VLDB Journal manuscript No.
(will be inserted by the editor)

Model-based Approximate Querying in Sensor Networks*

Amol Deshpande[1], **Carlos Guestrin**[2], **Samuel R. Madden**[3], **Joseph M. Hellerstein**[4,5], **Wei Hong**[4]

[1] University of Maryland at College Park
[2] Carnegie Mellon University
[3] Massachusetts Institute of Technology
[4] Intel Research, Berkeley
[5] University of California, Berkeley

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 article, 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.

* We thank Intel Corporation and Intel Research – Berkeley for supporting the authors' during much of this work. This article includes and extends results that were previously published in VLDB 2004 [18], and combines these techniques with the conditional planning approach published in ICDE 2005 [17].

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 (*e.g.*, [39, 56, 35]). 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.

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 [56, 39] 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 article, 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, a 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 $near$.

In this paper, we propose and evaluate two classes of observation plans: (1) *subset* observation plans, where the query processor specifies a subset of the attributes to be observed, along with a *traversal route* to be used for acquiring the values of these attributes, and (2) *branching (conditional)* observation plans, where the set of attributes to be observed next in the traversal depends on the actual values of the attributes observed thus far. We develop and evaluate algorithms to efficiently choose both classes of observation plans.

To explore the benefits of the model-based querying approach we propose, we have built 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*

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

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.*, [41]).

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 to the query. In the process of generating these estimates, the model may interrogate the sensor network for updated readings that will help to refine estimates for which the model's 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.*, [41]).

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.

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:** 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 [14] that we

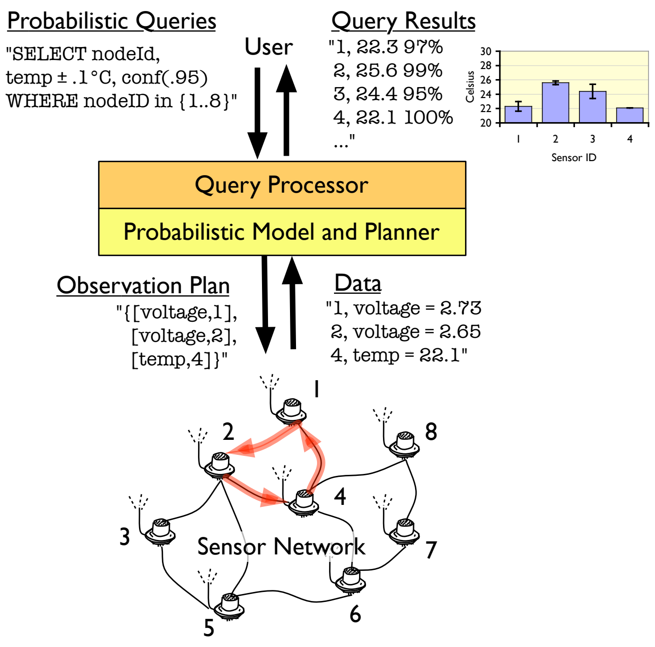


Fig. 1 Our architecture for model-based querying in sensor networks.

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 [14], 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.

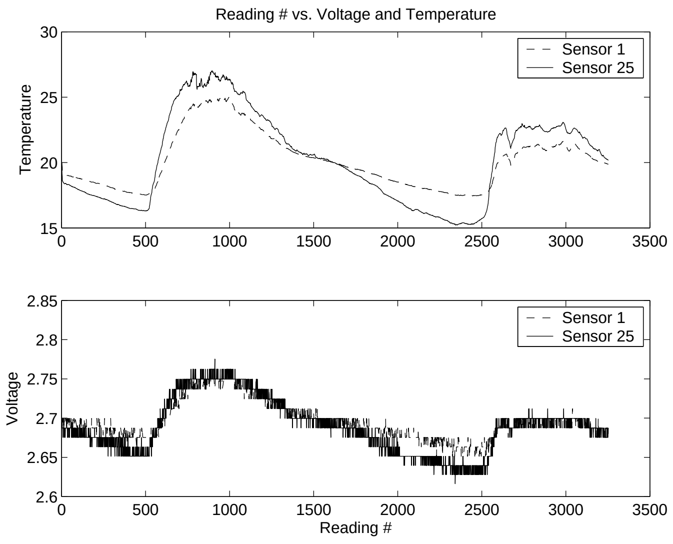


Fig. 2 Trace of voltage and temperature readings over a two day period from a single mote-based sensor. Notice the close correlation between the two attributes.

Sensor	**Energy Per Sample (@3V), mJ**
Solar Radiation [55]	.525
Barometric Pressure [33]	0.003
Humidity and Temperature[52]	0.5
Voltage	0.00009

Table 1 Summary of Power Requirements of Crossbow MTS400 Sensorboard (From [38]). 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.

2.1 Confidence intervals and correlation models

If the user in Figure 1 requested 100% confidence and no error tolerance, our model-based approach would most likely require us to interrogate every sensor. The returned result may 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). The confidence intervals computed from our probabilistic model after incorporating the available sensor values provide considerably more information than traditional sensor network systems like TinyDB [40] and Cougar [56] provide in this setting. With those systems, the user would simply get no data regarding those missing times and locations.

Conversely, if the user requested very wide confidence bounds, our model-based approach may be 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.*, [41]), 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, $X_1, \dots, X_d$ 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 article, 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).

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 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 the 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 discuss this optimization problem in more detail in Section 4.

[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.

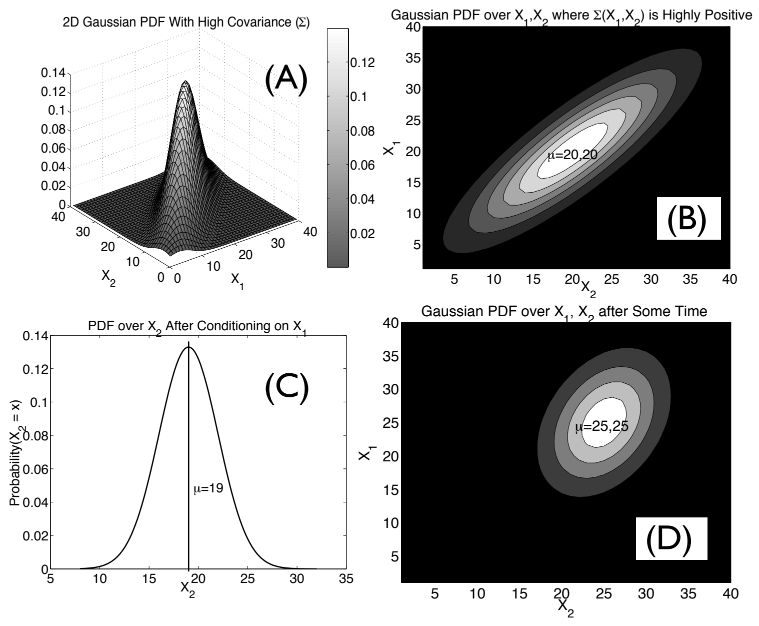


Fig. 3 Example of Gaussians: (a) 3D plot of a 2D Gaussian with high covariance; (b) the same Gaussian viewed as a contour plot; (c) the resulting Gaussian over X_2 after a particular value of X_1 has been observed; finally, (d) shows how, as uncertainty about X_1 increases from the time we last observed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

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 article, 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 7.

In this article, 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 7.

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 topology is relatively stable; highly dynamic topologies will need more sophisticated techniques, which is a problem we briefly discuss in Section 7.

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. When conditional observation plans are used, a plan may also contain additional branching information that lists the plans to be used for evaluating the rest of the query for different observed values of the attributes. 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. We note that this approach to routing is substantially different than approaches used in other sensor network databases (*e.g.*, Cougar [56] and TinyDB [40]) that use (scoped) flooding approaches to disseminate queries throughout a network. We adopted a tour-based approach because it allows us to initiate data collection from the root and visit only a small, predefined subset of the nodes.

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 [32, 49], 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 [14] 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

[3] In practice [48], 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

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 article, we primarily focus on temperature readings, though we briefly discuss other attributes as well in Section 6. 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.

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 7.

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:

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

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).

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

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

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 $p(X_1, \ldots, X_n)$ 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

[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: $p(x|y) = \frac{p(y|x)p(x)}{p(y)}$.

$p(\mathbf{Y} \mid \mathbf{o})$ over the remaining attributes are given by:

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

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 $\mathcal{A}$:

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

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 c, we can define a random variable Z:

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

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

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

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

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:*

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

Thus, the variance of Y is given by a weighted sum of the 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 6, 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 6, 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.

This section focuses on the case where our plans are *sequential* – that is, the order in which attributes are observed is fixed at the time the observation plan is generated at the base station. In Section 5 we describe our approach to generating *conditional plans* that evaluate different predicates and predicate orderings depending on the values observed as the plan is executed in the network.

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 [37], 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* $R_i(\mathcal{O})$:

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

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

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

[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.

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. $\square$

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 article, 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:

$$\begin{array}{ll} \text{minimize}_{\mathcal{O} \subseteq \{1,\ldots,n\}} & C(\mathcal{O}), \\ \text{such that} & R(\mathcal{O}) \geq 1 - \delta. \end{array} \tag{4}$$

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 Conditional Observation Plans

In this section, we extend the definition of observation plans introduced in the previous section to allow the possibility of *conditional plans* – that is, plans in which a different subset of nodes is observed depending on the value of a particular attribute at a particular node. This has the potential to significantly reduce the number of variables that must be observed; some examples include:

– Suppose we are evaluating a range query with a high *a priori* probability of being true for a particular variable that is highly correlated with many other variables. If that variable is observed to be very far outside the queried range, we may be able to infer that all other variables in the query will also be outside the range.
– If two sensors s_1 and s_2 are uncorrelated when a given sensor s is in a particular range r, but otherwise relatively highly correlated, we may only need to observe both of them if s is within r. Otherwise, just observing s_1 may be sufficient. Examples of this occur in our lab deployment when one sensor has sun shining on it, causing its light value to be unusually high. The temperature readings on such bright nodes are often much higher than the temperature readings on other sensors in the lab that they are otherwise correlated with.

