Distributed Regression: an Efficient Framework for Modeling Sensor Network Data

Carlos Guestrin[†] Peter Bodik[‡] Romain Thibaux[‡] Mark Paskin[‡] Samuel Madden[†]
[†]Intel Research - Berkeley Lab
[‡]Computer Science Division, University of California, Berkeley

ABSTRACT

We present *distributed regression,* an efficient and general framework for *in-network* modeling of sensor data. In this framework, the nodes of the sensor network collaborate to optimally fit a global function to each of their local measurements. The algorithm is based upon *kernel linear regression*, where the model takes the form of a weighted sum of local basis functions; this provides an expressive yet tractable class of models for sensor network data. Rather than transmitting data to one another or outside the network, nodes communicate constraints on the model parameters, drastically reducing the communication required. After the algorithm is run, each node can answer queries for its local region, or the nodes can efficiently transmit the parameters of the model to a user outside the network. We present an evaluation of the algorithm based upon data from a 48-node sensor network deployment at the Intel Research - Berkeley Lab, demonstrating that our distributed algorithm converges to the optimal solution at a fast rate and is very robust to packet losses.

Categories and Subject Descriptors

G.3 [**Mathematics of Computing**]: Probability and Statistics—*Correlation and regression analysis, Multivariate statistics*; C.1.4 [**Processor Architectures**]: Parallel architectures—*Wireless sensor networks, Distributed architectures, Mobile processors*; I.2.6 [**Artificial Intelligence**]: Learning—*Machine learning, Parameter learning*; I.2.11 [**Artificial Intelligence**]: Distributed Artificial Intelligence—*Coherence and coordination*; C.2.4 [**Computer-Communication Networks**]: Distributed systems—*Distributed applications*

General Terms

Algorithms

Keywords

regression, wireless sensor networks, machine learning, distributed algorithms

IPSN'04, April 26–27, 2004, Berkeley, California, USA.

1. INTRODUCTION

Networks of small, low-power devices that can sense, actuate, and communicate information about their environment are proving a useful tool for many tasks in science and industry. In recent years, developments in hardware and low-level software have led to viable, multi-hundred node networks being deployed for remote monitoring of environmental and climatological data (Mic03; MPS+02).

Such monitoring systems are typically used in one of two modes of operation: either the data from the sensors is extracted from the network and analyzed off-line (MPS+02); or the information obtained from the sensors is aggregated using simple local operations that compute, for example, averages, maxima, or histograms (Mad03; IEGH02). The reduction of communication through aggregation is attractive since extraction of complete data sets can be very expensive, requiring large amounts of communication that drains the limited energy of these devices.[1] Unfortunately, although aggregation conserves energy, it can lose much of the original structure in the data, providing only coarse statistics that smooth over interesting local variations.

In this paper, we propose a method for extracting much more complete information about the shape and structure of sensor data than most aggregation schemes provide while still using much less communication than methods that retrieve every reading from every sensor. To do this, we leverage the fact that measurements from multiple sensors in the same space are often highly correlated. Thus, the effective dimensionality of the data may be significantly lower than the total number of sensor measurements. A few previous published sensor network querying algorithms have exploited this type of structure, but these methods only address very specific tasks, such as computing contour levels of sensor values (NM03).

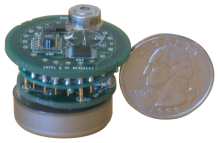

Figure 1: Sensor node used in deployment.

We present a general framework for answering complex queries in a sensor network. This framework can accurately represent the structure of the original data while significantly decreasing the communication requirements, by projecting the readings into a lower dimensionality representation. Specifically, we use *linear regression*, where the data is approximated by a weighted linear combination of basis functions, to perform this projection. We model local correlation in the data using *kernel linear regression*, where the support of a kernel function determines the set of measurements that are used to estimate each basis function coeffi-

[1]Communication is widely viewed as the dominating power cost in many sensor networks applications (PK00; HSW+00); (Mad03) claims that, for some data collection tasks, 94% of energy in motes is spent on some aspect of communication.

cient. The coefficients of these basis functions and locations of kernels then provide a compact, structured model of the measurements.

We describe a simple and efficient distributed message passing algorithm for performing regression within a sensor network. Our algorithm maintains the resulting representation in a distributed fashion, and is able to extract this representation from the network at a minimal communication cost. Our approach overlays a junction tree—a data structure commonly used for probabilistic inference in graphical models (CDLS99)—on top of the routing tree often used for data collection in sensor nets. By passing a pair of messages between adjacent nodes in this tree, we optimize the basis function coefficients to minimize the reconstruction error.

The primary contributions of this work are as follows:

- we describe kernel-based regression and argue that it is well suited to predicting the behavior of the spatio-temporally correlated data common in sensor nets;
- we show that a solution to the kernel regression problem, based upon Gaussian elimination, can be formulated as a communication-efficient distributed message passing algorithm that can be practically implemented in a sensor network;
- we show that tree-based communication topologies common in sensor networks are capable of effectively supporting this message passing algorithm; and,
- we present an evaluation of our approach using an implementation in TinyOS on the TOSSIM simulator (LLW+03), demonstrating the compactness and accuracy of the models generated by our distributed regression algorithm, and the fast convergence rate, and high robustness to packet losses.

2. REGRESSION IN SENSOR NETS

A sensor network is composed by a set of n nodes $(N_1, \ldots, N_n)$ that communicate over a wireless network. Figure 5 illustrates a simple example of a sensor network that we deployed in the Intel Research Lab in Berkeley, using the sensor nodes, or *motes*, shown in Figure 1. This deployment, based on TinyDB (Mad03), periodically collects a set of environmental attributes, such as light, temperature, and humidity, as well as network connectivity and routing topology.

2.1 Regression models

Each node in the network generates large amounts of data; for example, in the deployment used to generate data for this paper, each sensor produces a reading every 30 seconds, or 120 readings an hour. This sample rate directly affects the expected lifetime of the network: at this rate, we anticipate a network lifetime of about one month. Switching to a slower rate, where each sensor transmits, for example, every 10 minutes, would increase the life to more than 18 months, but, in general, lowering the sampling rate may cause us to miss high frequency events. Thus, approaches that reduce communication, while still retaining the structure in the data, will substantially extend network lifetime.

An alternative to extracting all of the measurements is to build a model of this data in the network and transmit only the model coefficients. For example, instead of extracting the humidity measurement from node N_i every 30 seconds, we can fit a degree-three polynomial to the last 100 measurements: $f(t) = w_0 + w_1 t + w_2 t^2 + w_3 t^3$. Then we only need to extract 4 parameters from the network: w_1, w_2, w_3 and w_4. More generally, given a set of basis functions (*e.g.*, 1, t, t^2, and t^3), we would like to continuously fit their parameters and thereby reduce the dimensionality of the data.

