Das *et al.* [16] and Motwani *et al.* [51].

Approximate and best effort caches [55, 54], as well as systems for online-aggregation [61] and stream query processing [52, 10, 12] include some notion of answer quality and include the ability to discard some tuples. Most related work focuses on quality with respect to summaries, aggregates, or staleness of individual objects already stored in the system, as opposed to readings being actively acquired by the query processor.

Several proposals for probabilistic data models have been made in the literature. For example, ProbView[42] provides a data model based on discrete pdfs, and shows how to answer a number of queries in such a domain. Getoor [26] explores a number of probabilistic extensions to the relational model, and shows how statistical models can be learned from relations. Barbara *et al.* [3] present some initial results on probabilistic data models. Faradjian *et al.* [24] present a data model based on continuous pdfs. None of these approaches focus on the data acquisition issues, but rather on representing uncertainty in the database, and on types of query processing that can be applied to probabilistic attributes.

The probabilistic modeling techniques we describe are based on standard results in machine learning and statistics (*e.g.*, [62, 50, 14]). There are also a number of proposed techniques for outlier detection [4, 1, 2]. We believe, however, that our approach is one of the first architectures that combines model-based approximate query answering with query processing and optimization and an uncertainty-aware data model and query language.

8 Conclusions

The integration of database systems with probabilistic modeling will enable database systems to tolerate loss, detect faulty or erroneous inputs, and identify correlations that can be used to improve query performance while enabling a range of new types of queries. Such models are particularly useful in acquisitional settings such as sensornets and Internet monitoring where the data acquisition costs are typically very high; even for non-acquisitional applications, being able to deal with noisy and lossy data in a seamless manner, and the ability to model and reason about data correlations can prove to be tremendously useful. There are a number of architectural and algorithmic challenges associated with fully integrating these techniques into database systems and, although we believe we have taken some initial steps towards this end, we look forward to many years of fruitful cross-disciplinary research. We envision this research leading to significant improvements in the utility and efficiency of database systems at managing real-world data.

References

[1] C. C. Aggarwal and P. S. Yu. Outlier detection for high dimensional data. In *Proceedings of SIGMOD*, pages 37–46, 2001.

[2] A. Arning, R. Agrawal, and P. Raghavan. A linear method for deviation detection in large databases. In *Proceedings of KDD*, 1995.

[3] D. Barbara, H. Garcia-Molina, and D. Porter. The management of probabilistic data. *IEEE TKDE*, 4(5):487–502, 1992.

[4] V. Barnett and T. Lewis. *Outliers in Statistical Data*. John Wiley and Sons, New York, 1994.

[5] R. E. Bellman. *Dynamic Programming*. Princeton, 1957.

[6] J. Bernardo and A. Smith. *BAYESIAN THEORY*. Wiley, 1994.

[7] C. Boutilier, R. Dearden, and M. Goldszmidt. Exploiting structure in policy construction. In *Proc. IJCAI*, pages 1104–1111, 1995.

[8] X. Boyen and D. Koller. Tractable inference for complex stochastic processes. In *Proc. UAI*, 1998.

[9] California Department of Transportation. Caltrans realtime freeway speed map. Web Site. `http://www.dot.ca.gov/traffic/`.

[10] D. Carney, U. Centiemel, M. Cherniak, C. Convey, S. Lee, G. Seid-

man, M. Stonebraker, N. Tatbul, and S. Zdonik. Monitoring streams - a new class of data management applications. In *VLDB*, 2002.

[11] K. Chakrabarti, M. N. Garofalakis, R. Rastogi, and K. Shim. Approximate query processing using wavelets. In *VLDB 2000, Proceedings of 26th International Conference on Very Large Data Bases, September 10-14, 2000, Cairo, Egypt*, pages 111–122, 2000.

[12] S. Chandrasekaran, O. Cooper, A. Deshpande, M. J. Franklin, J. M. Hellerstein, W. Hong, S. Krishnamurthy, S. R. Madden, V. Raman, F. Reiss, and M. A. Shah. TelegraphCQ: Continuous dataflow processing for an uncertain world. In *CIDR*, 2003.

[13] O. Cooper, A. Edakkunni, M. Franklin, W. Hong, S. Jeffery, S. Krishnamurthy, F. Reiss, S. Rizvi, and E. Wu. Hifi: A unified architecture for high fan-in systems. In *VLDB*, 2004.