Conditional plans, which we denote $\mathcal{B}$, consist of a tree of nodes; at each node i, we observe the value of an attribute A_i and then execute one of two possible sub-plans depending on its value. Each node i in the plan runs at some physical sensor which has access to the value of A_i. We allow for multiple split points over the same attribute.

From an architectural perspective, *conditional planning* requires only a few small changes to the design of the system. These changes are entirely confined to the portion of BBQ that runs within the sensor network: the external interface continues to function just as it did before. Figure 4 shows how the architecture differs: notice that the top layers of the system are the same, but that at the lower levels plans may now have branches in them. Plan execution works very much as in the previous sections: once a plan is generated, it is sent to the first node in the plan, which evaluates the first predicate and executes the appropriate sub-plan $\mathcal{B}'$ (based on the predicate's value), sending the query to the first node in $\mathcal{B}'$.

Conditional plans introduce several complications to the basic plan generation algorithm described in the previous section:

– They require a way to evaluate the cost of execution of a conditional plan, so that we can pick the best plan that satisfies the user specified confidence bounds and query. Plan cost should include *plan size* since our conditional plans may be quite large.
– They require a metric to evaluate the benefit of a particular conditional plan relative to other (conditional or non-conditional) plans.
– They require an algorithm to generate a conditional plan. This is substantially more complicated than algorithms for non-conditional plans described in the previous section because there are a huge number of possible split points at

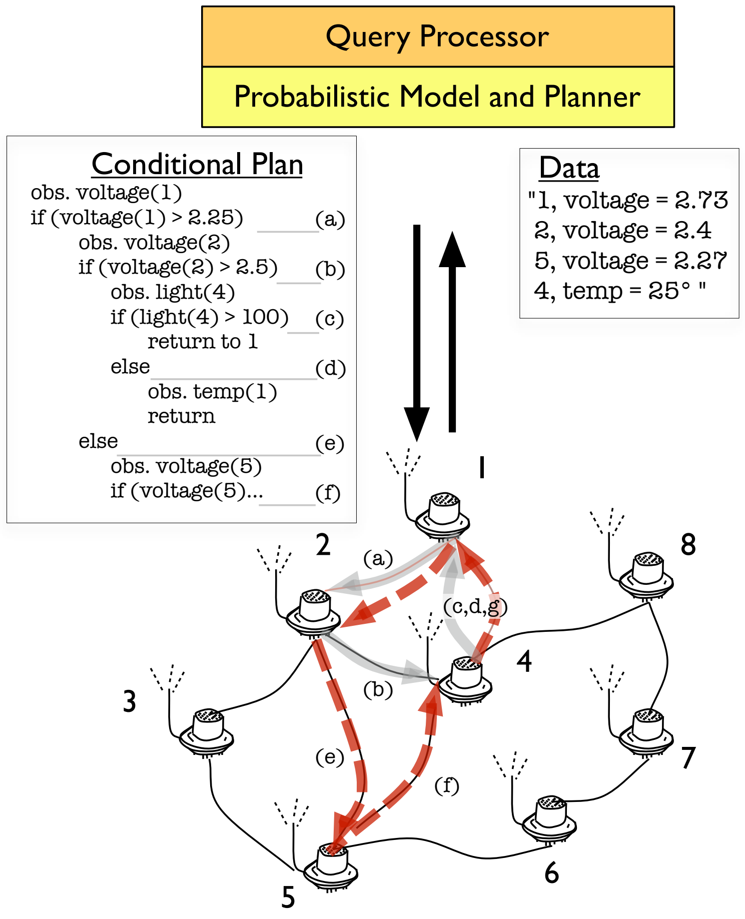


Fig. 4 Diagram illustrating a conditional execution plan in a sensor network. At each node, there are two possible paths that may be followed; the darker, dashed lines indicate the path that was actually executed. A portion of the conditional plan is shown being sent into the network; the decision points are labeled (a) through (f) and are shown at the appropriate node in the network traversal.

each step of the algorithm, and evaluating the benefit of a particular split is computationally expensive.

We discuss each of these three issues in turn in the next three sections.

5.1 Cost of a conditional plan

To introduce our approach to evaluating the cost of a conditional plan, we introduce some additional notation for representing plans. We choose to represent a plan, $\mathcal{B}$, as a simple binary decision tree[7], where each split point s_j specifies a binary *conditioning predicate*, $T_j(X_i \geq x_i)$, that splits the plan into two alternate conditional plans, $\mathcal{B}_{T_j(\mathbf{x})=\mathit{true}}$ and $\mathcal{B}_{T_j(\mathbf{x})=\mathit{false}}$, where $T_j(\mathbf{x})$ is the truth value of T_j on the tuple $\mathbf{x}$. Each conditioning predicate depends only on the value of a single attribute. In a centralized setting, at split point s_j the query processor would evaluate the predicate $T_j(\mathbf{x})$ and choose to execute one of these two subplans depending on the value of the predicate. In our distributed sensor network setting, we must assign a node *nodeOf*$[s_j]$ to each split point s_j, representing the network location where the sensor value used in the predicate in s_j can be retrieved. Note that, if, for a split point s_j, the attribute X from node i has already been observed at an earlier node in the plan, we do not need to visit node i to obtain the attribute value. In such cases, we set *nodeOf*$[s_j]$ to be the parent of s_j in the plan, denoted by *Parent*$[s_j]$.

In Section 4, we defined simple observation plans that traverse the network to collect the measurements $\mathcal{O}$. During execution of a conditional plan $\mathcal{B}$, we simply traverse the binary tree defined by $\mathcal{B}$, acquiring any attributes we need to evaluate the conditioning predicates. For a particular tuple $\mathbf{x}$, *i.e.*, an assignment to all attributes, our plan will traverse a single path from the root to a leaf of the binary tree $\mathcal{B}$. Analogously to Section 4, a traversal from the root to a leaf of the tree corresponds to a list of nodes to be visited. However, the specific traversal will depend on the specific choices made in the splitting points.

The cost of a specific traversal through a binary tree is the sum of the cost of the attributes that are acquired by plan $\mathcal{B}$ in this traversal, plus the incurred communication cost to visit these nodes in the network. Specifically, at each split point s_j in this traversal, if the attribute in the predicate T_j has already been acquired, then this split point has zero *atomic cost*. However, if the attribute X_i in T_j has not yet been acquired, then the atomic cost of this node is $C_a(i)$ plus the transmission cost $C_t(v, i)$, where v is *nodeOf*[*Parent*$[s_j]$]. The transmission cost $C_t(v, i)$ can be specified by the cost of the shortest path from v to i in our network graph, as in Section 4.1.[8] Note that the atomic cost of a leaf is only the cost of returning from the network node associated with this leaf to the base station, as no attributes are acquired at this point. For simplicity, we annotate each split point s_j of our plan with this atomic cost $C(s_j)$.

We can now formally define the *traversal cost* $C(\mathcal{B}, \mathbf{x})$ of applying plan $\mathcal{B}$ to the tuple $\mathbf{x}$ of the plan recursively by:

$$C(\mathcal{B}, \mathbf{x}) = C(\texttt{Root}(\mathcal{B})) + \begin{cases} 0, & \text{if } |\mathcal{B}| = 1; \\ C(\mathcal{B}_{T_j(\mathbf{x})}, \mathbf{x}), & \text{otherwise;} \end{cases} \tag{5}$$

where $\texttt{Root}(\mathcal{B})$ is the root split point of the tree for plan $\mathcal{B}$, $C(\texttt{Root}(\mathcal{B}))$ is the atomic cost of this root node as defined above, and $|\mathcal{B}| = 1$ indicates that we have reached a leaf and can stop the recursion.

[7] Our approach can, of course, be extended to more general decision trees, or acyclic decision diagrams [5].

[8] We can also consider the plan size when estimating these costs, as the decision tree is pruned every time an observation is made.

In this setting, planning is an optimization problem that involves searching the space of available conditional plans that satisfy the user's query for the plan $\mathcal{B}^*$ with minimal expected cost:

$$\begin{aligned}\mathcal{B}^* &= \arg\min_{\mathcal{B}} C(\mathcal{B}), \\ &= \arg\min_{\mathcal{B}} E_{\mathbf{x}}\left[C(\mathcal{B},\mathbf{x})\right], \\ &= \arg\min_{\mathcal{B}} \int_{\mathbf{x}} P(\mathbf{x})C(\mathcal{B},\mathbf{x})d\mathbf{x}. \end{aligned} \tag{6}$$

Using the recursive definition of the cost $C(\mathcal{B},\mathbf{x})$ of evaluating a tuple $\mathbf{x}$ in Equation (5), we can similarly specify a recursive definition of the expected cost $C(\mathcal{B})$ of a plan. For this recursion, we must specify, at each split point s_j of the plan, the conditional probability that the associated predicate T_j will be true or false, given the predicates evaluated thus far in the plan. We use $\mathbf{t}$ to denote an assignment to this set of predicates. Using this notation, the expected plan cost is given by:

$$C(\mathcal{B},\mathbf{t}) = C(\mathtt{Root}(\mathcal{B})) + \begin{cases} 0, & \text{if } |\mathcal{B}| = 1; \\ \begin{array}{l} P(T_j \mid \mathbf{t})C(\mathcal{B}_{T_j},\mathbf{t}\wedge T_j)+ \\ P(\neg T_j \mid \mathbf{t})C(\mathcal{B}_{\neg T_j},\mathbf{t}\wedge \neg T_j), \end{array} & \text{otherwise;} \end{cases} \tag{7}$$

where $C(\mathcal{B},\mathbf{t})$ is the expected cost of the (sub)plan $\mathcal{B}$ starting from its root split point $\mathtt{Root}(\mathcal{B})$, given that the predicates $\mathbf{t}$ have been observed. At this point, the expected cost depends on the value of the new predicate T_j. With probability $P(T_j \mid \mathbf{t})$, T_j will be true and we must solve the subproblem $C(\mathcal{B}_{T_j},\mathbf{t}\wedge T_j)$, *i.e.*, the subplan $\mathcal{B}_{T_j}$ after observing the original predicate values $\mathbf{t}$ and the new predicate value $T_j =$ *true*. Similarly, with probability $P(\neg T_j \mid \mathbf{t}) = 1 - P(T_j \mid \mathbf{t})$, T_j will be false and we must solve $C(\mathcal{B}_{\neg T_j},\mathbf{t}\wedge \neg T_j)$, *i.e.*, the subplan $\mathcal{B}_{\neg T_j}$ after observing $\mathbf{t}$ and the $T_j =$ *false*. As before, when we reach a leaf ($|\mathcal{B}| = 1$), the recursion stops. Now the expected cost of a plan $\mathcal{B}$ is defined using Equation (7) by $C(\mathcal{B},\emptyset)$.

