Model-Driven Data Acquisition in Sensor Networks*

Amol Deshpande[†] Carlos Guestrin[‡] Samuel R. Madden[§ ‡]
Joseph M. Hellerstein[† ‡] Wei Hong[‡]

[†] UC Berkeley
{amol,jmh}@cs.berkeley.edu

[‡] Intel Research Berkeley
{guestrin,whong}@intel-research.net

[§] MIT
madden@csail.mit.edu

Abstract

Declarative queries are proving to be an attractive paradigm for interacting with networks of wireless sensors. The metaphor that "the sensornet is a database" is problematic, however, because sensors do not exhaustively represent the data in the real world. In order to map the raw sensor readings onto physical reality, a *model* of that reality is required to complement the readings. In this paper, we enrich interactive sensor querying with statistical modeling techniques. We demonstrate that such models can help provide answers that are both more meaningful, and, by introducing approximations with probabilistic confidences, significantly more efficient to compute in both time and energy. Utilizing the combination of a model and live data acquisition raises the challenging optimization problem of selecting the best sensor readings to acquire, balancing the increase in the confidence of our answer against the communication and data acquisition costs in the network. We describe an exponential time algorithm for finding the optimal solution to this optimization problem, and a polynomial-time heuristic for identifying solutions that perform well in practice. We evaluate our approach on several real-world sensor-network data sets, taking into account the real measured data and communication quality, demonstrating that our model-based approach provides a high-fidelity representation of the real phenomena and leads to significant performance gains versus traditional data acquisition techniques.

1 Introduction

Database technologies are beginning to have a significant impact in the emerging area of wireless sensor networks (sensornets). The sensornet community has embraced declarative queries as a key programming paradigm for large sets of sensors. This is seen in academia in the calls for papers for leading conferences and workshops in the sensornet area [2, 1], and in a number of prior research publications ([21],[30],[17], etc). In the emerging industrial arena, one of the leading vendors (Crossbow) is bundling a query processor with their devices, and providing query processor training as part of their customer support. The area of sensornet querying represents an unusual opportunity for database researchers to apply their expertise in a new area of computer systems.

*This work was supported by Intel Corporation, and by NSF under the grant IIS-0205647.

**Proceedings of the 30th VLDB Conference,
Toronto, Canada, 2004**

Declarative querying has proved powerful in allowing programmers to "task" an entire network of sensor nodes, rather than requiring them to worry about programming individual nodes. However, the metaphor that "the sensornet is a database" has proven misleading. Databases are typically treated as complete, authoritative sources of information; the job of a database query engine has traditionally been to answer a query "correctly" based upon all the available data. Applying this mindset to sensornets results in two problems:

1. **Misrepresentations of data:** In the sensornet environment, it is impossible to gather *all* the relevant data. The physically observable world consists of a set of continuous phenomena in both time and space, so the set of relevant data is in principle infinite. Sensing technologies acquire *samples* of physical phenomena at discrete points in time and space, but the data acquired by the sensornet is unlikely to be a random (i.i.d.) sample of physical processes, for a number of reasons (non-uniform placement of sensors in space, faulty sensors, high packet loss rates, etc). So a straightforward interpretation of the sensornet readings as a "database" may not be a reliable representation of the real world.
2. **Inefficient approximate queries:** Since a sensornet cannot acquire all possible data, any readings from a sensornet are "approximate", in the sense that they only represent the true state of the world at the discrete instants and locations where samples were acquired. However, the leading approaches to query processing in sensornets [30, 21] follow a completist's approach, acquiring as much data as possible from the environment at a given point in time, even when *most of that data provides little benefit in approximate answer quality*. We show examples where query execution cost – in both time and power consumption – can be orders of magnitude more than is appropriate for a reasonably reliable answer.

1.1 Our contribution

In this paper, we propose to compensate for both of these deficiencies by incorporating statistical *models* of real-world processes into a sensornet query processing architecture. Models can help provide more robust interpretations of sensor readings: for example, they can account for biases in spatial sampling, can help identify sensors that are providing faulty data, and can extrapolate the values of missing sensors or sensor readings at geographic locations where sensors are no longer operational. Furthermore, models provide a framework for optimizing the acquisition of sensor readings: sensors should

be used to acquire data only when the model itself is not sufficiently rich to answer the query with acceptable confidence.

Underneath this architectural shift in sensornet querying, we define and address a key optimization problem: given a query and a model, choose a data acquisition plan for the sensornet to best refine the query answer. This optimization problem is complicated by two forms of dependencies: one in the statistical *benefits* of acquiring a reading, the other in the system *costs* associated with wireless sensor systems.

First, any non-trivial statistical model will capture correlations among sensors: for example, the temperatures of geographically proximate sensors are likely to be correlated. Given such a model, the benefit of a single sensor reading can be used to improve estimates of other readings: the temperature at one sensor node is likely to improve the confidence of model-driven estimates for nearby nodes.

The second form of dependency hinges on the connectivity of the wireless sensor network. If a sensor node far is not within radio range of the query source, then one cannot acquire a reading from far without forwarding the request/result pair through another node $near$. This presents not only a non-uniform cost model for acquiring readings, but one with dependencies: due to multi-hop networking, the acquisition cost for $near$ will be much lower if one has already chosen to acquire data from far by routing through $near$.

To explore the benefits of the model-based querying approach we propose, we are building a prototype called BBQ[1] that uses a specific model based on time-varying multivariate Gaussians. We describe how our generic model-based architecture and querying techniques are specifically applied in BBQ. We also present encouraging results on real-world sensornet trace data, demonstrating the advantages that models offer for queries over sensor networks.

2 Overview of approach

In this section, we provide an overview of our basic architecture and approach, as well as a summary of BBQ. Our architecture consists of a declarative query processing engine that uses a probabilistic model to answer questions about the current state of the sensor network. 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 sensing node 5, voltage on sensing node 12). Typically, there is one such attribute per sensor type per sensing node. 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. Such models can be learned from historical data using standard algorithms (*e.g.*, [23]).

Users query for information about the values of particular attributes or in certain regions of the network, much as they would in a traditional SQL database. Unlike database queries, however, sensornet queries request real-time information about the environment, rather than information about a stored collection of data. The model is used to estimate sensor readings in the current time period; these estimates form the answer the query. In the process of generating these

[1]BBQ is short for Barbie-Q: A Tiny-Model Query System

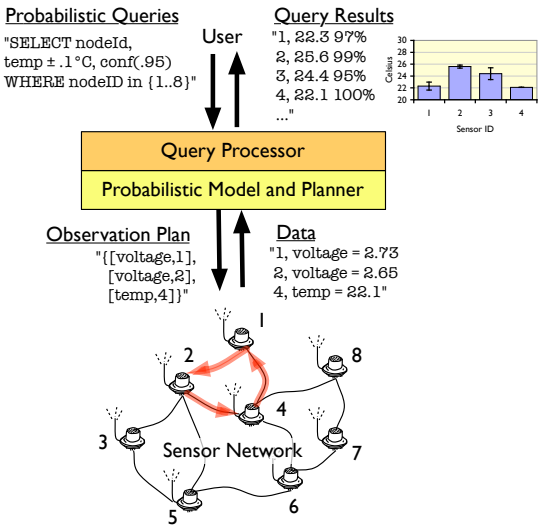

Figure 1: Our architecture for model-based querying in sensor networks.

estimates, the model may interrogate the sensor network for updated readings that will help to refine estimates for which its uncertainty is high. As time passes, the model may also update its estimates of sensor values, to reflect expected temporal changes in the data.