This modeling process can be achieved using *regression*. In regression, a sensor measures a function $f(t)$ at some time t (*e.g.*, humidity). Over time, it collects a set of data points $(f(t_1), \ldots, f(t_m))$. We are given a set of basis functions $H = (h_1, \ldots, h_k)$, and we would like to fit these basis functions to the measurements, that is, to find basis function coefficients $\mathbf{w} = (w_1, \ldots, w_k)^\intercal$ such that:

$$\widehat{f}(t) = \sum_i w_i h_i(t) \approx f(t).$$

By choosing the number of coefficients to be far smaller than the number of measurements ($k \ll m$), the coefficient vector $\mathbf{w}$ becomes a compressed representation of the measurements.

To formalize this notion of approximation, we must define an error metric. We focus our presentation on *root mean squared error* (RMS). That is, we would like to pick the parameters $\mathbf{w}^*$ that minimize the RMS:

$$\begin{aligned} \mathbf{w}^* &= \arg\min_{\mathbf{w}} \sqrt{\frac{1}{m}\sum_{j=1}^{m}\left(f(t_j) - \widehat{f}(t_j)\right)^2}, \\ &= \arg\min_{\mathbf{w}} \sqrt{\frac{1}{m}\sum_{j=1}^{m}\left(f(t_j) - \sum_i w_i h_i(t_j)\right)^2}. \quad (1) \end{aligned}$$

We can perform such optimization using *linear regression* (GV89). To present the algorithm, we need to define a basis matrix $\mathbf{H}$ with one column for each basis function and one row for each measurement. Similarly, we define a measurement vector $\mathbf{f}$ with one row for each measurement:

$$\mathbf{H} = \begin{pmatrix} h_1(t_1) & h_2(t_1) & \cdots & h_k(t_1) \\ h_1(t_2) & h_2(t_2) & \cdots & h_k(t_2) \\ \vdots & \vdots & & \vdots \\ h_1(t_m) & h_i(t_m) & \cdots & h_k(t_m) \end{pmatrix}, \quad \mathbf{f} = \begin{pmatrix} f(t_1) \\ f(t_2) \\ \vdots \\ f(t_m) \end{pmatrix}.$$

Note that $\mathbf{H}$ is a $m \times k$ matrix, and $\mathbf{f}$ is a $m \times 1$ vector. using this notation, we can restate our optimization problem (1) in matrix notation as: $\mathbf{w}^* = \arg\min_{\mathbf{w}} \|\mathbf{H}\mathbf{w} - \mathbf{f}\|$. Setting the gradient of this quadratic objective to zero gives the optimal coefficients:

$$\mathbf{w}^* = \left(\mathbf{H}^\intercal \mathbf{H}\right)^{-1} \mathbf{H}^\intercal \mathbf{f}. \quad (2)$$

It is instructive to understand each term in the right hand side of this equation:

$$\mathbf{A} = \mathbf{H}^\intercal\mathbf{H} = \begin{pmatrix} \langle h_1 \bullet h_1\rangle & \langle h_1 \bullet h_2\rangle & \cdots & \langle h_1 \bullet h_k\rangle \\ \langle h_2 \bullet h_1\rangle & \langle h_2 \bullet h_2\rangle & \cdots & \langle h_2 \bullet h_k\rangle \\ \vdots & \vdots & & \vdots \\ \langle h_k \bullet h_1\rangle & \langle h_k \bullet h_2\rangle & \cdots & \langle h_k \bullet h_k\rangle \end{pmatrix},$$

$$\mathbf{b} = \mathbf{H}^\intercal\mathbf{f} = \begin{pmatrix} \langle h_1 \bullet f\rangle \\ \langle h_2 \bullet f\rangle \\ \vdots \\ \langle h_k \bullet f\rangle \end{pmatrix},$$

where we denote dot products by $\langle f \bullet g\rangle = \sum_{i=1}^{m} f(t_i)g(t_i)$. The first term $\mathbf{A}$ is the *dot-product matrix*, where each element denotes the dot product between two basis functions. The second term $\mathbf{b}$ is the *projected measurement vector*, where each element is simply the projection of the measurement vector into the space of a particular basis function. Thus, given the measurement vector and the basis functions, we can compute the optimal regression weights with

simple matrix operations. Typically, one computes the dot-product matrix $\mathbf{A}$ and the projected measurement vector $\mathbf{b}$, and obtains the optimal weights by solving the linear system $\mathbf{Aw} = \mathbf{b}$, using, for example, Gaussian elimination (GV89).

In a sequential process, such as the measurement of temperature over time, we may want to fit a regression model using a *sliding window*. That is, we fit the coefficients of our basis functions with respect to the measurements performed in the last T minutes. Suppose that we have computed the matrix $\mathbf{A}$ and the vector $\mathbf{b}$ for measurements at times $t_1, \ldots, t_{m-1}$, and we observe a new measurement at time t_m. If we define

$$\mathbf{A}(t_m) = \begin{pmatrix} h_1(t_m)h_1(t_m) & \cdots & h_1(t_m)h_k(t_m) \\ \vdots & \vdots & \vdots \\ h_k(t_m)h_1(t_m) & \cdots & h_k(t_m)h_k(t_m) \end{pmatrix}, \text{ and}$$

$$\mathbf{b}(t_m) = \begin{pmatrix} h_1(t_m)f(t_m) \\ \vdots \\ h_k(t_m)f(t_m) \end{pmatrix},$$

we can incrementally update $\mathbf{A}$ and $\mathbf{b}$ simply by:

$$\mathbf{A} \leftarrow \mathbf{A} + \mathbf{A}(t_m) \qquad \mathbf{b} \leftarrow \mathbf{b} + \mathbf{b}(t_m) \tag{3}$$

Similarly, if measurement t_1 falls outside our time window, we can remove the influence of $f(t_1)$ by:

$$\mathbf{A} \leftarrow \mathbf{A} - \mathbf{A}(t_1) \qquad \mathbf{b} \leftarrow \mathbf{b} - \mathbf{b}(t_1) \tag{4}$$

Thus, we can incrementally update $\mathbf{A}$ and $\mathbf{b}$ as we receive new measurements. The basis function coefficients can be computed at any time by solving the linear system $\mathbf{Aw} = \mathbf{b}$.

2.2 Modeling spatial data

In addition to significant amount of redundancy in readings from a sensor over time, there is redundancy between measurements performed by different nodes. In our deployment, sensors that are close to each other measure similar temperatures. Thus, rather than reducing the dimensionality of the data by building a regression model for each sensor in isolation, we can also model spatial correlations, further compressing the data.

More formally, we can view the measurements f as a function of time and of the location of the node. For example, in a 2D deployment our measurements function will be $f(x, y, t)$. This formalism generalizes to 3D deployments, or other more complex parameters of the nodes, such as orientation. In general, we denote the measurement function by $f(\mathbf{x}, t)$, where $\mathbf{x}$ is a vector (*e.g.*, $\mathbf{x} = (x, y, z)$). We can now define basis functions over both spatial parameters (e.g., the nodes' location) and time. In general, we have a set of basis functions H as before, where each basis $h(\mathbf{x}, t)$ is now a function of $\mathbf{x}$ and t. The same regression framework presented in the previous section can be used to compute the optimal weights $\mathbf{w}^*$ of this new basis set. If the position of a sensor changes over time, we can use incremental update rules, analogous to the ones in Equations (3) and (4), to adjust the influence of this sensor in the parameter estimate.