Example 5.1 *Consider the simple example of exhaustively enumerating the plans for a query over three attributes, $\{X_1, X_2, X_3\}$, each with binary domain $\{1,2\}$. Our query, Q, in this example is simply $Q = (X_1 = 1 \wedge X_2 = 1)$. For sake of exposition, we will assume that the confidence required in the answer is 100% ($\delta = 0$), i.e., no error is tolerated. Figure 5 shows three possible plans in this enumeration; there are 12 total possible plans in this case*[9]*. Each node in this figure is labeled with the attribute acquired at that node, and the P_i values denote the conditional probability of the outcome along each edge occurring, given the outcomes already specified in the plan branch. Terminal outcomes are denoted as true or false, indicating that a tuple that reaches this node is output or rejected.*

[9] The number of plans is larger if 100% confidence is not desired, or if the probability distribution contains *zeroes*. For example, a plan that simply observes X_3 and stops might achieve the desired confidence level and might be a valid plan in some cases.

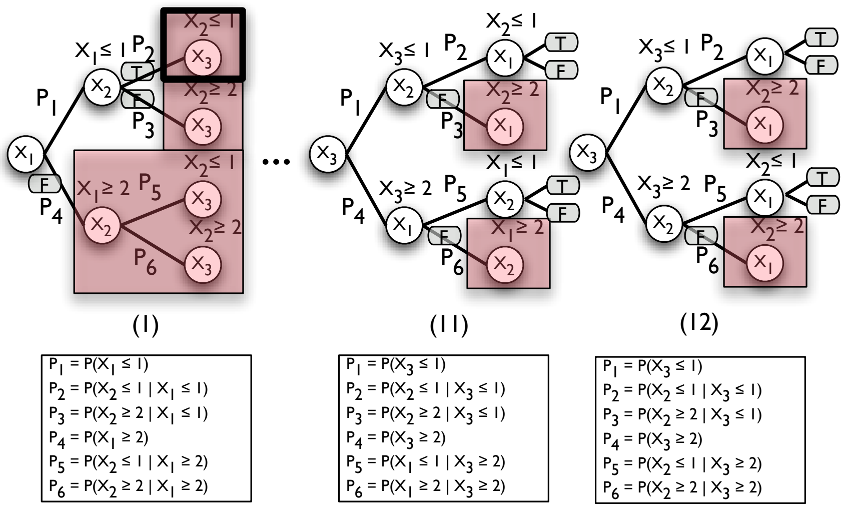


Fig. 5 Set of possible plans for the two predicate query $X_1 = 1 \wedge X_2 = 1$, for $\delta = 0$, with three attributes, X_1, X_2, and X_3 available for use in the query. The labels in the nodes indicate the attribute acquired at that point, and the labels on the edges denote the probability of the outcome along that edge. Each P_i expands into the conditional probability expansion given at the bottom of the figure. Terminal points in the tree are labeled with their outcome: *true* if the query is satisfied with probability $1 - \delta$ or *false* if it fails with probability $1 - \delta$. Grayed out regions do not need to be explored because the confidence in the truth value of the query is greater than $1 - \delta$ given the observations so far.

In general, if at some branch of the plan, our confidence in the truth value of $\mathcal{Q}$ is greater than $1-\delta$ we do not need to further expand the tree. We can evaluate this confidence using the techniques for determining whether a range query is satisfied described in Section 3.1, conditioning on the outcomes of earlier plan branches. For example, in Plan (11) in Figure 5, after $X_3 \leq 1$ has been observed, we can stop evaluating the plan if:

$$P(X_1 = 1 \wedge X_2 = 1 | X_3 \leq 1) \geq 1 - \delta$$

or if:

$$P(X_1 \neq 1 \vee X_2 \neq 1 | X_3 \leq 1) \geq 1 - \delta$$

For example, in Figure 5, the grayed-out regions represent branches of the plan that do not need to be evaluated since a query predicate has failed. The outlined box at the top of Plan (1) indicates that all query predicates have been evaluated on this branch, so X_3 does not need to be acquired.

Given the plans in Figure 5, it is straightforward to read off the expected cost as defined in Equation (7). For example, for Plan (11) the cost is:

$$\begin{aligned} C(\textit{Plan (11)}) = {} & C_3 + \\ & P(X_3 \leq 1)(C_2 + P(X_2 \leq 1 \mid X_3 \leq 1)C_1) + \\ & P(X_3 \geq 2)(C_1 + P(X_1 \leq 1 \mid X_3 \geq 2)C_2), \end{aligned}$$

where, for example, when branching on X_2, we do not need to consider the branch for the assignment $X_2 = 2$ as this assignment makes φ false and we replace the grayed-out box by a leaf with value false.

The key observation here is that the cheapest possible plan is not always the one that immediately acquires the values of the attributes in the query predicates. In our example, plan (12) could be cheaper than plan (1), if observing X_3 has low cost and dramatically skews the probabilities of the attributes X_1 and X_2. In particular, if $X_3 = 1$ increases the probability of $X_2 = 2$, then observing X_3 may allow us to select the particular attribute that is more likely to determine if $\varphi = $ false. Thus, if $X_3 = 1$, the query processor may avoid acquiring X_1, which it does first in plan (1). $\square$

5.2 Improvement in confidence with conditional planning

As was the case in Section 4, in addition to defining the cost of a conditional plan, we also need to measure the expected benefit $R(\mathcal{B})$. As in Section 4.2, we define the benefit of a plan $\mathcal{B}$ as our overall confidence that the truth value of the predicates in the user query will be predicted correctly.

During the execution of a conditional plan $\mathcal{B}$ for a query Q with $|Q|$ predicates, we will traverse a path from the root to a single leaf, l in the plan. We define our average benefit, $R(\mathcal{B}, \mathbf{o}_l)$ as our confidence in $\mathcal{B}$ being satisfied at l given a tuple of observations $\mathbf{o}_l$ up to that leaf. The value of $R(\mathcal{B}, \mathbf{o}_l)$ is simply the average probability that the truth value of any query predicate will be predicted correctly. More formally, this can be expressed as:

$$R(\mathcal{B}, \mathbf{o}_l) = \max[P(\mathcal{B}|\mathbf{o}_l), 1-P(\mathcal{B}|\mathbf{o}_l)] = \frac{1}{|Q|}\sum_{i \in Q} \max[P(Q_i|\mathbf{o}_l), 1 - P(Q_i|\mathbf{o}_l)]$$

Where Q_i is the ith predicate of Q, and $P(Q_i|\mathbf{o}_l)$ is the probability that predicate Q_i is true given the values of all attributes observed in $\mathbf{o}_l$.

As before, we need to generalize this equation to compute the expected benefit of $\mathcal{B}$ over all tuples. We can compute this simply by summing the expected benefit of every possible path p from root to leaf times the probability that a tuple traverses p, as follows.

The expected benefit of a path p terminating in leaf node l_p is the probability that the predicted truth value at l_p is correct given the value of the conditioning predicates observed along p. We use $\mathbf{t}_{l_p}$ to denote a complete assignment to the set of boolean predicates along a path. Then, we simply define $R_{\mathbf{t}_{l_p}}(\mathcal{B}, \mathbf{t}_{l_p})$ analogously to $R(\mathcal{B}, \mathbf{o})$, substituting observed truth values of the predicates, $\mathbf{t}_{l_p}$, for actual tuple values:

$$R_{\mathbf{t}_{l_p}}(\mathcal{B}, \mathbf{t}_{l_p}) = \frac{1}{|Q|}\sum_{i \in Q} \max[P(Q_i|\mathbf{t}_{l_p}), 1 - P(Q_i|\mathbf{t}_{l_p})]. \tag{8}$$

As in Section 4, computing the value of $P(Q_i|\mathbf{t}_{l_p})$ requires an integral over the set of all observations $\mathbf{o}_{\mathbf{t}_{l_p}}$ that satisfy $\mathbf{t}_{l_p}$, that is:

$$\int_{\mathbf{o}\in\mathbf{o}_{\mathbf{t}_{l_p}}} P(Q_i, \mathbf{o})d\mathbf{o}.$$

To compute the expected benefit over all paths, we can recursively sum the probability of each path times its benefit, as in Equation (7):

$$R(\mathcal{B}, \mathbf{t}) = \begin{cases} R_{\mathbf{t}}(\mathcal{B}, \mathbf{t}), & \text{if } |\mathcal{B}| = 1; \\ P(T_j \mid \mathbf{t})R(\mathcal{B}_{T_j}, \mathbf{t} \wedge T_j) + & \text{otherwise;} \\ P(\neg T_j \mid \mathbf{t})R(\mathcal{B}_{\neg T_j}, \mathbf{t} \wedge \neg T_j), & \end{cases} \tag{9}$$

where $R(\mathcal{B}, \mathbf{t})$ is the expected benefit of the (sub)plan $\mathcal{B}$ starting from its root, given that the predicates $\mathbf{t}$ have been observed. At this point, the expected benefit depends on the value of the first predicate T_j in $\mathcal{B}$. With probability $P(T_j \mid \mathbf{t})$, T_j will be true and we must solve the subproblem $R(\mathcal{B}_{T_j}, \mathbf{t} \wedge T_j)$, *i.e.*, the subplan $\mathcal{B}_{T_j}$ after observing the original predicate values $\mathbf{t}$ and the new predicate value $T_j =$ *true*. Similarly, with probability $P(\neg T_j \mid \mathbf{t}) = 1 - P(T_j \mid \mathbf{t})$, T_j will be false and we must solve $R(\mathcal{B}_{\neg T_j}, \mathbf{t} \wedge \neg T_j)$, *i.e.*, the subplan $\mathcal{B}_{\neg T_j}$ after observing $\mathbf{t}$ and the $T_j =$ *false*. As before, when we reach a leaf ($|\mathcal{B}| = 1$), the recursion stops and we compute the total cost of the assignment to all predicates using Equation (8). Now the expected cost of a benefit $\mathcal{B}$ is defined using Equation (9) by $R(\mathcal{B}, \emptyset)$.