In BBQ, we use a specific model based on time-varying multivariate Gaussians; we describe this model below. We emphasize, however, that our approach is general with respect to the model, and that more or less complex models can be used instead. New models require no changes to the query processor and can reuse code that interfaces with and acquires particular readings from the sensor network. The main difference occurs in the algorithms required to solve the probabilistic inference tasks described in Section 3. These algorithms have been widely developed for many practical models (*e.g.*, [23]).

Figure 1 illustrates our basic architecture through an example. Users submit SQL queries to the database, which are translated into probabilistic computations over the model (Section 3). The queries include 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-representation shown in the upper right of Figure 1). 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. Based on the model, the system decides that the most efficient way to answer the query with the requested confidence 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 acquires samples in this order, 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

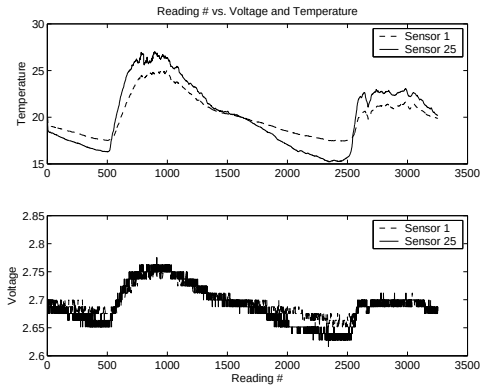

Figure 2: Trace of voltage and temperature readings over a two day period from a single mote-based sensor. Notice the close correlation between the two attributes.

was over temperature. This happens for two reasons:

1. **Correlations in Value:** Temperature and voltage are highly correlated, as illustrated by Figure 2 which shows the temperature and voltage readings for two days of sensor readings from a pair of Berkeley Mica2 Motes [6] that we deployed in the Intel Research Lab in Berkeley, California. Note how voltage tracks temperature, and how temperature variations across motes, even though of noticeably different magnitudes, are very similar. The relationship between temperature and voltage is due to the fact that, for many types of batteries, as they heat or cool, their voltages vary significantly (by as much as 1% per degree). The voltages may also decrease as the sensor nodes consume energy from the batteries, but the time scale at which that happens is much larger than the time scale of temperature variations, and so the model can use voltage changes to infer temperature changes.
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 [6], the temperature sensor requires several orders of magnitude more energy to sample as simply reading battery voltage (see Table 1).

One of the important properties of many probabilistic models (including the one used in BBQ) is that they can capture correlations between different attributes. We will see how we can exploit such correlations during optimization to generate efficient query plans in Section 4.

2.1 Confidence intervals and correlation models

The user in Figure 1 could have requested 100% confidence and no error tolerance, in which case the model would have required us to interrogate every sensor. The returned result could still include some uncertainty, as the model may not have readings from particular sensors or locations at some points in time (due to sensor or communications failures, or lack of sensor instrumentation at a particular location). These confidence intervals computed from our probabilistic model provide considerably more information than traditional sensor network systems like TinyDB and Cougar provide in this setting. With those systems, the user would simply get no data regarding those missing times and locations.

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. In fact, in our experiments with BBQ on several real-world data sets, we see a number of cases where strong correlations between sensors during certain times of the day mean that even queries with relatively tight confidence bounds can be answered with a very small number of sensor observations. In many cases, these tight confidences can be provided *despite the fact that sensor readings have changed significantly*. This is because known correlations between sensors make it possible to predict these changes: for example, in Figure 2, it is clear that the temperature on the two sensors is correlated given the time of day. During the daytime (e.g., readings 600-1200 and 2600-3400), sensor 25, which is placed near a window, is consistently hotter than sensor 1, which is in the center of our lab. A good model will be able to infer, with high confidence that, during daytime hours, sensor readings on sensor 25 are 1-2 degrees hotter than those at sensor 1 without actually observing sensor 25. Again, this is in contrast to existing sensor network querying systems, where sensors are continuously sampled and readings are always reported whenever small absolute changes happen.

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.*, [23]), but can be difficult in some applications. Before presenting the specific model class used in BBQ, we note that, in general, a probabilistic model is only as good at prediction as the data used to train it. Thus, it may be the case that the temperature between sensors 1 and 25 would not show the same relationship during a different season of the year, or in a different climate – in fact, one might expect that when the outside temperature is very cold, sensor 25 will read less than sensor 1 during the day, just as it does during the night time. Thus, 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; in fact, the model we use in BBQ 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.

2.2 BBQ

In BBQ, we use a specific probabilistic model based on time-varying multivariate Gaussians. 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 3(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 3(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 3(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.

In BBQ, we use historical data to construct the initial representation of this pdf p. In the implementation described in this paper, we obtained such data using TinyDB (a traditional sensor network querying system)[2]. 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 3.2, but illustrate it graphically with our 2-dimensional Gaussian in Figures 3(B) - 3(D). Suppose that we have an initial Gaussian shown in Figure 3(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 3(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 3(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 both the mean and variance may have shifted from their initial values, as shown in Figure 3(D).

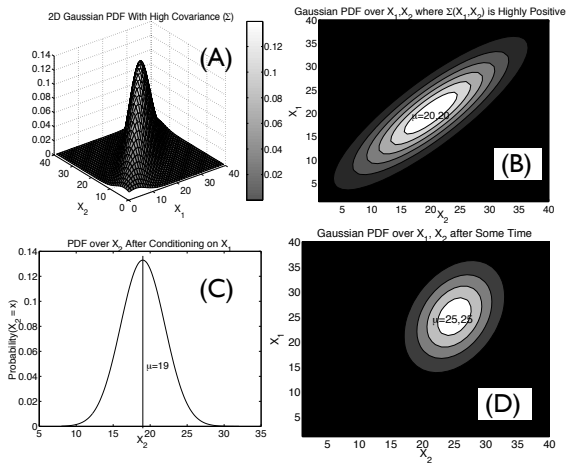

Figure 3: Example of Gaussians: (a) 3D plot of a 2D Gaussian with high covariance; (b) the same Gaussian viewed as a contour plot; (c) the resulting Gaussian over X_2 after a particular value of X_1 has been observed; finally, (d) shows how, as uncertainty about X_1 increases from the time we last observed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

2.3 Supported queries

Answering queries probabilistically based on a distribution (*e.g.*, the Gaussian representation described above) is conceptually straightforward. Suppose, for example, that a query asks for an ϵ approximation to the value of a set of attributes, with confidence at least $1 - \delta$. We can use our pdf to compute the expected value, μ_i, of each attribute in the query. These will be our reported values. We can the 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 the 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 discuss this optimization problem in more detail in Section 4.

In Section 3, we show how our query and optimization engine are used in BBQ to answer a number of SQL queries, including (i) simple selection queries requesting the value of one or more sensors, or the value of all sensors in a given geographic region, (ii) whether or not a predicate over one or more sensor readings is true, and (iii) grouped aggregates such as AVERAGE.

For the purposes of this paper, we focus on multiple one-shot queries over the current state of the network, rather than continuous queries. We can provide simple continuous query functionality by issuing a one-shot query at regular time intervals. In our experimental section, we compare this approach to existing continuous query systems for sensor networks (like TinyDB). We also discuss how knowledge of a standing, continuous query could be used to further optimize our performance in Section 6.

In this paper, there are certain types of queries which we do not address. For example, BBQ is not designed for outlier detection – that is, it will not immediately detect when a single sensor is reading something that is very far from its expected value or from the value of neighbors it has been correlated with in the past. We suggest ways in which our approach can be amended to handle outliers in Section 6.