[14] R. Cowell, P. Dawid, S. Lauritzen, and D. Spiegelhalter. *Probabilistic Networks and Expert Systems*. Spinger, New York, 1999.

[15] I. Crossbow. Wireless sensor networks (mica motes). `http://www.xbow.com/Products/Wireless_Sensor_Networks.htm`.

[16] A. Das, J. Gehrke, and M. Riedewald. Approximate join processing over data streams. In *Proceedings of SIGMOD*, 2003.

[17] T. Dean and K. Kanazawa. A model for reasoning about persistence and causation. *Computational Intelligence*, 5(3):142–150, 1989.

[18] A. Deshpande, M. Garofalakis, and R. Rastogi. Independence is Good: Dependency-Based Histogram Synopses for High-Dimensional Data. In *Proceedings of SIGMOD*, May 2001.

[19] A. Deshpande, C. Guestrin, W. Hong, and S. Madden. Intel lab data. Web Page. `http://berkeley.intel-research.net/labdata`.

[20] A. Deshpande, C. Guestrin, W. Hong, and S. Madden. Exploiting correlated attributes in acquisitional query processing. In *ICDE*, 2005.

[21] A. Deshpande, C. Guestrin, S. Madden, J. Hellerstein, and W. Hong. Model-driven data acquisition in sensor networks. In *VLDB*, 2004.

[22] A. Deshpande, C. Guestrin, S. Madden, J. M. Hellerstein, and W. Hong. Model-driven data acquisition in sensor networks. In *VLDB*, 2004.

[23] A. Deshpande, S. Nath, P. Gibbons, and S. Seshan. Cache-and-query for wide area sensor databases. In *Proceedings of SIGMOD*, 2003.

[24] A. Faradjian, J. Gehrke, and P. Bonnet. GADT: A Probability Space ADT For Representing and Querying the Physical World. In *ICDE*, 2002.

[25] N. Friedman. Learning belief networks in the presence of missing values and hidden variables. In *Proc. 14th International Conference on Machine Learning*, pages 125–133, 1997.

[26] L. Getoor. *Learning Statistical Models from Relational Data*. PhD thesis, Stanford University, 2001.

[27] L. Getoor, B. Taskar, and D. Koller. Selectivity estimation using probabilistic models. In *Proceedings of SIGMOD*, May 2001.

[28] P. B. Gibbons. Distinct sampling for highly-accurate answers to distinct values queries and event reports. In *Proc. of VLDB*, Sept 2001.

[29] P. B. Gibbons and M. Garofalakis. Approximate query processing: Taming the terabytes (tutorial), September 2001.

[30] P. B. Gibbons and Y. Matias. New sampling-based summary statistics for improving approximate query answers. In *SIGMOD*, 1998.

[31] C. Guestrin, P. Bodik, R. Thibaux, M. Paskin, and S. Madden. Distributed regression: an efficient framework for modeling sensor network data. In *Proceedings of IPSN*, 2004.

[32] C. E. Guestrin, D. Koller, and R. Parr. Multiagent planning with factored MDPs. In *14th Neural Information Processing Systems (NIPS-14)*, pages 1523–1530, Vancouver, Canada, December 2001.

[33] D. Hand, H. Mannila, and P. Smyth. *Principles of Data Mining*. MIT Press, 2001.

[34] D. Heckerman. A tutorial on learning with bayesian networks, 1995.

[35] J. Hellerstein, W. Hong, S. Madden, and K. Stanek. Beyond average: Towards sophisticated sensing with queries. In *Proceedings of the First Workshop on Information Processing in Sensor Networks (IPSN)*, March 2003.

[36] J. M. Hellerstein, M. J. Franklin, S. Chandrasekaran, A. Deshpande, K. Hildrum, S. Madden, V. Raman, and M. Shah. Adaptive query processing: Technology in evolution. *IEEE Data Engineering Bulletin*, 23(2):7–18, 2000.

[37] Intel Research. Exploratory research - deep networking. Web Site. `http://www.intel.com/research/exploratory/heterogeneous.htm#preventativ%emaintenance`.