Example 5.2 *As in Example 4.1, the integral required to compute the value of $R(\mathcal{B}, \mathbf{t}_{l_p})$ for a particular assignment to $\mathbf{t}_{l_p}$ for a Gaussian model may require the use of numerical integration techniques. Note, however, that for value and AVERAGE queries in Gaussian models, as discussed in Example 4.1, $R_i(\mathcal{O}) = R_i(\mathbf{o}), \forall \mathbf{o}$, that is, the specific observed attribute value does not affect the reward function. Thus, the optimal observation plan is indifferent to the observed attribute value for such value or average queries. Therefore, there is no advantage to using conditional plans over the simple sequential plans discussed in Section 4 when dealing with such queries. Of course, for range queries and aggregation with selection, conditional plans can provide significant advantage, as shown in our experiments in Section 6.* $\square$

As with unconditional plans in Section 4.2, we must compute the expected improvement in confidence for our answer as we make observations. In consider a split point

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[10]. 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.

[10] 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.

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* $R_i(\mathcal{O})$:

$$R_i(\mathcal{O}) = \int p(\mathbf{o}) R_i(\mathbf{o}) d\mathbf{o}. \tag{10}$$

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

5.3 Optimizing conditional plans

In the previous sections, we defined the expected benefit $R(\mathcal{B})$ and cost $C(\mathcal{B})$ of a conditional plan $\mathcal{B}$. Of course, as in Section 4.3, different plans will lead to different benefit and cost levels. Our user will define a desired confidence level $1 - \delta$. Analogously to Equation (4) for the simpler problem of selecting observations, we would like to pick a plan $\mathcal{B}$ that meets this confidence at a minimum cost:

$$\begin{array}{ll} \text{minimize}_{\mathcal{B}} & C(\mathcal{B}), \\ \text{such that} & R(\mathcal{B}) \geq 1 - \delta. \end{array} \tag{11}$$

This general optimization problem is again NP-hard [17]. Thus, efficient and exact optimization algorithms are unlikely to exist (unless P=NP). In this paper, we describe a simple heuristic search technique for optimizing conditional plans that appears to be very effective in practice.

In Section 4.3, we presented a *subset selection* heuristic that selects a set of attributes to observe. Note that such a subset is a simple conditional plan with no branching. We thus use such subsets as the base line for our conditional planning heuristic: Each leaf l of our conditional plans is associated with a subset plan, as in Section 4.3. After observing the predicates in the path p to leaf l, our plan will continue by observing a subset plan $\mathcal{O}_l$. Estimating the benefit of these augmented trees is analogous to Equation (9), though now, when we reach a leaf, the atomic benefit $R_{\mathbf{t}_l}(\mathcal{B}, \mathbf{t}_l)$ needs to include the benefit of the subset plan $\mathcal{O}_l$, as defined in Section 4.2. Note that, when we reach l, we have already observed a set of predicates $\mathbf{t}_l$. Thus, the benefit $\mathcal{O}_l$ must be computed from the conditional distribution $P(\mathbf{o} \mid \mathbf{t}_l)$. We denote this quantity by $R(\mathcal{O}_l \mid \mathbf{t}_l)$. The cost of these augmented plans is computed in an analogous fashion, where the leaves include the cost $C(\mathcal{O}_l \mid \mathbf{t}_l)$ of the subset plan $\mathcal{O}_l$, given the observations made on the conditional plan up to leaf l.

Our heuristic conditional planning algorithm starts with a minimal conditional plan: a single node associated with the subset plan obtained from Section 4.3. The basic operation of our heuristic search is to select a leaf, and replace this leaf with a single predicate split. This operation splits the leaf into two new leaves. Each of these new leaves is then associated with a new subset plan: one leaf will have a plan for the true outcome of the predicate, and the other leaf will consider the negative outcome.

At each iteration of our heuristic, we greedily pick the leaf that offers the best expected cost-benefit for splitting. Let us first consider the expected cost of splitting a leaf l with a predicate T_j. We denote this quantity by SplitC(l, T_j). This cost is the increase from $C(\mathcal{O}_l \mid \mathbf{t}_l)$ to the new plan where the leaf l is split into two new leaves u and v:

$$\begin{aligned}\text{SplitC}(l, T_j) = {} & P(\mathbf{t}_l)C(T_j \mid \mathbf{t}_l) + \\ & P(\mathbf{t}_l)P(T_j \mid \mathbf{t}_l)C(\mathcal{O}_u \mid \mathbf{t}_l \wedge T_j) + P(\mathbf{t}_l)P(\neg T_j \mid \mathbf{t}_l)C(\mathcal{O}_v \mid \mathbf{t}_l \wedge \neg T_j) \\ & -P(\mathbf{t}_l)C(\mathcal{O}_l \mid \mathbf{t}_l),\end{aligned} \tag{12}$$

where $C(T_j \mid \mathbf{t}_l)$ is the cost of observing the attribute in T_j after observing the attributes in $\mathbf{t}_l$, as discussed in Section 5.1. Note that, in order to compute the expected cost, we must multiply benefits by the probability $P(\mathbf{t}_l)$ of reaching this leaf.

The expected benefit of splitting a leaf, SplitR(l, T_j), is computed in an analogous manner:

$$\begin{aligned}\text{SplitR}(l, T_j) = {} & P(\mathbf{t}_l)P(T_j \mid \mathbf{t}_l)R(\mathcal{O}_u \mid \mathbf{t}_l \wedge T_j) + P(\mathbf{t}_l)P(\neg T_j \mid \mathbf{t}_l)R(\mathcal{O}_v \mid \mathbf{t}_l \wedge \neg T_j) \\ & -P(\mathbf{t}_l)R(\mathcal{O}_l \mid \mathbf{t}_l).\end{aligned} \tag{13}$$

Note that if both $R(\mathcal{O}_u \mid \mathbf{t}_l \wedge T_j)$ and $R(\mathcal{O}_u \mid \mathbf{t}_l \wedge \neg T_j)$ are greater than $1 - \delta$, then the weighted average benefit:

$$P(T_j \mid \mathbf{t}_l)R(\mathcal{O}_u \mid \mathbf{t}_l \wedge T_j) + P(\neg T_j \mid \mathbf{t}_l)R(\mathcal{O}_v \mid \mathbf{t}_l \wedge \neg T_j)$$

will also be greater than $1 - \delta$, as required by the user. However, this constraint is very conservative. Suppose, for example, that $P(T_j \mid \mathbf{t}_l) = P(\neg T_j \mid \mathbf{t}_l) = 0.5$, and that $1 - \delta = 0.9$. If $R(\mathcal{O}_u \mid \mathbf{t}_l \wedge T_j) = 0.95$, we are allowed to have $R(\mathcal{O}_v \mid \mathbf{t}_l \wedge \neg T_j)$ as low as 0.85 and still guarantee our user specified bound, at a potentially lower cost. Thus, we must trade-off the distribution of benefit between leaves of our conditional plan.

The optimal strategy to achieve this trade-off is to compute the cost at the leaves for each possible benefit level, and then use a dynamic programming pass over the tree representing the plan to compute the optimal benefit levels at the leaves. Though polynomial, this algorithm requires us to compute a subset plan for each leaf and each possible benefit level. This process proved to be impractically slow in our experiments. Instead, we applied a simple heuristic: when considering splitting a leaf l into leaves u and v, we solve the subset plan problem for leaf u to the desired level of benefit $1 - \delta$, as in Section 4.3. Once this plan is computed, its expected benefit $R(\mathcal{O}_u \mid \mathbf{t}_l \wedge T_j)$ may be higher than $1 - \delta$, say, $1 - \beta$, where $\beta > \delta$. When such a situation occurs, we allow a lower target benefit $R(\mathcal{O}_v \mid \mathbf{t}_l \wedge \neg T_j)$

for leaf v. Specifically, the subset plan of leaf v must achieve at least:

$$\frac{(1-\delta) - P(T_j \mid \mathbf{t}_l)(1-\beta)}{P(\neg T_j \mid \mathbf{t}_l)}.$$

Using this heuristic, each split should give about the same expected benefit. Thus SplitR(l, T_j) should be close to zero for all leaves l, for any predicate T_j. On the other hand, splitting some leaves will reduce the expected cost more significantly than other leaves. Thus, our heuristic conditional planning algorithm can simply greedily select the leaf l that most rapidly decreases the expected cost SplitC(l, T_j).

We must also consider which predicate T_j to apply. Here, we focus on predicates of the form $T_j(X_i \geq x_i)$. Our greedy algorithm thus selects the next leaf l, and attribute X_i and value x_i that most significantly reduces SplitC(l, T_j). This method is implemented efficiently with a priority queue and caching of leaf benefit values. Furthermore by, using an indexing technique we describe in [17], we can speed-up the estimation of these values from a set of samples of the underlying probabilistic model.