2.4 Networking model and observation plan format

Our initial implementation of BBQ focuses on static sensor networks, such as those deployed for building and habitat monitoring. For this reason, we assume that network topologies change relatively slowly. We capture network topology information when collecting data by including, for each sensor, a vector of link quality estimates for neighboring sensor nodes. We use this topology information when constructing query plans by assuming that nodes that were previously connected will still be in the near future. When executing a plan, if we observe that a particular link is not available (*e.g.*, because one of the sensors has failed), we update our topology model accordingly.We can continue to collect new topology information as we query the network, so that new links will also become available. This approach will be effective if the

[2]Though these initial observations do consume some energy up-front, we will show that the long-run energy savings obtained from using a model will be much more significant.

topology is relatively stable; highly dynamic topologies will need more sophisticated techniques, which is a problem we briefly discuss in Section 6.

In BBQ, observation plans consist of a list of sensor nodes to visit, and, at each of these nodes, a (possibly empty) list of attributes that need to be observed at that node. The possibility of visiting a node but observing nothing is included to allow plans to observe portions of the network that are separated by multiple radio hops. We require that plans begin and end at sensor id 0 (the *root*), which we assume to be the node that interfaces the query processor to the sensor network.

2.5 Cost model

During plan generation and optimization, we need to be able to compare the relative costs of executing different plans in the network. As energy is the primary concern in battery-powered sensornets [15, 26], our goal is to pick plans of minimum energy cost. The primary contributors to energy cost are communication and data acquisition from sensors (CPU overheads beyond what is required when acquiring and sending data are small, as there is no significant processing done on the nodes in our setting).

Our cost model uses numbers obtained from the data sheets of sensors and the radio used on Mica2 motes with a Crossbow MTS400 [6] environmental sensor board. For the purposes of our model, we assume that the sender and receiver are well synchronized, so that a listening sensor turns on its radio just as a sending node begins transmitting[3]. On current generation motes, the time required to send a packet is about 27 ms. The ChipCon CC1000 radio on motes uses about 15 mW of energy in both send and receive modes, meaning that both sender and receiver consume about .4 mJ of energy. Table 1 summarizes the energy costs of acquiring readings from various sensors available for motes. In this paper, we primarily focus on temperature readings, though we briefly discuss other attributes as well in Section 5. Assuming we are acquiring temperature readings (which cost .5 J per sample), we compute the cost of a plan that visits s nodes and acquires a readings to be $(.4\times 2)\times s + .5\times a$ if there are no lost packets. In Section 4.1, we generalize this idea, and consider lossy communication. Note that this cost treats the entire network as a shared resource in which power needs to be conserved equivalently on each mote. More sophisticated cost models that take into account the relative importance of nodes close to the root could be used, but an exploration of such cost models is not needed to demonstrate the utility of our approach.

Sensor	**Energy Per Sample (@3V), mJ**
Solar Radiation [29]	.525
Barometric Pressure [16]	0.003
Humidity and Temperature[28]	0.5
Voltage	0.00009

Table 1: Summary of Power Requirements of Crossbow MTS400 Sensorboard (From [20]). Certain sensors, such as solar radiation and humidity (which includes a temperature sensor) require about a second per sample, explaining their high per-sample energy cost.

3 Model-based querying

As described above, the central element in our approach is the use of a probabilistic model to answer queries about the attributes in a sensor network. This section focuses on a few specific queries: range predicates, attribute-value estimates, and standard aggregates. We provide a review of the standard methodology required to use a probabilistic model to answer these queries. This probabilistic model can answer many other significantly more complex queries as well; we outline some of these directions in Section 6.

3.1 Probabilistic queries

A *probability density function* (pdf), or *prior density*, $p(X_1, \ldots, X_n)$ assigns a probability for each joint value $x_1, \ldots, x_n$ for the attributes $X_1, \ldots, X_n$.

Range queries: We begin by considering range queries that ask if an attribute X_i is in the range $[a_i, b_i]$. Typically, we would need to query the sensor network to obtain the value of the attribute and then test whether the query is true or false. Using a probabilistic model, we can compute the probability $P(X_i \in [a_i, b_i])$. If this probability is very high, we are confident that the predicate $X_i \in [a_i, b_i]$ is true. Analogously, if the probability is very low, we are confident that the predicate is false. Otherwise, we may not have enough information to answer this query with sufficient confidence and may need to acquire more data from the sensor network. The probability $P(X_i \in [a_i, b_i])$ can be computed in two steps: First, we *marginalize*, or project, the pdf $p(X_1, \ldots, X_n)$ to a density over only attribute X_i:

$$p(x_i) = \int p(x_1, \ldots, x_n) dx_1 \ldots dx_{i-1} dx_{i+1} \ldots dx_n.$$

Marginalization gives us the pdf over only X_i. We can then compute $P(X_i \in [a_i, b_i])$ simply by:

$$P(X_i \in [a_i, b_i]) = \int_{a_i}^{b_i} p(x_i) dx_i. \quad (1)$$

Range queries over multiple attributes can be answered by marginalizing the joint pdf to that set of attributes. Thus, we can use the joint probability density $p(X_1, \ldots, X_n)$ to provide probabilistic answers to any range query. If the user specifies a confidence level $1-\delta$, for $\delta \in [0, 1]$, we can answer the query if this confidence is either $P(X_i \in [a_i, b_i]) > 1-\delta$ or $P(X_i \in [a_i, b_i]) < \delta$. However, in some cases, the computed confidences may be low compared to the ones required by the query, and we need to make new observations, that is, to acquire new sensor readings.

Suppose that we observe the value of attribute X_j to be x_j, we can now use Bayes' rule to *condition* our joint pdf

[3]In practice, this is done by having the receiver periodically sample the radio, listening for a preamble signal that indicates a sender is about to begin transmission; when this preamble is heard, it begins listening continuously. Though this periodic radio sampling uses some energy, it is small, because the sampling duty cycle can be 1% or less (and is an overhead paid by any application that uses the radio).

$p(X_1, \ldots, X_n)$ on this value[4], obtaining:

$$p(X_1, \ldots, X_{j-1}, X_{j+1}, \ldots, X_n \mid x_j) = \frac{p(X_1, \ldots, X_{j-1}, x_j, X_{j+1}, \ldots, X_n)}{p(x_j)}.$$

The *conditional probability density function* $p(X_1, \ldots, X_{j-1}, X_{j+1}, \ldots, X_n \mid x_j)$, also referred as the *posterior density* given the observation x_j, will usually lead to a more confident estimate of the probability ranges. Using marginalization, we can compute $P(X_i \in [a_i, b_i] \mid x_j)$, which is often more certain than the prior probability $P(X_i \in [a_i, b_i])$. In general, we will make a set of observations $\mathbf{o}$, and, after conditioning on these observations, obtain $p(\mathbf{X} \mid \mathbf{o})$, the posterior probability of our set of attributes $\mathbf{X}$ given $\mathbf{o}$.

Example 3.1 *In BBQ, the pdf is represented by a multivariate Gaussian with mean vector μ and covariance matrix Σ. In Gaussians, marginalization is very simple. If we want to marginalize the pdf to a subset $\mathbf{Y}$ of the attributes, we simply select the entries in μ and Σ corresponding to these attributes, and drop the other entries obtaining a lower dimensional mean vector $\mu_{\mathbf{Y}}$ and covariance matrix $\Sigma_{\mathbf{YY}}$. For a Gaussian, there is no closed-form solution for Equation (1). However, this integration problem is very well understood, called the* error function *(erf), with many well-known, simple approximations.*