2.3 Kernel regression in sensor nets

The framework described thus far is a powerful tool for modeling the data that is generated by sensor networks. We would like to develop a distributed implementation of the regression optimization that avoids uploading all of the data from the network. However, without further assumptions on the dot-product matrix $\mathbf{A}$, a distributed solution to the linear system $\mathbf{Aw} = \mathbf{b}$ is expensive. As we will demonstrate in the following sections, an efficient, distributed solution is possible when the matrix $\mathbf{A}$ is sparse. We will now present a refinement of the regression method, *kernel (linear) regression* (MN83), that can yield a sparse matrix $\mathbf{A}$ while still preserving the essential structure of sensor network data.

We can view the environment where our sensor network is deployed as consisting of overlapping regions, where measurements within a region have qualitatively similar behaviors. In our lab deployment in Figure 5, we found that temperature measurements tend to be highly correlated within each area of the lab. For example, when the sun shines through the windows, motes near these windows tend to be hot, the ones nearby are a little cooler, while the temperature of motes on the other end of the lab depends on variations across the lab. Thus, we would like to define basis functions that model the phenomena within each region, that appropriately smooth the overlaps between regions, and attenuate the influence of sensors in distant regions.

We can formalize this notion of regions using *kernel functions*. Each region j is defined by a non-negative function $K_j(\mathbf{x})$ that maps each location $\mathbf{x}$ to a non-negative number. The region is defined by the support of K_j, i.e., the set of locations $\mathbf{x}$ where $K_j(\mathbf{x}) > 0$. Kernel functions are otherwise arbitrary; often kernel functions are chosen to decrease smoothly to zero at the boundary of the region to guarantee smoothness of the final regression function $\widehat{f}$.

Using these kernel functions, we can define the *normalized kernel weight* $\kappa_j(\mathbf{x})$ of kernel j at location $\mathbf{x}$,

$$\kappa_j(\mathbf{x}) = \frac{K_j(\mathbf{x})}{\sum_{v=1}^{l} K_v(\mathbf{x})},$$

representing the degree to which the location $\mathbf{x}$ is associated with region j, normalized by the sum of the value of all l at $\mathbf{x}$. If we now associate with each region j a set of basis functions H^j, we can use the normalized kernel weights to combine the weighted basis functions of each region:

$$\begin{aligned} \widehat{f}(\mathbf{x}, t) &= \sum_{j=1}^{l} \kappa_j(\mathbf{x}) \sum_{h_i^j \in H^j} w_i^j h_i^j(\mathbf{x}, t) \,, \\ &= \sum_{j=1}^{l} \sum_{h_i^j \in H^j} w_i^j \left[\kappa_j(\mathbf{x}) h_i^j(\mathbf{x}, t) \right]. \end{aligned} \tag{5}$$

If K_j has no support at location $\mathbf{x}$, then the estimate at $\mathbf{x}$ will not be influenced by the weighted basis functions of region j; if $\mathbf{x}$ is in the support of several kernels, then the estimate is the weighted average of the basis functions of these kernels, where the weights are dictated by the kernel value.

Kernel regression is really a special case of linear regression, since we can view each $\left[\kappa_j(\mathbf{x}) h_i^j(\mathbf{x}, t)\right]$ in Equation (5) as a "basis function." This view allows us to use the same regression framework presented in Section 2.1 to compute jointly the optimal weights of the basis functions associated with each kernel. Figure 2 summarizes the kernel regression framework.

We began this section by motivating kernel regression by the need for a sparse dot-product matrix $\mathbf{A}$. In the kernel regression case, each entry in $\mathbf{A}$ is given by $\langle \kappa_j h_i^j \bullet \kappa_u h_v^u \rangle$. If kernels K_j and K_u do not have any support in common, *i.e.*, their regions do not overlap, then this dot-product is 0. Therefore, if each kernel only overlaps with a few other kernels, then the matrix $\mathbf{A}$ will be very sparse. Furthermore,

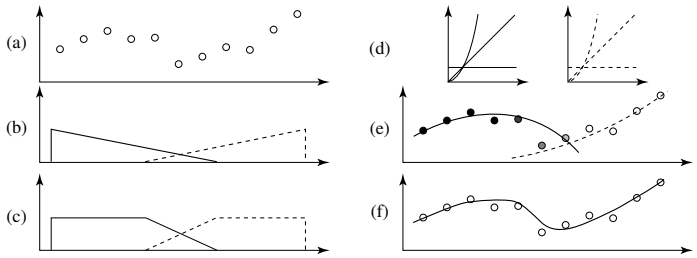

Figure 2: The kernel regression framework: (a) a data set; (b) two overlapping regions defined by simple block kernel functions K_1 and K_2; (c) the normalized kernel weights κ_1 and κ_2; (d) the two sets of basis functions H^1 and H^2; (e) the weighted combination of basis functions in each region; (f) the kernel regression estimate, which is chosen to minimize the mean squared error of all the data.

each entry in the projected measurement vector $\mathbf{b}$ takes the form $\langle \kappa_j h_i^j \bullet f \rangle$. Since κ_j is equal to 0 outside of region j, this entry can be computed using only the measurements obtained in the region j. The sparsity in $\mathbf{A}$ and the locality in the entries of $\mathbf{b}$ are the key properties that will allow us to design an efficient and distributed algorithm for computing the globally optimal basis function coefficients.

3. DISTRIBUTED REGRESSION ALGORITHM

This section describes our network model, query dissemination procedure, and distributed regression algorithm.

3.1 Network model and query dissemination

Nodes in a sensor net can usually communicate with a small set of other nodes. Often in these networks, messages are transmitted along a (spanning) *routing tree* in the network graph. We use E_i to denote the neighbors of N_i in this routing tree. To simplify the presentation, we now focus on networks with a lossless, fixed routing tree. We discuss unreliable communication in Section 3.5. If we are performing spatial queries, we further assume that each node has access to its location (just as it would any other parameter used in the query). If the location of the nodes is not known *a priori*or the nodes' locations change over time, we assume that the positions are obtained by a sensor net localization algorithm (*e.g.*, (PMBT01)). We use $\mathbf{x}_i$ to denote the location (or other required parameter value) of node N_i.

A regression query specifies a set of kernels $\mathcal{K}$, and, for each kernel K_j, a set of basis functions H^j whose parameters we would like to obtain. For example, if we are using simple kernels with rectangular support regions and polynomial basis functions, then the user must specify, for each kernel, the corners of the rectangles and the degree of the polynomial basis that will be associated with this kernel.