6 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.

6.1 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.

1. **Garden:** 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 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 train the model using the training data.
   Observation and Communication Costs: Because of the short physical distances between these sensors, the observation costs tend to dominate the cost of any observation plan. In general it is not easy to assign precise costs to various operations done in a sensor network because of the complex operations that make up the more abstract operations. For example, in computing the communication cost, though it is possible to measure how much energy a given transmission takes, whether the radio was already on, the number of retransmissions required, and the sleep schedule that the sensor is operating under, can makes such accounting for energy non-trivial.

We simulate the fact that observation costs dominate the communication costs by setting the observation costs to be about a factor of 5 more than the communication cost. We set the observation cost to be 10mJ per observation, whereas the energy cost for a message transmission varies from a minimum of 1.25 mJ to a maximum of 6.25 mJ between two nodes in the network.

2. **Lab:** 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.

The split between training and test sets were chosen to provide training sets sufficiently large to represent the variability in the data, while leaving enough test data for a statistically-significant evaluation of the approaches.

6.2 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 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.

Average Queries: Finally, we also present results for *average* queries, that ask for the average value of a quantity over all sensors within a specified error bound ϵ with a specified confidence δ, indicating that the probability of the reported value deviating from the true average by more than ϵ is less than δ.

6.3 Comparison systems

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

TinyDB-style Querying: In this model, the query is disseminated into the sensor network using an overlay tree structure [40], 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.* [43] and the TiNA system [53].

6.4 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. These traces are used to evaluate how well BBQ does at predicting the temperature within 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. Similarly, to measure the accuracy of average queries, we compute the number of hours when BBQ provided an answer further from the true average by more than the specified error bound.

For TinyDB, all queries are answered “correctly” (as we are assuming sufficient retransmissions are done to successfully deliver messages). 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, thus, 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.

6.5 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 6 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

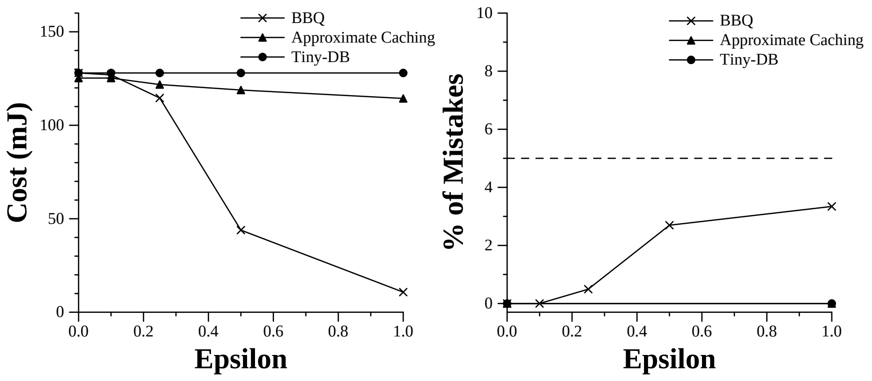


Fig. 6 Figure illustrating the relative costs of BBQ versus TinyDB and Approximate Caching on the garden data, with varying the interval width parameter, epsilon, used by BBQ and Approximate Caching, and using a confidence interval of 95% in BBQ.

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 "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 for BBQ this error is .12; in other cases the relative performance is reversed).

Figure 7 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).

6.6 Garden Dataset: Sensitivity Analysis

With this set of experiments, we examine if the effectiveness of BBQ in the above set of experiments was a result of higher observation costs in the Garden Deployment. As we discussed earlier, the observation costs in this network were about a factor of 5 more than communication costs. We artificially vary this ratio from 100 to 0, and plot the relative costs of the three techniques for 4 such ratios in Figure

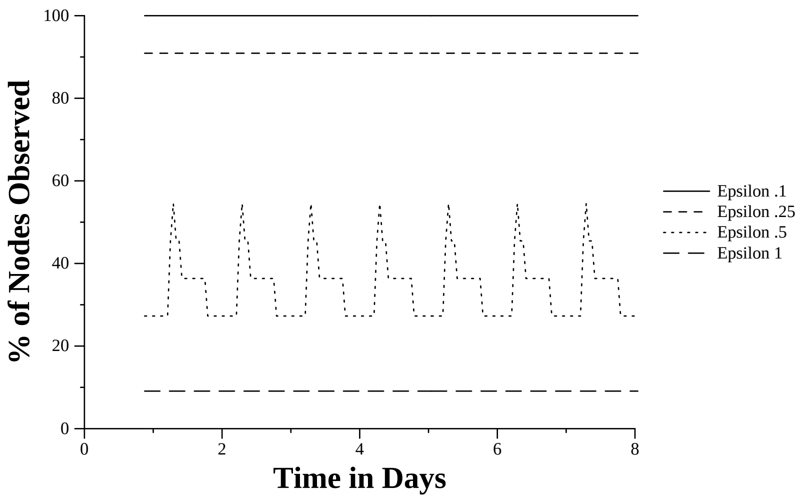


Fig. 7 Figure showing the number of sensors observed over time for varying epsilons for *value* queries with a 95% confidence on the garden data.

8. The approximate caching techniques suffer from having to make observations at every time step, and hence lower observation costs should decrease the performance benefits of BBQ over approximate caching. As we can see from Figure 8 (iv), even when the observation costs are 0, BBQ still performs significantly better than approximate caching (by a factor of almost 20 when ϵ is equal to 1).

As this sensitivity analysis demonstrates, the approximate caching technique is dominated by BBQ for all values of the sensing cost/communication cost ratios. In addition, there are significant limitations to the approximate caching approach. It does not allow us to exploit correlations between multiple sensors, and between attributes, a significant strength of our approach. Furthermore, the approximate caching technique is most appropriate for continuous queries, since nodes must be advised of every change in the query. Given these limitations of the approximate caching technique, we do not compare BBQ against approximate caching in the rest of this section.

6.7 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 9(a) and (b). Figure 9(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 9(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

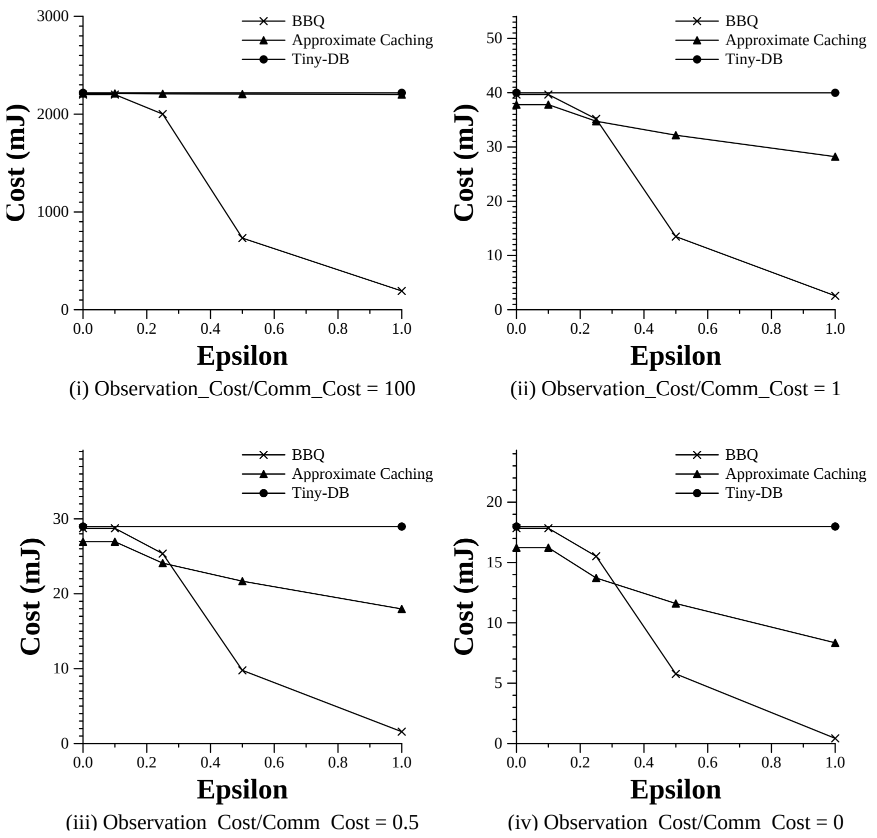


Fig. 8 Comparing the effectiveness of the algorithms for varying ratios of communication and observation costs. As we can see, over a wide range of ratios, the performance of BBQ is significantly better than either of the other two techniques. Note that the *y*-axes are plotted on different scales across the graphs.

bounds by a small amount, as variances in our training data are somewhat different than variances in our test data.

6.8 Garden Dataset: Comparing Observation Planning Algorithms

Figure 10 shows the experimental results comparing the greedy and the optimal observation planning algorithms. We plot this for value of epsilon equal to 0.5. We also show the results for a naive randomized algorithm that begins by (1) randomly generating 5 observation plans that observe one attribute each, (2) checking if any of them satisfies the confidence bound, and choosing the least cost plan among those that do in that case, and (3) repeating Step 1 with observation plans that observe *one* more attribute otherwise.

As we can see, the performance of the greedy algorithm is very close to optimal in most cases, whereas Random-5 performs somewhat worse than either of the

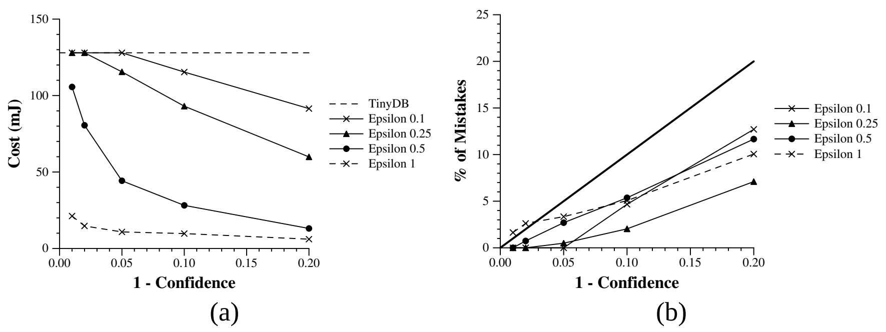


Fig. 9 Energy per query (a) and percentage of errors (b) versus confidence interval size and epsilon, for *value* queries on the garden data. The 45° line in (b) represents the target number of mistakes for the given confidence level, points below this line have made less mistakes than the confidence bounds allow them to make.