Interestingly, if we condition a Gaussian on the value of some attributes, the resulting pdf is also a Gaussian. The mean and covariance matrix of this new Gaussian can be computed by simple matrix operations. Suppose that we observe value $\mathbf{o}$ for attributes $\mathcal{O}$, the mean $\mu_{\mathbf{Y}|\mathbf{o}}$ and covariance matrix $\Sigma_{\mathbf{Y}|\mathbf{o}}$ of the pdf $p(\mathbf{Y} \mid \mathbf{o})$ over the remaining attributes are given by:

$$\begin{aligned} \mu_{\mathbf{Y}|\mathbf{o}} &= \mu_{\mathbf{Y}} + \Sigma_{\mathbf{Y}\mathcal{O}}\Sigma_{\mathcal{O}\mathcal{O}}^{-1}(\mathbf{o} - \mu_{\mathcal{O}}), \\ \Sigma_{\mathbf{Y}|\mathbf{o}} &= \Sigma_{\mathbf{YY}} - \Sigma_{\mathbf{Y}\mathcal{O}}\Sigma_{\mathcal{O}\mathcal{O}}^{-1}\Sigma_{\mathcal{O}\mathbf{Y}}, \end{aligned} \quad (2)$$

where $\Sigma_{\mathbf{Y}\mathcal{O}}$ denotes the matrix formed by selecting the rows $\mathbf{Y}$ and the columns $\mathcal{O}$ from the original covariance matrix Σ. Note that the posterior covariance matrix $\Sigma_{\mathbf{Y}|\mathbf{o}}$ does not depend on the actual observed value $\mathbf{o}$. We thus denote this matrix by $\Sigma_{\mathbf{Y}|\mathcal{O}}$. In BBQ, by using Gaussians, we can thus compute all of the operations required to answer our queries by performing only basic matrix operations. □

Value queries: In addition to range queries, a probability density function can, of course, be used to answer many other query types. For example, if the user is interested in the value of a particular attribute X_i, we can answer this query by using the posterior pdf to compute the mean $\bar{x}_i$ value of X_i, given the observations $\mathbf{o}$:

$$\bar{x}_i = \int x_i \, p(x_i \mid \mathbf{o}) dx_i.$$

We can additionally provide confidence intervals on this estimate of the value of the attribute: for a given error bound $\varepsilon > 0$, the confidence is simply given by $P(X_i \in [\bar{x}_i - \varepsilon, \bar{x}_i + \varepsilon] \mid \mathbf{o})$, which can be computed as in the range queries in Equation (1). If this confidence is greater than the user specified value $1 - \delta$, then we can provide a probably approximately correct value for the attribute, without observing it.

AVERAGE aggregates: Average queries can be answered in a similar fashion, by defining an appropriate pdf. Suppose that we are interested in the average value of a set of attributes $\mathcal{A}$. For example, if we are interested in the average temperature in a spatial region, we can define $\mathcal{A}$ to be the set of sensors in this region. We can now define a random variable Y to represent this average by $Y = (\sum_{i \in \mathcal{A}} X_i)/|\mathcal{A}|$. The pdf for Y is simply given by appropriate marginalization of the joint pdf over the attributes in $\mathcal{A}$:

$$p(Y = y \mid \mathbf{o}) = \int p(x_1, \ldots, x_n \mid \mathbf{o}) \, \mathbb{1}\left[\left(\sum_{i \in \mathcal{A}} x_i/|\mathcal{A}|\right) = y\right] dx_1 \ldots dx_n,$$

where $\mathbb{1}[\cdot]$ is the indicator function.[5] Once $p(Y = y \mid \mathbf{o})$ is defined, we can answer an average query by simply defining a value query for the new random variable Y as above. We can also compute probabilistic answers to more complex aggregation queries. For example, if the user wants the average value of the attributes in $\mathcal{A}$ that have value greater than t, we can define a random variable Z:

$$Z = \frac{\sum_{i \in \mathcal{A}} X_i \mathbb{1}(X_i > c)}{\sum_{i \in \mathcal{A}} \mathbb{1}(X_i > c)},$$

where $\frac{0}{0}$ is defined to be 0. The pdf of Z is given by:

$$p(Z = z \mid \mathbf{o}) = \int p(x_1, \ldots, x_n \mid \mathbf{o}) \, \mathbb{1}\left[\left(\frac{\sum_{i \in \mathcal{A}, x_i > c} x_i}{\sum_{i \in \mathcal{A}, x_i > c} 1}\right) = y\right] dx_1 \ldots dx_n.$$

In general, this inference problem, *i.e.*, computing these integrals, does not have a closed-form solution, and numerical integration techniques may be required.

Example 3.2 *BBQ focuses on Gaussians. In this case, each posterior mean $\bar{x}_i$ can be obtained directly from our mean vector by using the conditioning rule described in Example 3.1. Interestingly, the sum of Gaussian random variables is also Gaussian. Thus, if we define an AVERAGE query $Y = (\sum_{i \in \mathcal{A}} X_i)/|\mathcal{A}|$, then the pdf for Y is a Gaussian. All we need now is the variance of Y, which can be computed in closed-form from those of each X_i by:*

$$\begin{aligned} E[(Y - \mu_Y)^2] &= E[(\textstyle\sum_{i \in \mathcal{A}} X_i - \mu_i)^2/|\mathcal{A}|^2], \\ &= \tfrac{1}{|\mathcal{A}|^2}\left(\textstyle\sum_{i \in \mathcal{A}} E[(X_i - \mu_i)^2] \right. \\ &\quad +2 \textstyle\sum_{i \in \mathcal{A}} \sum_{j \in \mathcal{A}, j \neq i} \\ &\quad \left. E[(X_i - \mu_i)(X_j - \mu_j)]\right). \end{aligned}$$

Thus, the variance of Y is given by a weighted sum of the

[4]The expression $p(x|y)$ is read "the probability of x given y", and represents the pdf of variable x given a particular value of y. Bayes' rule allows conditional probabilities to be computed in scenarios where we only have data on the inverse conditional probability: $p(x|y) = \frac{p(y|x)p(x)}{p(y)}$.

[5]The indicator function translates a Boolean predicate into the arithmetic value 1 (if the predicate is true) and 0 (if false).

variances of each X_i, plus the covariances between X_i and X_j, all of which can be directly read off the covariance matrix Σ. Therefore, we can answer an AVERAGE query over a subset of the attributes $\mathcal{A}$ in closed-form, using the same procedure as value queries. For the more general queries that depend on the actual value of the attributes, even with Gaussians, we require a numerical integration procedure. □

3.2 Dynamic models

Thus far, we have focused on a single static probability density function over the attributes. This distribution represents *spatial* correlation in our sensor network deployment. However, many real-world systems include attributes that evolve over time. In our deployment, the temperatures have both temporal and spatial correlations. 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.

In particular, for each (discrete) time index t, we should estimate a pdf $p(X_1^t, \ldots, X_n^t \mid \mathbf{o}^{1\ldots t})$ that assigns a probability for each joint assignment to the attributes at time t, given $\mathbf{o}^{1\ldots t}$, all observations made up to time t. A dynamic model describes 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 $p(X_1^t, \ldots, X_n^t \mid \mathbf{o}^{1\ldots t})$. Thus, we can use all measurements made up to time t to improve our estimate of the pdf at time $t+1$.

For simplicity, we restrict our presentation to *Markovian* models, 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 very simple, yet often effective, model for representing a stochastic dynamical system. Here, 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 this transition model, we can compute $p(X_1^{t+1}, \ldots, X_n^{t+1} \mid \mathbf{o}^{1\ldots t})$ using a simple marginalization operation:

$$p(x_1^{t+1}, \ldots, x_n^{t+1} \mid \mathbf{o}^{1\ldots t}) = \int p(x_1^{t+1}, \ldots, x_n^{t+1} \mid x_1^t, \ldots, x_n^t) p(x_1^t, \ldots, x_n^t \mid \mathbf{o}^{1\ldots t}) dx_1^t \ldots dx_n^t.$$

This formula assumes that the transition model $p(\mathbf{X}^{t+1} \mid \mathbf{X}^t)$ is the same for all times t. In our deployment, for example, in the mornings the 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. In our experimental results in Section 5, we address this problem by simply 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 $t+1$, as in Section 3.1, 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 $X_1, \ldots, X_n$. This process of pushing our estimate for the density at time t through the transition model and then conditioning on the measurements at time $t+1$ is often called *filtering*. In contrast to the static model described in the previous section, filtering allows us to condition our estimate on the complete history of observations, which, as we will see in Section 5, can significantly reduce the number of observations required for obtaining confident approximate answers to our queries.