We assume that the kernels are disseminated through the network along with the query, using query dissemination techniques, such as those used in TinyDB (Mad03) or directed diffusion (IGE00). After this dissemination phase, each node N_i will have access to a list $\mathcal{K}_i$ of kernels that have non-zero value at location $\mathbf{x}_i$. Note that if a node's location changes, then this node must have access to the kernels that have support at this new location. This information can be obtained from nearby sensors using an extension of the query dissemination procedure.

Once the query has been disseminated, if node N_i needs to compute the value of our kernel regression function at its location, N_i only needs access to the coefficients of the basis functions associated with the kernels in $\mathcal{K}_i$:

$$\begin{aligned}\widehat{f}(\mathbf{x}_i,t) &= \sum_{j=1}^{l} \kappa_j(\mathbf{x}_i) \sum_{h_u^j \in H^j} w_u^j h_u^j(\mathbf{x}_i,t), \\ &= \sum_{K_j \in \mathcal{K}_i} \kappa_j(\mathbf{x}_i) \sum_{h_u^j \in H^j} w_u^j h_u^j(\mathbf{x}_i,t),\end{aligned}$$

as the term $\kappa_j(\mathbf{x}_i)$ is equal to 0 for any kernel that is not in $\mathcal{K}_i$. We use $\mathbf{C}_i$ to denote the index of the basis function coefficients w_u^j such that $K_j \in \mathcal{K}_i$, that is, the set of basis function coefficients that node N_i must access in order to compute our regression function at $\mathbf{x}_i$.

3.2 A Gaussian elimination step in kernel distributed regression

Our distributed regression algorithm builds on a distributed application of Gaussian elimination (GV89) to solve the linear system $\mathbf{Aw} = \mathbf{b}$ for kernel regression described in Section 2.3. We begin by showing a simple example with two nodes and three kernels (each with one basis function), where $\mathcal{K}_1 = \{K_1, K_2\}$ and $\mathcal{K}_2 = \{K_2, K_3\}$, $\mathbf{C}_1 = \{1, 2\}$ and $\mathbf{C}_2 = \{2, 3\}$. In this simple case, we have a simple (sparse) linear system with 3 equations and 3 unknowns:

$$\begin{pmatrix} a_{11}^{(1)} & a_{12}^{(1)} & 0 \\ a_{21}^{(1)} & a_{22}^{(1)} + a_{22}^{(2)} & a_{23}^{(2)} \\ 0 & a_{32}^{(2)} & a_{33}^{(2)} \end{pmatrix} \begin{pmatrix} w_1 \\ w_2 \\ w_3 \end{pmatrix} = \begin{pmatrix} b_1^{(1)} \\ b_2^{(1)} + b_2^{(2)} \\ b_3^{(2)} \end{pmatrix},$$

where

$$\begin{aligned} b_i^{(u)} &= \textstyle\sum_{k=1}^{m} \left[\kappa_i(\mathbf{x}_u, t_k) h_i(\mathbf{x}_u, t_k)\right] f(\mathbf{x}_u, t_k), \\ a_{ij}^{(u)} &= \textstyle\sum_{k=1}^{m} \left[\kappa_i(\mathbf{x}_u, t_k) h_i(\mathbf{x}_u, t_k)\right] \left[\kappa_j(\mathbf{x}_u, t_k) h_j(\mathbf{x}_u, t_k)\right]. \end{aligned}$$

Note, for example, that $a_{13}^{(1)} = 0$ as node N_1 is not in the support region of kernel K_3, *i.e.*, $\kappa_3^{(1)}$. In Gaussian elimination, we turn a full matrix into a triangular matrix by subtracting equality constraints from each other appropriately. For example, we can multiply the first constraint by $a_{21}^{(1)}/a_{11}^{(1)}$ and then subtract this constraint from the second one, obtaining:

$$\begin{pmatrix} a_{11}^{(1)} & a_{12}^{(1)} & 0 \\ 0 & a_{22}^{(1)} + a_{22}^{(2)} - \frac{a_{21}^{(1)}}{a_{11}^{(1)}} a_{12}^{(1)} & a_{23}^{(2)} \\ 0 & a_{32}^{(2)} & a_{33}^{(2)} \end{pmatrix} \begin{pmatrix} w_1 \\ w_2 \\ w_3 \end{pmatrix}$$

$$= \begin{pmatrix} b_1^{(1)} \\ b_2^{(1)} + b_2^{(2)} - \frac{a_{21}^{(1)}}{a_{11}^{(1)}} b_1^{(1)} \\ b_3^{(2)} \end{pmatrix},$$

The lower two rows now becomes a linear system with 2 equations and 2 unknowns that does not depend on w_1. If we solve this linear system, we obtain the values of the weights w_2 and w_3. At this point, w_2 can be substituted into the first constraint, yielding the value of of w_1.

This simple example illustrates our algorithm. Nodes N_1 and N_2 will maintain a distributed representation of the $\mathbf{A}$ matrix and of the $\mathbf{b}$ vector: $\left(\mathbf{A}^{(1)}, \mathbf{b}^{(1)}\right)$ in N_1 and $\left(\mathbf{A}^{(2)}, \mathbf{b}^{(2)}\right)$ in N_2, such that: $\mathbf{A} = \mathbf{A}^{(1)} + \mathbf{A}^{(2)}$, and $\mathbf{b} = \mathbf{b}^{(1)} + \mathbf{b}^{(2)}$, where,

$$\mathbf{A}^{(1)} = \begin{pmatrix} a_{11}^{(1)} & a_{12}^{(1)} & 0 \\ a_{21}^{(1)} & a_{22}^{(1)} & 0 \\ 0 & 0 & 0 \end{pmatrix}, \qquad \mathbf{b}^{(1)} = \begin{pmatrix} b_1^{(1)} \\ b_2^{(1)} \\ 0 \end{pmatrix},$$

and similarly for node N_2. Now, if we want to repeat the Gaussian elimination step described above, node N_1 can compute the terms

$$\left(a_{22}^{(1)} - (a_{21}^{(1)}/a_{11}^{(1)})a_{12}^{(1)}\right) \quad \text{and} \quad \left(b_2^{(1)} - (a_{21}^{(1)}/a_{11}^{(1)})b_1^{(1)}\right)$$

locally from the $\mathbf{A}^{(1)}$ matrix and $\mathbf{b}^{(1)}$ vector. These terms can then be sent to N_2 as a message $\left(\mathbf{A}^{(12)}, \mathbf{b}^{(12)}\right)$. Once N_2 receives this message, it can compute the value of the weights $\mathbf{w}_{(\mathbf{C}_2)} = (w_2, w_3)^\intercal$ by solving a local linear system with two equations and two unknowns:

$$\left(\mathbf{A}^{(2)} + \mathbf{A}^{(12)}\right)\mathbf{w}_{(\mathbf{C}_2)} = \mathbf{b}^{(2)} + \mathbf{b}^{(12)} \ .$$

Similarly, if node N_2 sends N_1 an analogous message with the matrix $\mathbf{A}^{(21)}$ and vector $\mathbf{b}^{(21)}$, then node N_1 can compute the optimal value of the weights in $\mathbf{C}_1$ by solving the local linear system $\left(\mathbf{A}^{(1)} + \mathbf{A}^{(21)}\right)\mathbf{w}_{(\mathbf{C}_1)} = \mathbf{b}^{(1)} + \mathbf{b}^{(21)}$. (Note that, whereas in standard Gaussian elimination we would have a *back-substitution* step, these symmetric messages avoid back-substitution; this symmetry simplifies our algorithm in the presence of communication failures.)