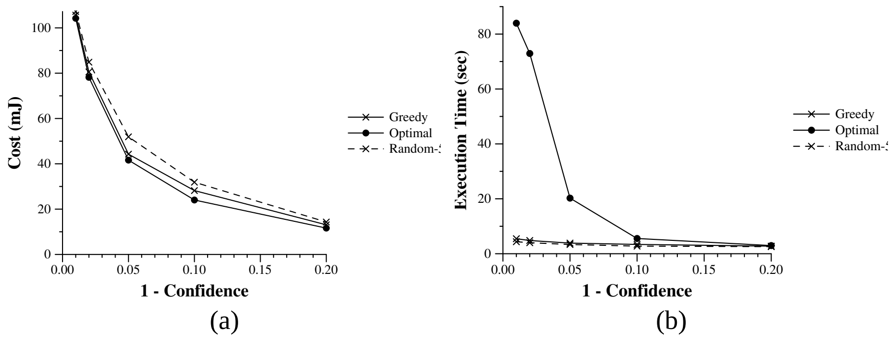


Fig. 10 (a) Energy per query, and (b) observation planning times for various planning algorithms versus the desired confidence on the garden data. We use a *value* query with an epsilon of 0.5.

two algorithms. Comparing the execution times of these three algorithms, we can see that the execution time of the optimal algorithm is significantly higher than either of the other two algorithms, in some cases, by orders of magnitude. This experiment demonstrates that the effectiveness and viability of the greedy planning algorithm.

6.9 Garden Dataset: Comparing Dynamic vs Static

Next, we evaluate the effectiveness of dynamic models that incorporate observations made in the past to perform prediction. In Figure 11, we plot the ratio of execution costs of BBQ using static and dynamic models versus confidence for various values of epsilon. We show plots for both *value* and *average* queries. As we can see, the benefits of using dynamic models can be very significant over time, especially for higher values of epsilon. For smaller values of epsilon, both algorithms tend to make a lot of observations, thus decreasing the effectiveness of using dynamic models.

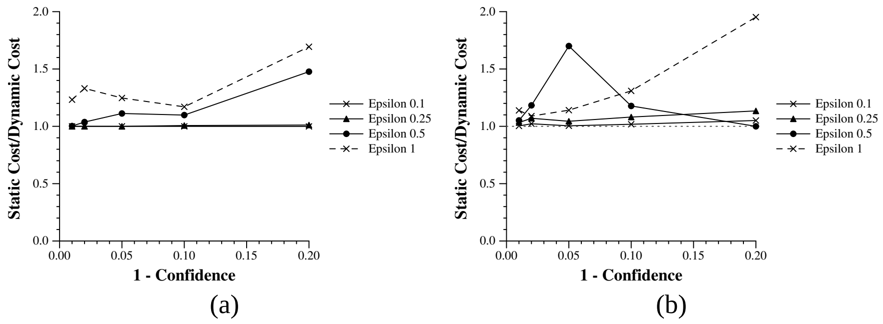


Fig. 11 Ratio of costs of dynamic vs static for *value* and *average* queries on the garden data.

6.10 Garden Dataset: Average Queries

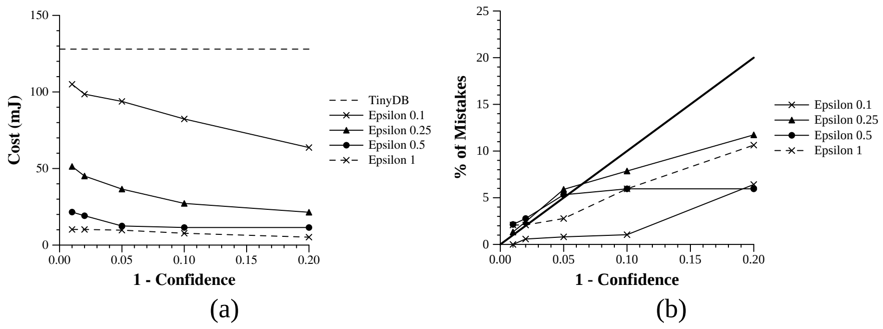


Fig. 12 Energy per query (a) and percentage of errors (b) versus confidence interval size and epsilon, for *average* queries on the garden data.

Figure 12 shows the results of running BBQ for *average* queries for varying values of epsilon and confidences. As we can see, in most cases, we easily achieve our required error bounds, and the number of observations required to do so is typically very small.

6.11 Garden Dataset: Range queries

We ran a number of experiments with range queries (also over the *garden* data set). Figure 13 summarizes the average number of observations required for a 95% confidence with three different range queries (temperature in [16,17], temperature in [19,20], and temperature in [21,22]). Note that the ϵ parameter does not play a role in range queries, as discussed in Section 3.1. 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

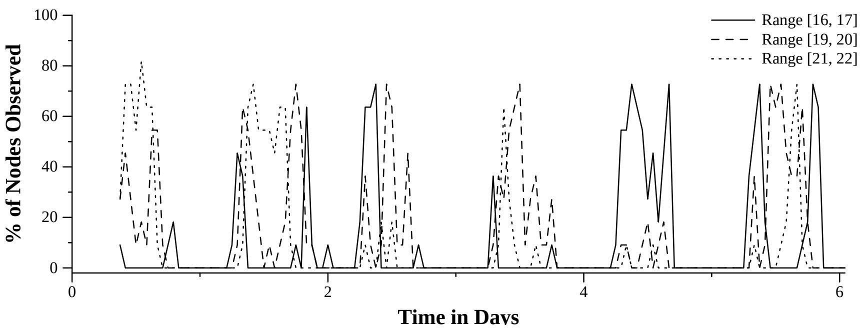


Fig. 13 Graph showing BBQ's performance on three different range queries, for the garden data set with confidence set to 95%.

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 19-20, but never for other ranges (because it can be sure the temperature is above 21 degrees, but not 22!)

6.12 Lab *dataset*

We also ran similar experiments on the *lab* dataset. We report one set of experiments for this dataset as the results are similar to those found in the Garden data.

Figure 14(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 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 14(b) shows that BBQ was able to achieve the specified confidence bounds in almost all the cases.

Figure 15 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 among 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. The traversal observes a few nodes at the edges of the lab because there is more variance near the windows.

6.13 Conditional Planning Experiments

In this section, we present experimental results demonstrating the effectiveness of conditional planning to further reduce the number of observations made during query evaluation. Recall that for the Gaussian probability distributions, conditional planning can only make a difference for *range* queries, and not *value* or *average* queries (Section 5.2). Hence we will present results for two different range queries. The query processing techniques compared are:

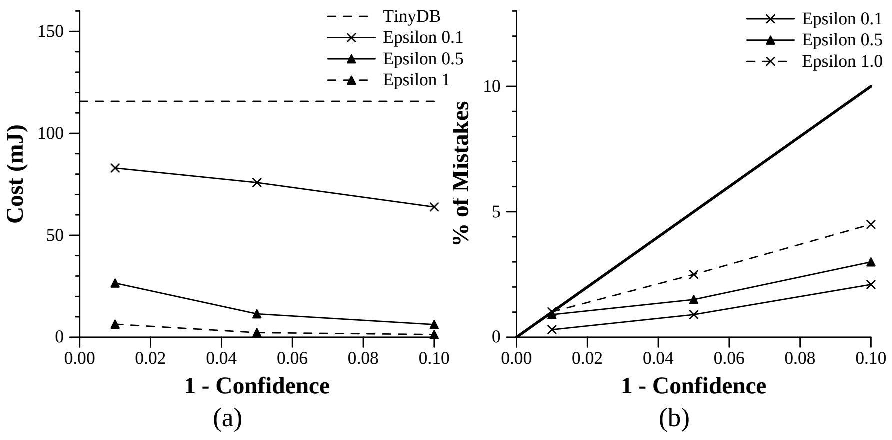


Fig. 14 Energy per query (a) and percentage of errors (b) versus confidence interval size and epsilon for the Lab Data.

- **TinyDB:** As above this denotes the cost of collecting all the data from the sensor networks using the minimal routing tree over the network. Since all data is collected at all times, this technique always gives correct answers as long as there is no missing data.
- **BBQ-Static:** Running BBQ with *dynamic* models turned off.
- **BBQ-Static-Cond-n:** Running BBQ with *dynamic* models turned off, but using conditional plans with at most n branches.