Example 3.3 *In BBQ, we focus on Gaussian distributions; for these distributions the filtering process is called a* Kalman filter. *The transition model $p(X_1^{t+1}, \ldots, X_n^{t+1} \mid X_1^t, \ldots, X_n^t)$ can be learned from data with two simple steps: First, we learn a mean and covariance matrix for the joint density $p(X_1^{t+1}, \ldots, X_n^{t+1}, X_1^t, \ldots, X_n^t)$. That is, we form tuples $\langle X_1^{t+1}, \ldots, X_n^{t+1}, X_1^t, \ldots, X_n^t \rangle$ for our attributes at every consecutive times t and $t+1$, and use these tuples to compute the joint mean vector and covariance matrix. Then, we use the conditioning rule described in Example 3.1 to compute the transition model:*

$$p(\mathbf{X}^{t+1} \mid \mathbf{X}^t) = \frac{p(\mathbf{X}^{t+1}, \mathbf{X}^t)}{p(\mathbf{X}^t)}.$$

Once we have obtained this transition model, we can answer our queries in a similar fashion as described in Examples 3.1 and 3.2. □

4 Choosing an observation plan

In the previous section, we showed that our pdfs can be conditioned on the value $\mathbf{o}$ of the set of observed attributes to obtain a more confident answer to our query. Of course, the choice of attributes that we observe will crucially affect the resulting posterior density. In this section, we focus on selecting the attributes that are expected to increase the confidences in the answer to our particular query at minimal cost. We first formalize the notion of cost of observing a particular set of attributes. Then, we describe the expected improvement in our answer from observing this set. Finally, we discuss the problem of optimizing the choice of attributes.

4.1 Cost of observations

Let us denote a set of observations by $\mathcal{O} \subseteq \{1, \ldots, n\}$. The expected cost $C(\mathcal{O})$ of observing attributes $\mathcal{O}$ is divided additively into two parts: the data acquisition cost $C_a(\mathcal{O})$, representing the cost of sensing these attributes, and the expected data transmission cost $C_t(\mathcal{O})$, measuring the communication cost required to download this data.

The acquisition cost $C_a(\mathcal{O})$ is deterministically given by the sum of the energy required to observe the attributes $\mathcal{O}$, as discussed in Section 2.5:

$$C_a(\mathcal{O}) = \sum_{i \in \mathcal{O}} C_a(i),$$

where $C_a(i)$ is the cost of observing attribute X_i.

The definition of the transmission cost $C_t(\mathcal{O})$ is somewhat trickier, as it depends on the particular data collection mechanism used to collect these observations from the network, and on the network topology. Furthermore, if the topology is unknown or changes over time, or if the communication links between nodes are unreliable, as in most sensor networks, this cost function becomes stochastic. For simplicity, we focus on networks with known topologies, but with unreliable communication. We address this reliability issue by introducing acknowledgment messages and retransmissions.

More specifically, we define our network graph by a set of edges $\mathcal{E}$, where each edge e_{ij} is associated two link quality estimates, p_{ij} and p_{ji}, indicating the probability that a packet from i will reach j and vice versa. With the simplifying assumption that these probabilities are independent, the expected number of transmission and acknowledgment messages required to guarantee a successful transmission between i and j is $\frac{1}{p_{ij}p_{ji}}$. We can now use these simple values to estimate the expected transmission cost.

There are many possible mechanisms for traversing the network and collecting this data. We focus on simply choosing a single path through the network that visits all sensors that observe attributes in $\mathcal{O}$ and returns to the base station. Clearly, choosing the best such path is an instance of the *traveling salesman problem*, where the graph is given by the edges $\mathcal{E}$ with weights $\frac{1}{p_{ij}p_{ji}}$. Although this problem is NP-complete, we can use well-known heuristics, such as k-OPT [19], that are known to perform very well in practice. We thus define $C_t(\mathcal{O})$ to be the expected cost of this (suboptimal) path, and our expected total cost for observing $\mathcal{O}$ can now be obtained by $C(\mathcal{O}) = C_a(\mathcal{O}) + C_t(\mathcal{O})$.

4.2 Improvement in confidence

Observing attributes $\mathcal{O}$ should improve the confidence of our posterior density. That is, after observing these attributes, we should be able to answer our query with more certainty[6]. For a particular value $\mathbf{o}$ of our observations $\mathcal{O}$, we can compute the posterior density $p(X_1, \ldots, X_n \mid \mathbf{o})$ and estimate our confidence as described in Section 3.1.

More specifically, suppose that we have a range query $X_i \in [a_i, b_i]$, we can compute the benefit $R_i(\mathbf{o})$ of observing the specific value $\mathbf{o}$ by:

$$R_i(\mathbf{o}) = \max\left[P(X_i \in [a_i, b_i] \mid \mathbf{o}), 1 - P(X_i \in [a_i, b_i] \mid \mathbf{o})\right],$$

that is, for a range query, $R_i(\mathbf{o})$ simply measures our confidence after observing $\mathbf{o}$. For value and average queries, we define the benefit by $R_i(\mathbf{o}) = P(X_i \in [\bar{x}_i - \varepsilon, \bar{x}_i + \varepsilon] \mid \mathbf{o})$, where $\bar{x}_i$ in this formula is the posterior mean of X_i given the observations $\mathbf{o}$.

However, the specific value $\mathbf{o}$ of the attributes $\mathcal{O}$ is not known *a priori*. We must thus compute the *expected benefit* $R_i(\mathcal{O})$:

$$R_i(\mathcal{O}) = \int p(\mathbf{o}) R_i(\mathbf{o}) d\mathbf{o}. \quad (3)$$

This integral may be difficult to compute in closed-form, and we may need to estimate $R_i(\mathcal{O})$ using numerical integration.

[6]This is not true in all cases; for range predicates, the confidence in the answer may *decrease* after an observation, depending on the observed value.

Example 4.1 *The descriptions in Examples 3.1-3.3 describe how the benefits $R_i(\mathbf{o})$ can be computed for a particular observed value $\mathbf{o}$ in the Gaussian models used in BBQ. For general range queries, even with Gaussians, we need to use numerical integration techniques to estimate the expected reward $R_i(\mathcal{O})$ in Equation (3).*

However, for value and AVERAGE queries we can compute this expression in closed-form, by exploiting the fact described in Example 3.1 that the posterior covariance $\Sigma_{\mathbf{Y}|\mathcal{O}}$ does not depend on the observed value $\mathbf{o}$. Note that for these queries, we are computing the probability that the true value deviates by more than ϵ from the posterior mean value. This probability is equal to the probability that a zero mean Gaussian, with covariance $\Sigma_{\mathbf{Y}|\mathcal{O}}$, deviates by more than ϵ from 0. This probability can be computed using the error function (erf) and the covariance matrix $\Sigma_{\mathbf{Y}|\mathcal{O}}$. Thus, for value and AVERAGE queries $R_i(\mathcal{O}) = R_i(\mathbf{o}), \forall \mathbf{o}$, allowing us to compute Equation (3) in closed-form. □

More generally, we may have range or value queries over multiple attributes. Semantically, we define this type of query as trying to achieve a particular marginal confidence over each attribute. We must thus decide how to trade off confidences between different attributes. For a query over attributes $\mathcal{Q} \subseteq \{1, \ldots, n\}$, we can, for instance, define the total benefit $R(\mathbf{o})$ of observing value $\mathbf{o}$ as either the minimum benefit over all attributes, $R(\mathbf{o}) = \min_{i \in \mathcal{Q}} R_i(\mathbf{o})$, or the average, $R(\mathbf{o}) = \frac{1}{|\mathcal{Q}|} \sum_{i \in \mathcal{Q}} R_i(\mathbf{o})$. In this paper, we focus on minimizing the total number of mistakes made by the query processor, and use the average benefit to decide when to stop observing new attributes.