Now suppose we have a larger linear system with similar structure. Each $\mathbf{A}^{(i)}$ and $\mathbf{b}^{(i)}$ will have zero entries for all rows and columns whose indices are not in $\mathbf{C}_i$. Let the intersection between the two clusters be denoted by $\mathbf{S} = \mathbf{C}_1 \cap \mathbf{C}_2$, and the remainder variables be $\mathbf{V}_1 = \mathbf{C}_1 \setminus \mathbf{S}$ and $\mathbf{V}_2 = \mathbf{C}_2 \setminus \mathbf{S}$. Using this notation, we can write our linear system as:

$$\begin{pmatrix} \mathbf{A}^{(1)}_{(\mathbf{V}_1,\mathbf{V}_1)} & \mathbf{A}^{(1)}_{(\mathbf{V}_1,\mathbf{S})} & \mathbf{0} \\ \mathbf{A}^{(1)}_{(\mathbf{S},\mathbf{V}_1)} & \mathbf{A}^{(1)}_{(\mathbf{S},\mathbf{S})} + \mathbf{A}^{(2)}_{(\mathbf{S},\mathbf{S})} & \mathbf{A}^{(2)}_{(\mathbf{S},\mathbf{V}_2)} \\ \mathbf{0} & \mathbf{A}^{(2)}_{(\mathbf{V}_2,\mathbf{S})} & \mathbf{A}^{(2)}_{(\mathbf{V}_2,\mathbf{V}_2)} \end{pmatrix} \mathbf{w} = \begin{pmatrix} \mathbf{b}^{(1)}_{(\mathbf{V}_1)} \\ \mathbf{b}^{(1)}_{(\mathbf{S})} + \mathbf{b}^{(2)}_{(\mathbf{S})} \\ \mathbf{b}^{(2)}_{(\mathbf{V}_2)} \end{pmatrix},$$

where the notation $\mathbf{A}_{(\mathbf{S},\mathbf{V})}$ indicates the rows and columns of $\mathbf{A}$ corresponding to the weights $\mathbf{S}$ and $\mathbf{V}$, respectively. Similarly, $\mathbf{b}_{(\mathbf{S})}$ denotes the rows of $\mathbf{b}$ corresponding to the weights in $\mathbf{S}$. We can generalize the Gaussian elimination step described above to eliminate the weights in $\mathbf{V}_1$ from all rows in $\mathbf{S}$ simultaneously. The message from N_1 to N_2 is now composed of a matrix $\mathbf{A}^{(12)}$ and a vector $\mathbf{b}^{(12)}$, over the shared variables $\mathbf{S}$:

$$\begin{aligned} \mathbf{A}^{(12)} &= \left(\mathbf{A}^{(1)}_{(\mathbf{S},\mathbf{S})} - \mathbf{A}^{(1)}_{(\mathbf{S},\mathbf{V}_1)} \left(\mathbf{A}^{(1)}_{(\mathbf{V}_1,\mathbf{V}_1)}\right)^{-1} \mathbf{A}^{(1)}_{(\mathbf{V}_1,\mathbf{S})} \right), \\ \mathbf{b}^{(12)} &= \left(\mathbf{b}^{(1)}_{(\mathbf{S})} - \mathbf{A}^{(1)}_{(\mathbf{S},\mathbf{V}_1)} \left(\mathbf{A}^{(1)}_{(\mathbf{V}_1,\mathbf{V}_1)}\right)^{-1} \mathbf{b}^{(1)}_{(\mathbf{V}_1)} \right), \end{aligned} \tag{6}$$

where $\mathbf{A}^{(12)}$ is a square matrix and $\mathbf{b}^{(12)}$ a vector whose sizes depend on the number of weights in $\mathbf{S}$. Once N_2 receives this message, it computes the optimal value of the variables in $\mathbf{C}_2$ by solving a simple, completely local, linear system.

3.3 Regression messages

Our complete algorithm, shown in Figure 3, is a generalization of this two node example. Each node N_i maintains a matrix $\mathbf{A}^{(i)}$ and a vector $\mathbf{b}^{(i)}$ that summarize the effect of this node's measurements in the dot-product matrix and the projected measurement vector, respectively, where the complete matrix and vector are given by:

$$\mathbf{A} = \textstyle\sum_{i=1}^{n} \mathbf{A}^{(i)} \ , \qquad \mathbf{b} = \textstyle\sum_{i=1}^{n} \mathbf{b}^{(i)}.$$

DistributedRegression(i),
On event: Initialize():
  $T \leftarrow$ size of time window.
  MsgInterval $\leftarrow$ timer interval to send messages.
  $\varepsilon \leftarrow$ precision.
  $E_i \leftarrow$ neighbors of N_i in the routing tree.
  $\mathbf{C}_i \leftarrow$ cluster of variables in node N_i.
  $\mathbf{A}^{(i)} \leftarrow \mathbf{0}, \quad \mathbf{b}^{(i)} \leftarrow \mathbf{0}, \quad \mathbf{w}^{(i)} \leftarrow \mathbf{0}$.
  For each $j \in E_i$:
    $\mathbf{C}_j \leftarrow$ cluster of variables in node N_j.
    $\mathbf{A}^{(ij)} \leftarrow \mathbf{0}, \quad \mathbf{b}^{(ij)} \leftarrow \mathbf{0}$.
    $\mathbf{A}^{(ji)} \leftarrow \mathbf{0}, \quad \mathbf{b}^{(ji)} \leftarrow \mathbf{0}$.

On event: NewData(t,v):
//at time t the sensor measured value v.
  For each element a_{uv} of $\mathbf{A}^{(i)}$, where $K_u \in \mathcal{K}_i$ and $K_v \in \mathcal{K}_i$:
    $a_{uv} \leftarrow a_{uv} + \kappa_u(\mathbf{x}_i)h_u(\mathbf{x}_i,t)\kappa_v(\mathbf{x}_i)h_v(\mathbf{x}_i,t)$.
  For each element b_u of $\mathbf{b}^{(i)}$, where $K_u \in \mathcal{K}_i$:
    $b_u \leftarrow b_u + \kappa_u(\mathbf{x}_i)h_u(\mathbf{x}_i,t)\ v$.
  Start event DeleteData(t,v) at time $t+T$.