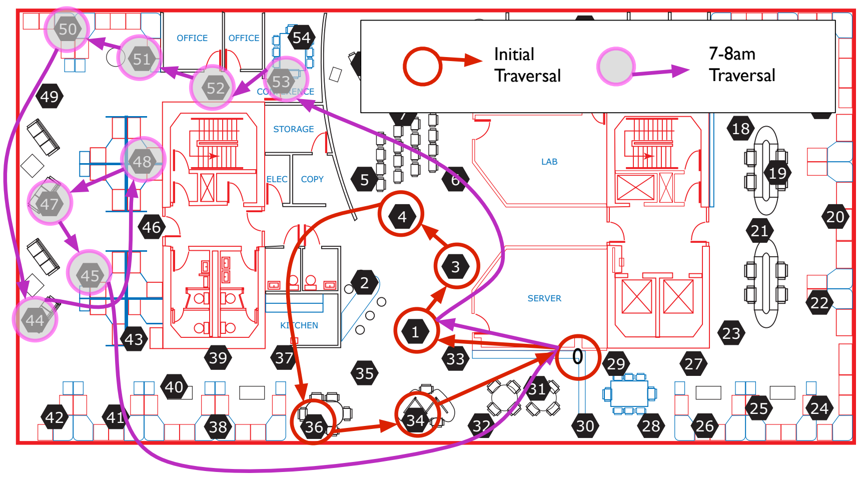
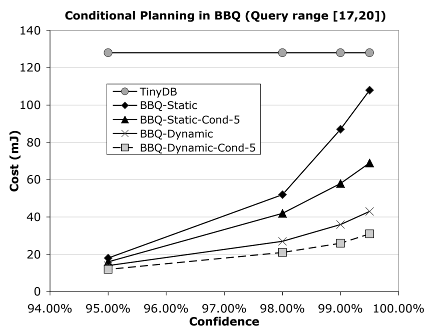
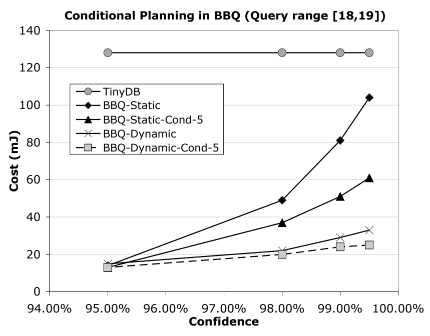


Fig. 15 Two traversals of the lab network, for a value-query over all 54 sensors with $\epsilon = .1$ and $\delta = .99$. An observation is made at every circled node. These paths correspond to times when the model's variance is high, *e.g.*, when the system is first started, and at around 8am when the sun begins to heat the lab, so many observations are needed. For other hours, very few observations are needed.



(a)



(b)

Fig. 16 Conditional planning experiments on the garden data: We plot the energy costs of various query evaluation techniques for two range queries, $\langle 17, 20 \rangle$, and $\langle 18, 19 \rangle$.

– **BBQ-Dynamic.** BBQ with *dynamic* models turned on.
– **BBQ-Dynamic-Cond-n.** BBQ with *dynamic* models, and conditional planning with at most n branches turned on.

Figure 16 shows the results of our experiments comparing these five techniques for two range queries, $[17, 20]$, and $[18, 19]$. As we can see, using conditional plans results in significant improvements for both BBQ-Static and BBQ. Overall, the performance benefit of conditional planning in the dynamic case is about 35% at high confidence levels (99.5% and 99%), and about 15% at low confidence levels (where few observations are made by any of the BBQ approaches.) In the static case, the performance benefit of conditional planning is about 90% at high confidence levels, and still only 15% at low confidence levels. The number of mistakes made by the algorithms is not plotted as it does not change appreciably as the number of branches is increased, and is typically within the required bounds.

The effectiveness of conditional planning with dynamic models is slightly less than the improvement gained with static models. The reason behind this is the strong temporal correlations present in our data. Recall that conditional planning can result in a smaller number of observations made at any time step compared to when conditional plans are not used. But when using dynamic models, the extra observations made by vanilla BBQ reduce the number of observations required in the subsequent steps. In essence with dynamic models, no observations are fully useless. In spite of that we see at least a 40% reduction in many of the cases.

6.13.1 Example Plan Figure 17 shows an example of a conditional plan built over the garden data set. We limited the number of conditioning predicates to 4. The nodes were distributed as shown in the small tree diagram on the left. In this deployment, nodes towards the top of the tree (7, 8, 9, and 10) tend to experience more extreme temperatures – hotter on a sunny day and colder on a cold night, as they are more exposed to the elements. Similarly, the nodes at the bottom of the tree are subject to the least extreme conditions. Generally, the variation in temperatures

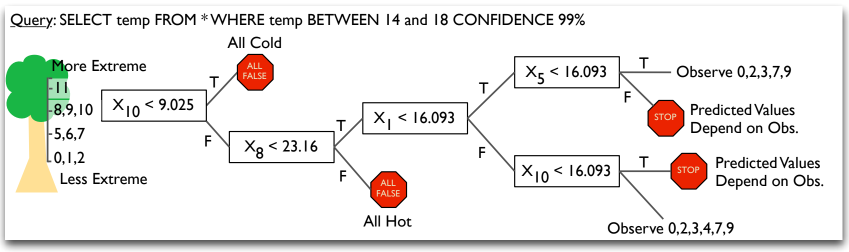


Fig. 17 Example of a conditional plan generated by BBQ on the garden data.

between the top and bottom of the tree is less than a few degrees, though on hot days it can be more than ten degrees. Variance tends to be lower when it is cool, and higher when it is hot. The query was a range query over temperature of all attributes, looking for nodes with temperatures in the range [14,18]. In the data set, temperatures varied from about 7^{o}C to about 30^{o}C.

The first few observations of nodes 10 and 8 are to determine whether all of the nodes are definitely below or above the query range respectively. In these cases, query evaluation can be stopped early. Then, the system splits on the value of node 1 decide whether to observe node 10 or node 5 next. If node 1 is reading less than the mid-point of the query range (16^o), the plan observes the value of node 5. If node 5 is also cool, the system has to observe many other sensors, indicating that at times when a few sensors are cool but not cold, it is approximately equally likely for sensors to be warmer than 14^o as it is for them to be cooler than 14^o. When node 5 is relatively warm, nodes 0 and 2 read about the same as node 1 (whose value has already been observed), while nodes 4 and higher read the same or warmer than 5. Thus, no additional observations are needed to answer the query with high confidence.

In the case where the reading of node 1 is greater than the midpoint, the system chooses to branch on the value of node 10 again (at no additional observation cost.) If node 10 is reading less than the midpoint, then, since it is not cold (based on the first observation of 10), the temperature is probably relatively constant throughout the tree, due to the fact that the variance of temperatures is less when it is cool outside. Thus, since node 1 satisfies the predicate, the probability that all sensors satisfy the predicate is high. If sensor 10 is hotter than the midpoint, however, additional observations are needed, since the temperature varies substantially during times when sensor 10 is warm.

7 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.

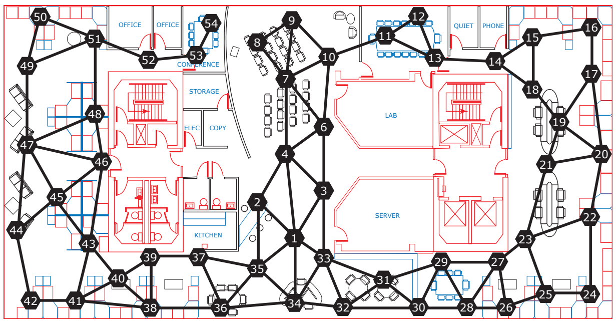


Fig. 18 Structure of graphical model learned for the temperature variables for a deployment in the Intel Berkeley Lab [45].

7.1 Representations for probability distributions

Probabilistic models are the centerpiece of our approach. Choosing the appropriate representation for a model is very important: the model must allow us to represent complex correlations compactly, to learn the parameters effectively, and to answer queries efficiently. The basic Gaussian models used in BBQ are efficient and compact, but many real-world problems may not satisfy the Gaussian assumption. We are investigating the use of *probabilistic graphical models* as a generalization to the basic Gaussian models that allows us to tackle complex correlation structures efficiently [46, 13].

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 18, where the attributes are the temperatures in various locations in the Intel Berkeley Lab. The graphical model in Figure 18 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 [13] is the key to efficient representation and probabilistic querying. In discrete settings, for example, a naive representation of $p(X_1, X_2, \ldots, X_n)$ 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 [27]. 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.

Additionally, graphical models allow us to efficiently address hidden variables, both in terms of answering queries and of learning about hidden variables [19].

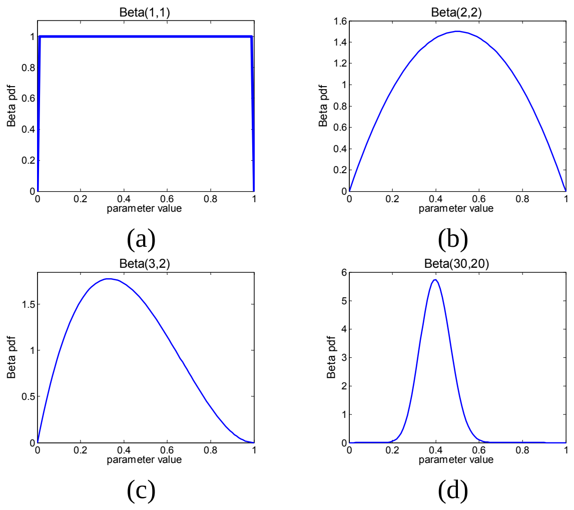


Fig. 19 Bayesian approach for learning the parameter of a coin: (a) prior distribution, Beta(1,1); posterior distributions over the coin parameter after observing: (b) 1 head, 1 tail; (b) 2 heads, 1 tail; (b) 29 heads, 19 tails.

In the example in Figure 18, each node could be associated with a faulty sensor hidden variable [36]. 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 [9, 15], and to answer causal queries [47].

7.2 Integrating learning and querying

The BBQ approach described above is 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 [7].

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 $p(\Theta \mid \mathbf{x})$:

$$p(\Theta \mid \mathbf{x}) \propto p(\mathbf{x} \mid \Theta) p(\Theta). \qquad (14)$$

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 (14); 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 19 illustrates the process of Bayesian learning for our coin example: We start with the Beta(1,1) in Figure 19(a); here, the distribution over possible coin parameters is almost uniform. Figures 19(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 query 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.

7.3 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. We saw in this paper how *conditional plans* can be used to exploit the correlations present in the data during execution of the query plan.

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 now than start a new traversal in the next time step.

The optimal solution to such long-term planning problems can be formulated as a *Markov decision process* (MDP) [6, 50]. 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 [8, 26]. Such approaches could be extended to address the long-term planning problem that arises in our setting.

7.4 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 [38, 28]. 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 [25, 45]. 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 [45]; or each node obtains a functional representation (*e.g.*, a curve fit) of the sensor (*e.g.*, temperature) field [25]. 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.