4.3 Optimization

In the previous sections, we defined the expected benefit $R(\mathcal{O})$ and cost $C(\mathcal{O})$ of observing attributes $\mathcal{O}$. Of course, different sets of observed attributes will lead to different benefit and cost levels. Our user will define a desired confidence level $1 - \delta$. We would like to pick the set of attributes $\mathcal{O}$ that meet this confidence at a minimum cost:

$$\begin{array}{ll} \text{minimize}_{\mathcal{O} \subseteq \{1,\ldots,n\}} & C(\mathcal{O}), \\ \text{such that} & R(\mathcal{O}) \geq 1 - \delta. \end{array}$$

This general optimization problem is known to be NP-hard. Thus, efficient and exact optimization algorithms are unlikely to exist (unless P=NP).

We have developed two algorithms for solving this optimization problem. The first algorithm exhaustively searches over the possible subsets of possible observations, $\mathcal{O} \subseteq \{1, \ldots, n\}$. This algorithm can thus find the optimal subset of attributes to observe, but has an exponential running time.

The second algorithm uses a greedy incremental heuristic. We initialize the search with an empty set of attributes, $\mathcal{O} = \emptyset$. At each iteration, for each attribute X_i that is not in our set ($i \notin \mathcal{O}$), we compute the new expect benefit $R(\mathcal{O} \cup i)$ and cost $C(\mathcal{O} \cup i)$. If some set of attributes $\mathcal{G}$ reach the desired confidence, (*i.e.*, for $j \in \mathcal{G}$, $R(\mathcal{O} \cup j) \geq 1 - \delta$), then, among the attributes in $\mathcal{G}$, we pick the one with lowest total cost $C(\mathcal{O} \cup j)$, and terminate the search returning $\mathcal{O} \cup j$. Otherwise, if $\mathcal{G} = \emptyset$, we have not reached our desired confidence, and we simply add the attribute with the highest benefit over

cost ratio to our set of attributes:

$$\mathcal{O} = \mathcal{O} \cup \left(\arg\max_{j \notin \mathcal{O}} \frac{R(\mathcal{O} \cup j)}{C(\mathcal{O} \cup j)} \right).$$

This process is then repeated until the desired confidence is reached.

5 Experimental results

In this section, we measure the performance of BBQ on several real world data sets. Our goal is to demonstrate that BBQ provides the ability to efficiently execute approximate queries with user-specifiable confidences.

5.1 Data sets

Our results are based on running experiments over two real-world data sets that we have collected during the past few months using TinyDB. The first data set, *garden*, is a one month trace of 83,000 readings from 11 sensors in a single redwood tree at the UC Botanical Garden in Berkeley. In this case, sensors were placed at 4 different altitudes in the tree, where they collected collected light, humidity, temperature, and voltage readings once every 5 minutes. We split this data set into non-overlapping training and test data sets (with 2/3 used for training), and build the model on the training data.

The second data set, *lab*, is a trace of readings from 54 sensors in the Intel Research, Berkeley lab. These sensors collected light, humidity, temperature and voltage readings, as well as network connectivity information that makes it possible to reconstruct the network topology. Currently, the data consists of 8 days of readings; we use the first 6 days for training, and the last 2 for generating test traces.

5.2 Query workload

We report results for two sets of query workloads:

Value Queries: The main type of queries that we anticipate users would run on a such a system are queries asking to report the sensor readings at all the sensors, within a specified error bound ϵ with a specified confidence δ, indicating that no more than a fraction $1-\delta$ of the readings should deviate from their true value by ϵ. As an example, a typical query may ask for temperatures at all the sensors within 0.5 degrees with 95% confidence.

Predicate Queries: The second set of queries that we use are selection queries over the sensor readings where the user asks for all sensors that satisfy a certain predicate, and once again specifies a desired confidence δ.

We also looked at *average queries* asking for averages over the sensor readings. Due to space constraints, we do not present results for these queries.

5.3 Comparison systems

We compare the effectiveness of BBQ against two simple strategies for answering such queries :

TinyDB-style Querying: In this model, the query is disseminated into the sensor network using an overlay tree structure [22], and at each mote, the sensor reading is observed. The results are reported back to the base station using the same tree, and are combined along the way back to minimize communication cost.

Approximate-Caching: The base-station maintains a view of the sensor readings at all motes that is guaranteed to be within a certain interval of the actual sensor readings by requiring the motes to report a sensor reading to the base-station if the value of the sensor falls outside this interval. Note that, though this model saves communication cost by not reporting readings if they do not change much, it does not save acquisition costs as the motes are required to observe the sensor values at every time step. This approach is inspired by work by Olston *et al.* [24].

5.4 Methodology

BBQ is used to build a model of the training data. This model includes a transition model for each hour of the day, based on Kalman filters described in Example 3.3 above. We generate traces from the test data by taking one reading randomly from each hour. We issue one query against the model per hour. The model computes the *a priori* probabilities for each predicate (or ϵ bound) being satisfied, and chooses one or more additional sensor readings to observe if the confidence bounds are not met. After executing the generated observation plan over the network (at some cost), BBQ updates the model with the observed values from the test data and compares predicted values for non-observed readings to the test data from that hour.

To measure the accuracy of our prediction with value queries, we compute the average number of mistakes (per hour) that BBQ made, *i.e.*, how many of the reported values are further away from the actual values than the specified error bound. To measure the accuracy for predicate queries, we compute the number of predicates whose truth value was incorrectly approximated.

For TinyDB, all queries are answered "correctly" (as we are not modeling loss). Similarly, for approximate caching, a value from the test data is reported when it deviates by more than ϵ from the last reported value from that sensor, and as such, this approach does not make mistakes either

We compute a cost for each observation plan as described above; this includes both the attribute acquisition cost and the communications cost. For most of our experiments, we measure the accuracy of our model at predicting temperature.

5.5 Garden dataset: Value-based queries

We begin by analyzing the performance of value queries on the *garden* data set in detail to demonstrate the effectiveness of our architecture. The query we use for this experiment requires the system to report the temperatures at all motes to within a specified epsilon, which we vary. In these experiments we keep confidence constant at 95%, so we expect to see no more than 5% errors. Figure 4 shows the relative cost and number of errors made for each of the three systems. We varied epsilon from between 0 and 1 degrees Celsius; as expected, the cost of BBQ (on the left of the figure) falls rapidly as epsilon increases, and the percentage of errors (shown on the right) stays well below the specified confidence threshold of 5% (shown as the horizontal line). Notice that for reasonable values of epsilon, BBQ uses significantly less communication than approximate caching or TinyDB, sometimes by an order of magnitude. In this case, approximate caching always reports the value to within epsilon, so it does not make

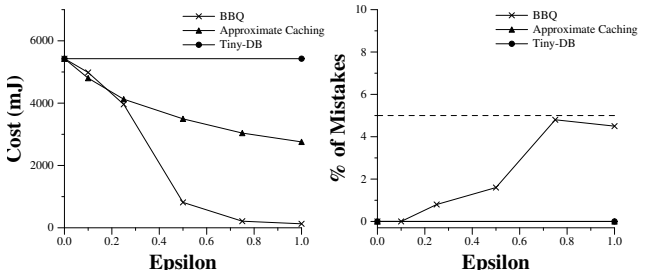

Figure 4: Figure illustrating the relative costs of BBQ versus TinyDB and Approximate Queries, with varying epsilons and a confidence interval of 95%.