On event: DeleteData(t,v):
//delete measurement of v from time t.
  For each element a_{uv} of $\mathbf{A}^{(i)}$, where $K_u \in \mathcal{K}_i$ and $K_v \in \mathcal{K}_i$:
    $a_{uv} \leftarrow a_{uv} - \kappa_u(\mathbf{x}_i)h_u(\mathbf{x}_i,t)\kappa_v(\mathbf{x}_i)h_v(\mathbf{x}_i,t)$.
  For each element b_u of $\mathbf{b}^{(i)}$, where $K_u \in \mathcal{K}_i$:
    $b_u \leftarrow b_u - \kappa_u(\mathbf{x}_i)h_u(\mathbf{x}_i,t)\ v$.

On event: ReceivedMessage(j,$\bar{\mathbf{A}}^{(ji)}$,$\bar{\mathbf{b}}^{(ji)}$):
  $\mathbf{A}^{(ji)} \leftarrow \bar{\mathbf{A}}^{(ji)}$, and $\mathbf{b}^{(ji)} \leftarrow \bar{\mathbf{b}}^{(ji)}$.

On event: SolveLocalLinearSystem():
  $\mathbf{w}^{(i)} \leftarrow \left(\mathbf{A}^{(i)} + \sum_{j\in E_i} \mathbf{A}^{(ji)}\right)^{-1} \left(\mathbf{b}^{(i)} + \sum_{j\in E_i} \mathbf{b}^{(ji)}\right)$.

Every MsgInterval:
//procedure to send messages.
  For each $j \in E_i$:
    $\bar{\mathbf{A}} \leftarrow \mathbf{A}^{(i)} + \sum_{k\in E_i\setminus j} \mathbf{A}^{(ki)}, \quad \bar{\mathbf{b}} \leftarrow \mathbf{b}^{(i)} + \sum_{k\in E_i\setminus j} \mathbf{b}^{(ki)}$.
    $\mathbf{S} \leftarrow \mathbf{C}_i \cap \mathbf{C}_j, \quad \mathbf{V} \leftarrow \mathbf{C}_i \setminus \mathbf{S}$.
    $\bar{\mathbf{A}}^{(ij)} \leftarrow \bar{\mathbf{A}}_{(\mathbf{S},\mathbf{S})} - \bar{\mathbf{A}}_{(\mathbf{S},\mathbf{V})} \left(\bar{\mathbf{A}}_{(\mathbf{V},\mathbf{V})}\right)^{-1} \bar{\mathbf{A}}_{(\mathbf{V},\mathbf{S})}$.
    $\bar{\mathbf{b}}^{(ij)} \leftarrow \bar{\mathbf{b}}_{(\mathbf{S})} - \bar{\mathbf{A}}_{(\mathbf{S},\mathbf{V})} \left(\bar{\mathbf{A}}_{(\mathbf{V},\mathbf{V})}\right)^{-1} \bar{\mathbf{b}}_{(\mathbf{V})}$.
    If $\left\|\bar{\mathbf{A}}^{(ij)} - \mathbf{A}^{(ij)}\right\|_\infty > \varepsilon$ or $\left\|\bar{\mathbf{b}}^{(ij)} - \mathbf{b}^{(ij)}\right\|_\infty > \varepsilon$:
      $\mathbf{A}^{(ij)} \leftarrow \bar{\mathbf{A}}^{(ij)}, \quad \mathbf{b}^{(ij)} \leftarrow \bar{\mathbf{b}}^{(ij)}$.
      Send message $\left(i, \mathbf{A}^{(ij)}, \mathbf{b}^{(ij)}\right)$ to node N_j.

Figure 3: Distributed regression algorithm.

When the node observes a new value, its local matrix and vector are updated using the incremental rule described in Section 2.1, and an event is scheduled to delete this value when it falls outside the time window.

Messages between two neighboring nodes N_i and N_j are composed of a square matrix and a vector with an entry for each variable that these two nodes share, $\mathbf{C}_i \cap \mathbf{C}_j$, as in the simple case above. These messages are computed with the generalized Gaussian elimination step. However, we now have more than two nodes, and when a node computes a message to a neighbor in the routing tree, it must take into account messages it has received from other neighboring nodes. The structure of these recursive messages follows a *nonserial dynamic programming* decomposition of the dot-product matrix (BB72). Intuitively, when a node N_i sends messages to a neighbor, it simply adds the messages of the other neighbors to its $\mathbf{A}^{(i)}$ matrix and $\mathbf{b}^{(i)}$ vector, and computes a message as in the simple case in Equation (6). (We omit further details due to lack of space, referring the reader to (PL03; BB72; GV89)).

The complexity of this algorithm depends on the size of the cluster $\mathbf{C}_i$ of each node. Let c be the size of the largest

cluster in the network. The messages between any two nodes are, in the worst, of size $c^2 + c$, *i.e.*, a matrix and a vector of size no larger than the number of weights in each node. It is important to note that the size of the message does not depend on the number of measurements performed by the node. For a network with n nodes, using an appropriate implementation, the sum of all messages required to propagate the regression information throughout the network is, in the worst case, $2n(c^2 + c)$. That is, we must make two passes through the network to ensure that the effect of the measurements of each node are propagated to every other node. If we would like to upload the basis function coefficients to a base station, we need, in the worst case, a total of dk additional communication, where d is the depth of the routing tree, as the coefficient of each basis function has to be propagated to the base station.

In our algorithm, each node stores the last message it sent to each neighbor. Using this information, the node can decide not to send a message again, if it has not changed significantly since the last message was sent. If the links have bit-rate constraints, then it may be possible to apply standard theoretical results on the effect of bit-precision on Gaussian elimination (GV89) to evaluate the error introduced by a lower precision representation of the messages.

At any point in running of our algorithm, the node can decide to solve its local linear system to obtain a preliminary estimate of the value of its weights.

3.4 Distributed construction of junction trees

The message passing algorithm presented above is guaranteed to compute the optimal weights for each basis function, but only when the tree of clusters has the *running intersection property* (CDLS99):

DEFINITION 1. *A cluster tree satisfies the* running intersection property *if for each pair of clusters* $\mathbf{C}_i$ *and* $\mathbf{C}_k$*, every cluster* $\mathbf{C}_j$ *on the (unique) path between* $\mathbf{C}_i$ *and* $\mathbf{C}_k$ *contains* $\mathbf{C}_i \cap \mathbf{C}_k$*. If this property is satisfied, then we call the tree a* junction tree. □

In the distributed regression algorithm presented above, two neighbors N_i and N_k in the routing tree communicate information only about the basis functions that they have in common, *i.e.*, the ones in $\mathbf{C}_i \cap \mathbf{C}_k$. However, N_i and N_k must communicate this information even if they are not neighbors in this tree. The running intersection property guarantees that all nodes on the path between N_i and N_k are representing and communicating information about $\mathbf{C}_i \cap \mathbf{C}_k$; this ensures that N_i and N_k can pool their information about the basis functions they share, even if they are not neighbors in the routing tree. When the running intersection property holds, the correctness of our distributed algorithm follows by induction on the structure of the junction tree (PL03).