7.5 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 be used to help detect outlier readings and reduce communication during times of normal operation, as with the fault detection case.

Though the approach we have described above does not handle outliers, it is important to emphasize that outlier readings are not the same as low-probability events that are properly modeled by a probability distribution. For example, in a Redwood forest, the temperature during most days is relatively cool (about 18 °C). However, there are some days during the Summer when the temperature can rise to 30 °C or higher. On such days, BBQ will observe the temperature of a few sensors and properly predict that many of the sensors are much warmer than usual. BBQ would not, however, correctly predict the temperature of an “outlier” sensor that had fallen off the tree and landed on the cold ground on the forest floor.

7.6 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 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.

8 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 [23, 22, 3] 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. [16, 21] 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.* [35] and Nath *et al.* [42] have shown that sketch based approximation techniques can be applied in sensor networks.

Work on approximate caching by Olston *et al.*, is also related [44, 43], 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. Furthermore, this caching approach does not exploit correlation between sensors, requiring every node that is outside the desired bound to report its value. Our approach, conversely, uses the probabilistic model to query only a subset of the sensors. The caching approach does, however, enable queries that detect outliers, something BBQ currently cannot do. The TiNA system from Beaver *et al.* [53]

shows how to apply a technique similar to Olston's approximate caching in the context of sensor networks.

There has been some recent work on approximate, probabilistic querying in sensor networks and moving object databases [10]. 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 a confidence bound 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.* [11] 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) [39] – 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 [31] and other aspects of the CONTROL project [29]. 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.

Our idea of conditional plans is quite similar to *parametric query optimization* [34, 24, 12, 20], where part of the query optimization process is postponed until the runtime. Typically, these techniques choose a *set* of query plans at query optimization, and identify a set of conditions that are used to select one of those plans at runtime. This earlier work differed substantially from ours in two essential ways: First, in these traditional approaches, the plan chosen at the runtime is used for executing the query over the entire dataset; thus, even if correlations were taken into account by these approaches, per-tuple variations, which we have seen to be prevalent and widely exploitable, could not be accounted for. Secondly, these approaches did not exploit data correlations while generating the plans.

Adaptive query processing techniques [30] attempt to reoptimize query execution plans during query execution itself. We believe that the idea of conditional plans that we propose is both orthogonal and complementary to adaptive query processing. If sufficiently accurate information about the data is available (as we assume in this work), then conditional plans can reap many of the benefits of adaptive query processing techniques *a priori* (by choosing different query plans for different parts of data). However, in many cases, such information may not be available, and adaptive techniques must be used. Babu *et al.* [4] address the problem of adaptively ordering *pipelined filters* (selection predicates) that may have correlations. Their focus is on finding good *sequential* plans (that may change with time), and they do not consider conditional plans. Shivakumar *et al.*, [54] propose using low-cost predicates to avoid evaluating expensive predicates. Their approach, however, constructs a sequential plan, not a conditional one.

The probabilistic querying techniques described here are built on standard results in machine learning and statistics (*e.g.*, [51, 41, 13]). The optimization problem we address is a generalization of the *value of information* problem [51]. This article, however, proposes and evaluates the first general architecture that combines model-based approximate query answering with optimizing the data gathered in a sensornet.

9 Conclusions

In this article, 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. Our approach includes a notion of *conditional plans* that introduce conditioning predicates to determine the best order in which sensor attributes should be observed. In a sensor network, this provides a number of advantages, including predicting the value of missing sensors and reducing the number of expensive sensor readings and radio transmissions that the network must perform. Beyond the encouraging, substantial 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. S. Acharya, P. Gibbons, V. Poosala, and S. Ramaswamy. Join synopses for approximate query answering, 1999.
4. S. Babu, R. Motwani, K. Munagala, I. Nishizawa, and J. Widom. Adaptive ordering of pipelined stream filters. In *SIGMOD*, 2004.
5. R. Bahar, E. Frohm, C. Gaona, G. Hachtel, E. Macii, A. Pardo, and F. Somenzi. Algebraic decision diagrams and their applications. In *IEEE International Conference on Computer-Aided Design*, pages 188–191, 1993.
6. R. E. Bellman. *Dynamic Programming*. Princeton, 1957.
7. J. Bernardo and A. Smith. *Bayesian Theory*. Wiley, 1994.
8. C. Boutilier, R. Dearden, and M. Goldszmidt. Exploiting structure in policy construction. In *Proc. IJCAI*, pages 1104–1111, 1995.

9. X. Boyen and D. Koller. Tractable inference for complex stochastic processes. In *Proc. UAI*, 1998.
10. R. Cheng, D. V. Kalashnikov, and S. Prabhakar. Evaluating probabilistic queries over imprecise data. In *SIGMOD*, 2003.
11. 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.
12. R. Cole and G. Graefe. Optimization of dynamic query evaluation plans. In *SIGMOD*, 1994.
13. R. Cowell, P. Dawid, S. Lauritzen, and D. Spiegelhalter. *Probabilistic Networks and Expert Systems*. Spinger, New York, 1999.
14. Crossbow, Inc. Wireless sensor networks. `http://www.xbow.com/Products/Wireless_Sensor_Networks.htm`.
15. T. Dean and K. Kanazawa. A model for reasoning about persistence and causation. *Computational Intelligence*, 5(3):142–150, 1989.
16. A. Deshpande, M. Garofalakis, and R. Rastogi. Independence is Good: Dependency-Based Histogram Synopses for High-Dimensional Data. In *SIGMOD*, May 2001.
17. A. Deshpande, C. Guestrin, S. Madden, and W. Hong. Exploiting correlated attributes in acquisitional query processing. In *ICDE*, 2005.
18. A. Desphande, C. Guestrin, S. Madden, J. M. Hellerstein, and W. Hong. Model-driven data acquisition in sensor networks. In *VLDB*, 2004.
19. 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.
20. S. Ganguly. Design and analysis of parametric query optimization algorithms. In *VLDB'98, Proceedings of 24rd International Conference on Very Large Data Bases*, 1998.
21. L. Getoor, B. Taskar, and D. Koller. Selectivity estimation using probabilistic models. In *SIGMOD*, May 2001.
22. P. B. Gibbons. Distinct sampling for highly-accurate answers to distinct values queries and event reports. In *Proc. of VLDB*, Sept 2001.
23. P. B. Gibbons and Y. Matias. New sampling-based summary statistics for improving approximate query answers. In *SIGMOD*, 1998.
24. G. Graefe and K. Ward. Dynamic query evaluation plans. In *SIGMOD*, 1989.
25. 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.
26. 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.
27. D. Heckerman. A tutorial on learning with bayesian networks, 1995.
28. 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.
29. 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.
30. 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.

31. J. M. Hellerstein, P. J. Haas, and H. Wang. Online aggregation. In *SIGMOD*, pages 171–182, Tucson, AZ, May 1997.
32. C. Intanagonwiwat, R. Govindan, and D. Estrin. Directed diffusion: A scalable and robust communication paradigm for sensor networks. In *MobiCOM*, Boston, MA, August 2000.
33. Intersema. Ms5534a barometer module. Technical report, October 2002. `http://www.intersema.com/pro/module/file/da5534.pdf`.
34. Y. E. Ioannidis, R. T. Ng, K. Shim, and T. K. Sellis. Parametric query optimization. In *18th International Conference on Very Large Data Bases*, 1992.
35. G. Kollios, J. Considine, F. Li, and J. Byers. Approximate aggregation techniques for sensor databases. In *ICDE*, 2004.
36. 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.
37. S. Lin and B. Kernighan. An effective heuristic algorithm for the tsp. *Operations Research*, 21:498–516, 1971.
38. S. Madden. The design and evaluation of a query processing architecture for sensor networks. Master's thesis, UC Berkeley, 2003.
39. 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.
40. S. Madden, W. Hong, J. M. Hellerstein, and M. Franklin. TinyDB web page. http://telegraph.cs.berkeley.edu/tinydb.
41. T. Mitchell. *Machine Learning*. McGraw Hill, 1997.
42. S. Nath, P. Gibbons, S. Seshan, and Z. Anderson. Synopsis diffusion for robust aggregation in sensor networks. In *Proceedings of SenSys*, 2004.
43. C. Olston and J.Widom. Best effort cache sychronization with source cooperation. *SIGMOD*, 2002.
44. C. Olston, B. T. Loo, and J. Widom. Adaptive precision setting for cached approximate values. In *ACM SIGMOD*, May 2001.
45. 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*.
46. J. Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann Publishers, 1988.
47. J. Pearl. *Causality : Models, Reasoning, and Inference*. Cambridge University Press, 2000.
48. J. Polastre. Design and implementation ofwireless sensor networks for habitat monitoring. Master's thesis, UC Berkeley, 2003.
49. G. Pottie and W. Kaiser. Wireless integrated network sensors. *Communications of the ACM*, 43(5):51 – 58, May 2000.
50. M. L. Puterman. *Markov decision processes: Discrete stochastic dynamic programming*. Wiley, 1994.
51. S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 1994.
52. Sensirion. Sht11/15 relative humidity sensor. Technical report, June 2002. `http://www.sensirion.com/en/pdf/Datasheet_SHT1x_SHT7x_0206.pdf`.
53. A. Sharaf, J. Beaver, A. Labrinidis, and K. Chrysanthis. Balancing energy efficiency and quality of aggregate data in sensor networks. *The VLDB Journal*, 13(4):384–403, 2004.
54. N. Shivakumar, H. Garcia-Molina, and C. Chekuri. Filtering with approximate predicates. In *VLDB*, 1998.

55. TAOS, Inc. Tsl2550 ambient light sensor. Technical report, September 2002. `http://www.taosinc.com/pdf/tsl2550-E39.pdf`.
56. Y. Yao and J. Gehrke. Query processing in sensor networks. In *Conference on Innovative Data Systems Research (CIDR)*, 2003.