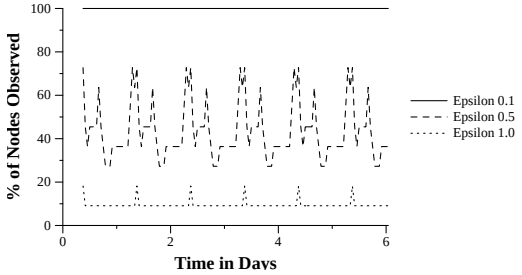

Figure 5: Figure showing the number of sensors observed over time for varying epsilons.

"mistakes", although the average observation error in approximate caching is close to BBQ (for example, in this experiment, with epsilon=.5, approximate caching has a root-mean-squared error of .46, whereas BBQ this error is .12; in other cases the relative performance is reversed).

Figure 5 shows the percentage of sensors that BBQ observes by hour, with varying epsilon, for the same set of garden experiments. As epsilon gets small (less than .1 degrees), it is necessary to observe all nodes on every query, as the variance between nodes is high enough that it cannot infer the value of one sensor from other sensor's readings with such accuracy. On the other hand, for epsilons 1 or larger, very few observations are needed, as the changes in one sensor closely predict the values of other sensors. For intermediate epsilons, more observations are needed, especially during times when sensor readings change dramatically. The spikes in this case correspond to morning and evening, when temperature changes relatively quickly as the sun comes up or goes down (hour 0 in this case is midnight).

5.6 Garden Dataset: Cost vs. Confidence

For our next set of experiments, we again look at the garden data set, this time comparing the cost of plan execution with confidence intervals ranging from 99% to 80%, with epsilon again varying between 0.1 and 1.0. The results are shown in Figure 6(a) and (b). Figure 6(a) shows that decreasing confidence intervals substantially reduces the energy per query, as does decreasing epsilon. Note that for a confidence of 95%, with errors of just .5 degrees C, we can reduce expected per-query energy costs from 5.4 J to less than 150 mJ – a factor of 40 reduction. Figure 6(b) shows that we meet or exceed our confidence interval in almost all cases (except 99% confidence). It is not surprising that we occasionally fail to satisfy these bounds by a small amount, as variances in our training data are somewhat different than variances in our test data.

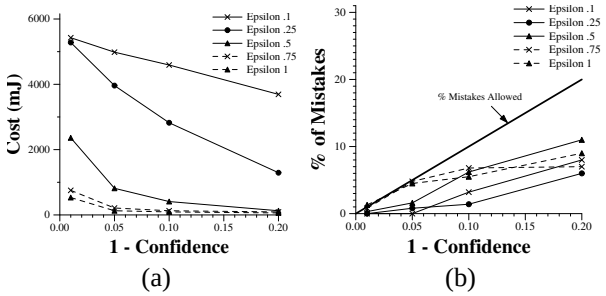

Figure 6: Energy per query (a) and percentage of errors (b) versus confidence interval size and epsilon.

We also ran experiments comparing (1) the performance of the greedy algorithm vs. the optimal algorithm, and (2) the performance of the dynamic (Kalman Filter) model that we use vs. a static model that does not incorporate observations made in the previous time steps into the model. As expected, the greedy algorithm performs slightly worse that the optimal algorithm, whereas using dynamic models results in less observations than using static models. Due to space constraints, we omit those experiments from this paper.

5.7 Garden Dataset: Range queries

We ran a number of experiments with range queries (also over the *garden* data set). Figure 7 summarizes the average number of observations required for a 95% confidence with three different range queries (temperature in [17,18], temperature in [19,20], and temperature in [20,21]). In all three cases, the actual error rates were all at or below 5% (ranging from 1.5-5%). Notice that different range queries require observations at different times – for example, during the set of readings just before hour 50, the three queries make observations during three disjoint time periods: early in the morning and late at night, the model must make lots of observations to determine whether the temperature is in the range 16-17, whereas during mid-day, it is continually making observations for the range 20-21, but never for other ranges (because it can be sure the temperature is above 20 degrees, but not 21!)

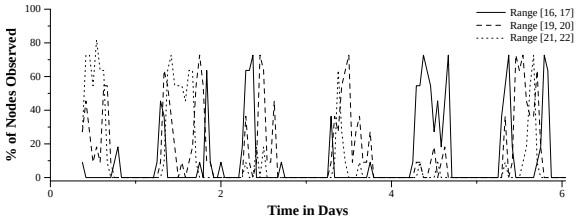

Figure 7: Graph showing BBQ's performance on three different range queries, for the garden data set with confidence set to 95%.

5.8 *Lab* dataset

We also ran similar experiments on the *lab* dataset, which because of the higher number of attributes in it, is a more interesting dataset. Contrary to our initial expectation, temperatures in the lab are actually harder to predict compared to the outdoors; human intervention (in particular, turning the air conditioning on and off) introduces a lot of randomness in this data. We report one set of experiments for this dataset, but defer a more detailed study to future work.

Figure 8(a) shows the cost incurred in answering a value query on this dataset, as the confidence bound is varied. For comparative purposes, we also plot the cost of answering the query using TinyDB. Once again, we see that the as the required confidence in answer drops, BBQ is able to answer the query more efficiently, and is significantly more cost-efficient than TinyDB for larger error bounds. Figure 8(b) shows that BBQ was able to achieve the specified confidence bounds in almost all the cases.

Figure 9 shows an example traversal generated by executing a value based query with confidence of 99% and epsilon of .5 degrees C over the *lab* data. The two paths shown are amongst the longer paths generated – one is the initial set of observations needed to improve the model's confidence, and the other is a traversal at 8am just as the day is starting to warm up.

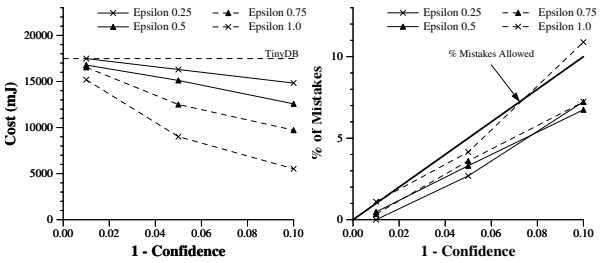

Figure 8: Energy per query (a) and percentage of errors (b) versus confidence interval size and epsilon for the Lab Data.

6 Extensions and future directions

In this paper, we focused on the core architecture for unifying probabilistic models with declarative queries. In this section we outline several possible extensions.

Conditional plans: In our current prototype, once an observation plan has been submitted to the sensor network, it is executed to completion. A simple alternative would be to generate plans that include early stopping conditions; a more sophisticated approach would be to generate conditional plans that explore different parts of the network depending on the values of observed attributes. We have begun exploring such conditional plans in a related project [8].

More complex models: In particular, we are interested in building models that can detect faulty sensors, both to answer fault detection queries, and to give correct answers to general queries in the presence of faults. This is an active research topic in the machine learning community (*e.g.*, [18]), and we expect that these techniques can be extended to our domain.

Outliers: Our current approach does not work well for outlier detection. To a first approximation, the only way to detect outliers is to continuously sample sensors, as outliers are fundamentally uncorrelated events. Thus, any outlier detection scheme is likely to have a high sensing cost, but we expect that our probabilistic techniques can still be used to avoid excessive communication during times of normal operation, as with the fault detection case.