To ensure the correctness of the distributed regression algorithm, it is therefore necessary to enforce the running intersection property (if it does not already hold). This can be accomplished by a simple message passing algorithm in which each pair of nodes in the routing tree communicate information regarding the initial clusters; the nodes then add basis function coefficients to their clusters in such a way that the required property holds. We can regard this algorithm as a deployment-time computation (it must be done only once for a fixed set of kernels and basis functions) that determines the minimal amount of information nodes must communicate in order to reach the globally optimal solution.

DISTRIBUTEDRUNNINGINTERSECTION(N_i),
On event: Initialize():
  $\mathcal{K}_i \leftarrow$ KERNELS THAT HAVE NON-ZERO SUPPORT AT $\mathbf{x}_i$.
  $E_i \leftarrow$ NEIGHBORS OF N_i IN THE ROUTING TREE.
  $\overline{\mathbf{C}}_i \leftarrow \bigcup_{K_j \in \mathcal{K}_i} \bigcup_{h_u^j \in H^j} \{w_u^j\}$.
  For EACH $j \in E_i$:
    $\mathbf{M}_{ji} \leftarrow \emptyset$, $\text{MSGSENT}_j \leftarrow false$.
  If $|E_i| = 1$:
    Start event SendMessages().

On event: ReceivedMessage(j,$\overline{\mathbf{M}}_{ji}$):
  $\mathbf{M}_{ji} \leftarrow \overline{\mathbf{M}}_{ji}$.
  Start event SendMessages().
  If $\forall k \in E_i : \mathbf{M}_{ki} \neq \emptyset$:
    Start event DoneRunningIntersection().
On event: SendMessages():
  For EACH $j \in E_i$:
    If $\text{MSGSENT}_j = false$ AND $\forall k \in E_i \setminus j : \mathbf{M}_{ki} \neq \emptyset$:
      Send message $\left(i, \mathbf{M}_{ij} = \overline{\mathbf{C}}_i \cup \bigcup_{k \in E_i \setminus j} \mathbf{M}_{ki}\right)$ TO NODE N_j.
      $\text{MSGSENT}_j = true$.

On event: DoneRunningIntersection():
  $\mathbf{C}_i \leftarrow \overline{\mathbf{C}}_i$
  For EACH $j \in E_i$:
    For EACH $k \in E_i \setminus j$:
      $\mathbf{C}_i \leftarrow \mathbf{C}_i \cup (\mathbf{M}_{ji} \cap \mathbf{M}_{ki})$.
  //Now local cluster of N_i satisfies the running intersection property.

Figure 4: Distributed algorithm to enforce the running intersection property.

The running intersection property can be enforced by a message passing algorithm, where each node N_i is initialized with $\overline{\mathbf{C}}_i$, the basis functions coefficients for the kernels that have support at $\mathbf{x}_i$. Each node N_i then sends to its neighbors N_j a *variables message* $\mathbf{M}_{ij}$ that is the union of all variables that appear in the initial clusters $\overline{\mathbf{C}}$ in every node in the subtree rooted at N_i (besides N_j). Thus, node N_i can guarantee that the running intersection property is satisfied if, when any variable appears in the subtree of one neighbor N_j and in the subtree of another neighbor N_k, then this variable appears in the final cluster $\mathbf{C}_i$ of node N_i. Our algorithm recursively computes the variables messages starting from the leaves of the tree. When node N_i has received messages from all neighbors, it is ready to compute its cluster $\mathbf{C}_i$. Figure 4 presents the complete distributed algorithm for finding clusters that satisfy the running intersection property. This algorithm is specified in terms of tasks, messages and responses to events, allowing us to implement it as an asynchronous distributed algorithm.

It is important to note that the complexity of our algorithm depends on the size of the final clusters $\mathbf{C}_i$. Different routing trees will yield different clusters. The algorithm we present in (PG03) attempts to optimize the routing tree in order to minimize the communication requirements of a wide range of algorithms, including the one in this paper.

3.5 Extension to unreliable networks

For simplicity of exposition we have focused on networks with reliable communication, but this is often an unrealistic assumption. Wireless transmission is lossy and network nodes can fail. Fortunately, the distributed regression algorithm can be made robust to such failures. In this section we sketch out the necessary extensions; for further details see (PG03).

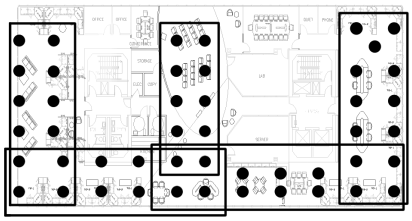

Figure 5: A map of the Intel - Berkeley lab deployment, with the placement of 47 sensors show in dark circles. The 5 rectangles indicate the support regions of the kernels used in the model.

We can view the distributed regression algorithm as consisting of three layers: the *routing layer*, which builds a spanning tree such that adjacent nodes have high quality communication links; the *junction tree layer*, which sends variable messages between nodes adjacent in the routing tree to enforce the running intersection property; and the *regression layer*, which sends messages about the basis function coefficients to optimize the regression estimate.

Routing trees are fundamental to communication in unreliable ad hoc networks, and there are several algorithms to choose from (HSW+00). We require a routing layer which can notify the junction tree and regression layers when the topology of the routing tree changes. When this happens, the junction tree layer can recompute and retransmit the variables messages to restore the running intersection property on the new routing tree. After this is accomplished, the regression layer can retransmit its messages to reoptimize the model. The junction tree and regression layers can be made robust to lossy communication by using acknowledgements for messages. In this setting, we can be assured that if the routing layer can find a good routing tree, then the algorithm will eventually converge to the correct result.

An alternative approach would be to fix a junction tree, viewed as an overlayed communication pattern on the network, and use multihop communication on a routing tree to achieve this communication pattern, removing the need for regenerating the junction tree when the underlying routing tree changes. Unfortunately, such point-to-point multihop communication is often very hard to implement on a real (unreliable) sensor network. By aligning our junction tree with the underlying routing tree, we can simply rely on local communication to optimize the coefficients of our basis functions.

Interestingly, our algorithm is very robust to packet losses and network failures. For example, if the network is segmented into two parts, each part will optimize its coefficients from the values of the measurements in that part of the network. When the connection returns, these estimates are quickly fused into a globally optimal solution.

4. EVALUATION AND APPLICATIONS

In this section, we look at the performance of the distributed regression algorithm presented above, showing that it provides a compact way, low-error way to represent sensor fields. We also look at several applications of our approach, showing that it can be used to answer a wide range of queries.