[38] Z. G. Ives, D. Florescu, M. Friedman, A. Levy, and D. S. Weld. An adaptive query execution system for data integration. In *Proceedings of SIGMOD*, 1999.

[39] A. Jain, E. Change, and Y.-F. Wang. Adaptive stream resource management using kalman filters. In *Proceedings of SIGMOD*, 2004.

[40] Jim Madden, Director UCSD Network Operations. Personal communication, July 2004.

[41] G. Kollios, J. Considine, F. Li, and J. Byers. Approximate aggregation techniques for sensor databases. In *ICDE*, 2004.

[42] L. V. S. Lakshmanan, N. Leone, R. Ross, and V. S. Subrahmanian. Probview: a flexible probabilistic database system. *ACM TODS*, 22(3):419–469, 1997.

[43] U. Lerner, B. Moses, M. Scott, S. McIlraith, and D. Koller. Monitoring a complex physical system using a hybrid dynamic bayes net. In *UAI*, 2002.

[44] C. Lin, C. Federspiel, and D. Auslander. Multi-Sensor Single Actuator Control of HVAC Systems. In *Internation Conference for Enhanced Building Operations*, 2002.

[45] S. Madden. *The Design and Evaluation of a Query Processing Architecture for Sensor Networks*. PhD thesis, UC Berkeley, 2003.

[46] S. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong. The design of an acquisitional query processor for sensor networks. In *Proceedings of SIGMOD*, 2003. To Appear.

[47] S. Madden, W. Hong, J. M. Hellerstein, and M. Franklin. TinyDB web page. `http://telegraph.cs.berkeley.edu/tinydb`.

[48] A. Mainwaring, J. Polastre, R. Szewczyk, and D. Culler. Wireless sensor networks for habitat monitoring. In *ACM Workshop on Sensor Networks and Applications*, 2002.

[49] MIT CSAIL Center for Information Security and Privacy. Home page. `http://csg.lcs.mit.edu/CISP/`.

[50] T. Mitchell. *Machine Learning*. McGraw Hill, 1997.

[51] R. Motwani, J. Widom, A. Arasu, B. Babcock, S. Babu, M. Datar, G. Manku, C. Olston, J. Rosenstein, and R. Varma. Query processing, resource management, and approximation in a data stream management system. In *Proceedings of CIDR*, 2003.

[52] R. Motwani, J. Window, A. Arasu, B. Babcock, S.Babu, M. Data, C. Olston, J. Rosenstein, and R. Varma. Query processing, approximation and resource management in a data stream management system. In *First Annual Conference on Innovative Database Research (CIDR)*, 2003.

[53] S. Nath and P. B. Gibbons. Synopsis diffusion for robust aggregation in sensor networks. In *Proceedings of VLDB*, 2004.

[54] C. Olston and J.Widom. Best effort cache sychronization with source cooperation. In *Proceedings of SIGMOD*, 2002.

[55] C. Olston, B. T. Loo, and J. Widom. Adaptive precision setting for cached approximate values. In *Proceedings of SIGMOD*, May 2001.

[56] M. A. Paskin and C. E. Guestrin. Robust probabilistic inference in distributed systems. In *UAI*, 2004. In the *20th International Conference on Uncertainty in Artificial Intelligence*.

[57] J. Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann Publishers, 1988.

[58] J. Pearl. *Causality : Models, Reasoning, and Inference*. Cambridge University Press, 2000.

[59] V. Poosala, Y. E. Ioannidis, P. J. Haas, and E. J. Shekita. “Improved Histograms for Selectivity Estimation of Range Predicates”. In *SIGMOD*, 1996.

[60] M. L. Puterman. *Markov decision processes: Discrete stochastic dynamic programming*. Wiley, 1994.

[61] V. Raman, B. Raman, and J. M. Hellerstein. Online dynamic reordering. *The VLDB Journal*, 9(3), 2002.

[62] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 1994.

[63] R. Szewczyk, A. Mainwaring, J. Polastre, and D. Culler. An analysis of a large scale habitat monitoring application. In *Proceedings of SenSys*, 2004.

[64] O. Wolfson, A. P. Sistla, B. Xu, J. Zhou, and S. Chamberlain. DOMINO: Databases fOr MovINg Objects tracking. In *Proceedings of SIGMOD*, Philadelphia, PA, June 1999.