Support for dynamic networks: Our current approach of re-evaluating plans when the network topology changes will not work well in highly dynamic networks. As a part

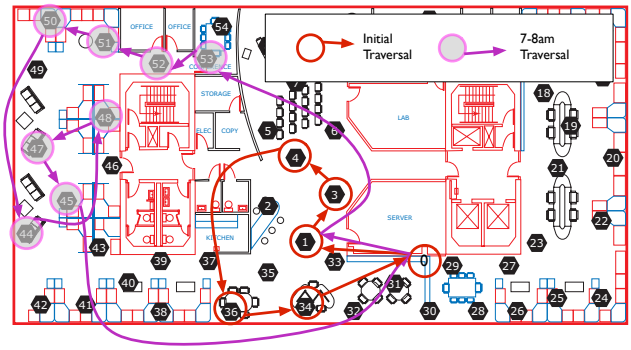

Figure 9: Two traversals of the lab network. An observation is made at every circle. The query was a value-query over all 54 sensors with $\epsilon = .1$ and $\delta = .99$. These paths correspond to times when the model's variance is high – e.g., when the system is first started, and at around 8am when the sun begins to heat the lab, so many observations are needed. For other hours, very few observations are needed.

of our instrumentation of our lab space, we are beginning a systematic study of how network topologies change over time and as new sensors are added or existing sensors move. We plan to use this information to extend our exploration plans with simple topology change recovery strategies that can be used to find alternate routes through the network.

Continuous queries: Our current approach re-executes an exploration plan that begins at the network root on every query. For continuous queries that repeatedly request data about the same sensors, it may be possible to install code in the network that causes devices to periodically push readings during times of high change (*e.g.*, every morning at 8 am).

7 Related work

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 [12, 10, 11] 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 with BBQ, such answers typically include tight bounds on the correctness of answers. AQUA, however, is designed to work in an environment where it is possible to generate an independent random sample of data (something that is quite tricky to do in sensor networks, as losses are correlated and communicating random samples may require the participation of a large part of the network). AQUA also does not exploit correlations, which means that it lacks the *predictive* power of representations based on probabilistic models. [7, 9] propose exploiting data correlations through use of graphical model techniques for approximate query processing, but neither provide any guarantees in the answers returned. Recently, Considine *et al.* have shown that sketch based approximation techniques can be applied in sensor networks [17].

Work on approximate caching by Olston *et al.*, is also related [25, 24], in the sense that it provides a bounded approximation of the values of a number of cached objects (sensors, in our case) at some server (the root of the sensor network). The basic idea is that the server stores cached values along

with absolute bounds for the deviation of those values; when objects notice that their values have gone outside the bounds known to be stored at the server, they send an update of our value. Unlike our approach, this work requires the cached objects to continuously monitor their values, which makes the energy overhead of this approach considerable. It does, however, enable queries that detect outliers, something BBQ currently cannot do.

There has been some recent work on approximate, probabilistic querying in sensor networks and moving object databases [3]. This work builds on the work by Olston *et al.* in that objects update cached values when they exceed some boundary condition, except that a pdf over the range defined by the boundaries is also maintained to allow queries that estimate the most likely value of a cached object as well as an confidence on that uncertainty. As with other approximation work, the notion of correlated values is not exploited, and the requirement that readings be continuously monitored introduces a high sampling overhead.

Information Driven Sensor Querying (IDSQ) from Chu *et al.* [4] uses probabilistic models for estimation of target position in a tracking application. In IDSQ, sensors are tasked in order according to maximally reduce the positional uncertainty of a target, as measured, for example, by the reduction in the principal components of a 2D Gaussian.

Our prior work presented the notion of *acquisitional query processing*(ACQP) [21] – that is, query processing in environments like sensor networks where it is necessary to be sensitive to the costs of acquiring data. The main goal of an ACQP system is to avoid unnecessary data acquisition. The techniques we present are very much in that spirit, though the original work did not attempt to use probabilistic techniques to avoid acquisition, and thus cannot directly exploit correlations or provide confidence bounds.

BBQ is also inspired by prior work on Online Aggregation [14] and other aspects of the CONTROL project [13]. The basic idea in CONTROL is to provide an interface that allows users to see partially complete answers with confidence bounds for long running aggregate queries. CONTROL did not attempt to capture correlations between the different attributes, such that observing one attribute had no effect on the systems confidence on any of the other predicates.

The probabilistic querying techniques described here are built on standard results in machine learning and statistics (*e.g.*, [27, 23, 5]). The optimization problem we address is a generalization of the *value of information* problem [27]. This paper, however, proposes and evaluates the first general architecture that combines model-based approximate query answering with optimizing the data gathered in a sensornet.

8 Conclusions

In this paper, we proposed a novel architecture for integrating a database system with a correlation-aware probabilistic model. Rather than directly querying the sensor network, we build a model from stored and current readings, and answer SQL queries by consulting the model. In a sensor network, this provides a number of advantages, including shielding the user from faulty sensors and reducing the number of expensive sensor readings and radio transmissions that the network must perform. Beyond the encouraging, order-of-magnitude reductions in sampling and communication cost offered by BBQ, we see our general architecture as the proper platform for answering queries and interpreting data from real world environments like sensornets, as conventional database technology is poorly equipped to deal with lossiness, noise, and non-uniformity inherent in such environments.

References

[1] IPSN 2004 Call for Papers. `http://ipsn04.cs.uiuc.edu/call_for_papers.html`.

[2] SenSys 2004 Call for Papers. `http://www.cis.ohio-state.edu/sensys04/`.

[3] R. Cheng, D. V. Kalashnikov, and S. Prabhakar. Evaluating probabilistic queries over imprecise data. In *SIGMOD*, 2003.

[4] M. Chu, H. Haussecker, and F. Zhao. Scalable information-driven sensor querying and routing for ad hoc heterogeneous sensor networks. In *Journal of High Performance Computing Applications.*, 2002.

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

[6] Crossbow, Inc. Wireless sensor networks. `http://www.xbow.com/Products/Wireless_Sensor_Networks.htm`.

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

[8] A. Desphande, C. Guestrin, W. Hong, and S. Madden. Exploiting correlated attributes in acquisitional query processing. Technical report, Intel-Research, Berkeley, 2004.

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

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

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

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

[13] J. M. Hellerstein, R. Avnur, A. Chou, C. Hidber, C. Olston, V. Raman, T. Roth, and P. J. Haas. Interactive data analysis with CONTROL. *IEEE Computer*, 32(8), August 1999.

[14] J. M. Hellerstein, P. J. Haas, and H. Wang. Online aggregation. In *SIGMOD*, pages 171–182, Tucson, AZ, May 1997.

[15] C. Intanagonwiwat, R. Govindan, and D. Estrin. Directed diffusion: A scalable and robust communication paradigm for sensor networks. In *MobiCOM*, Boston, MA, August 2000.

[16] Intersema. Ms5534a barometer module. Technical report, October 2002. `http://www.intersema.com/pro/module/file/da5534.pdf`.

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

[18] 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.

[19] S. Lin and B. Kernighan. An effective heuristic algorithm for the tsp. *Operations Research*, 21:498–516, 1971.

[20] S. Madden. The design and evaluation of a query processing architecture for sensor networks. Master's thesis, UC Berkeley, 2003.

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

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

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

[24] C. Olston and J.Widom. Best effort cache sychronization with source cooperation. *SIGMOD*, 2002.

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

[26] G. Pottie and W. Kaiser. Wireless integrated network sensors. *Communications of the ACM*, 43(5):51 – 58, May 2000.

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

[28] Sensirion. Sht11/15 relative humidity sensor. Technical report, June 2002. `http://www.sensirion.com/en/pdf/Datasheet_SHT1x_SHT7x_0206.pdf`.

[29] TAOS, Inc. Tsl2550 ambient light sensor. Technical report, September 2002. `http://www.taosinc.com/pdf/tsl2550-E39.pdf`.

[30] Y. Yao and J. Gehrke. Query processing in sensor networks. In *Conference on Innovative Data Systems Research (CIDR)*, 2003.