4.1 Evaluation of basic algorithm

To evaluate our distributed regression algorithm, we implemented it in TOSSIM (LLW+03), a whole program simulator

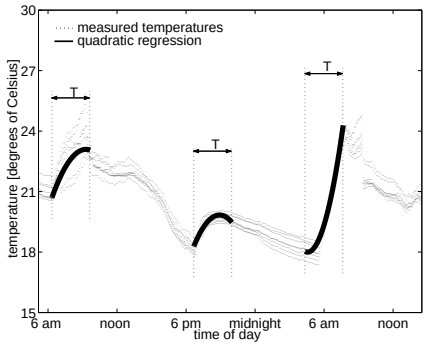

Figure 6: Quadratic regression (thick black curves) on data obtained from several sensors over a two day period. Fits are shown for several different sliding windows, with a sliding window size of 200 minutes.

for TinyOS. Aside from the issues associated with network loss discussed above, we expect that this implementation will run directly on motes. This simulator allows us to model a variety of network topologies and kernel functions, and to measure the communication costs of our algorithms in a realistic setting. In the experiments below, we used this simulator to collect measurements on the communication requirements of our algorithm.

In our initial evaluation, we ran our algorithm on a dataset of 2-minute samples of light, temperature, and humidity collected from our lab deployment of 48 sensors. Figure 5 shows a map of this deployment, with the placement of sensors indicated by small dark circles. The data measured in the lab is quite complex: the sun affects different areas of the lab at different times of the day, and the measurements have high spatial correlation, but include local variations due to the proximity to windows and air conditioning vents. We believe that the temporal and spatial properties of this dataset will also be present in many other sensor nets deployments.

By hand, we mapped 5 kernels on to this space, representing the 5 black regions shown in Figure 5. In this mapping, every sensor is in at least one kernel, while some sensors lie in two or three kernels. Intuitively, this represents the correlation across space – readings from the same area of space tend to be highly correlated, while readings from adjacent areas mix along their common edges. To give a sense of how regression performs, Figure 6 illustrates regression with a quadratic model over the data from several sensors over a two day period.

One application of regression is generating 3-dimensional and contour plots of a space, giving a picture of how temperature varies and changes over space and time. Figure 7 illustrates the results of our regression algorithm at different points in time, illustrating the large variations of the temperature in the lab. Figure 8 shows a contour plot of the temperature across our lab at 10 AM on October 28th – notice that the temperature varies from about 20 °C in the lower left corner to 29 °C in the upper right.

We also measured the error of the model constructed by running our distributed regression algorithm, varying two parameters: the number of basis functions, k and the size of the sliding window, T. We measured the root-mean-squared (RMS) between the model, $\widehat{f}(x, y, t)$, and the value at every point in the data set, $D(x, y, t)$.

To measure the ability of regression to predict the value at locations in the sensor field where there are no readings, we also experimented with subsampling of the data set D to a

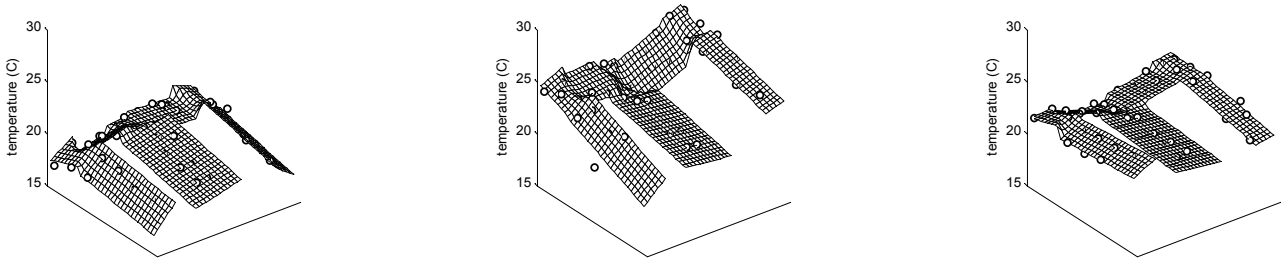

Figure 7: Kernel regression model obtained from temperature data collected at the Intel Research, Berkeley lab, using 5 kernel regions, with 3 basis function per region, at different times of the day (the circles represent the actual temperature at the sensor locations): (a) at night, locations near windows are colder; (b) in the morning, the East side of the lab faces the sun, significantly increasing the temperature; and (c) in the early evening, the temperature is uniformly warm.

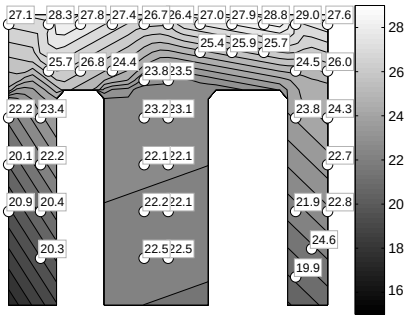

Figure 8: A contour plot generated by running kernel-based quadratic regression on the data collected at 10 AM on October 28th in the Intel Research, Berkeley lab. The labels represent the actual temperatures measured at the sensor locations. Note that this rich contour was obtained from a regression model with only 15 parameters.

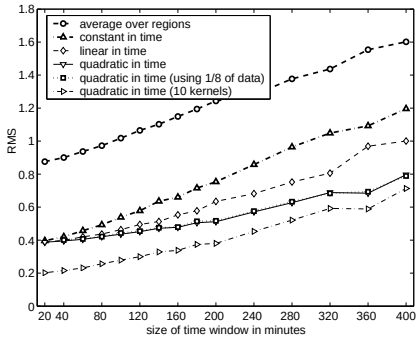

Figure 9: The RMS error of regression with varying time windows and numbers of basis functions per kernel for the data set collected from the Intel - Berkeley Lab, compared against simple averaging in each kernel.

dataset with 1/8th of the of the original data, and measured the RMS of regression applied to this dataset versus D.

The results of these error measurements for different basis sets with varying time windows and subsampling are shown in Figure 9. We experimented with three different basis function sets per kernel: either (1) linear-space, quadratic-time (e.g., $\widehat{f}(x,y,t) = c_1(x) + c_2(y) + c_3(t^2) + c_4(t) + c_5$), (2) linear-space, linear-time, or (3) linear-space, constant-time. We also measured the RMS of simply computing the average value of the readings in each kernel over the time window T. Note that regression performs quite well compared to averaging, and that, as expected, increasing the number of basis functions increases the quality of the fit. Surprisingly, regression using the reduced data set (with 1/8 the points) performs as well as regression with the entire data set; this is likely due to low variations in temperature within an 8 reading (16 minute) window.

Since average error over an entire data set does not capture the worst case performance of these approaches, we also plotted the error of these schemes at different times of day, using a time window size of two hours. The results of this experiment are shown in Figure 10. Notice that the linear and quadratic fit perform much better during times when the temperature changes dramatically (*e.g.*, when the sun rises and sets – see Figure 6 for a plot of the temperature reading from several sensors), but that all schemes perform well at times of low variance (*e.g.*, at night). This suggests that an adaptive scheme, where different numbers of basis functions are used depending on signal variability may be beneficial; such an exploration is left for future work.

We also measured the communication costs for our lab deployment (using 3 coefficients per kernel) and found that, in TOSSIM, the total number of bytes sent by all sensors was 5808 bytes versus 875 bytes to extract a single reading from every sensor. After this 5800 bytes of communication, each node can predict the behavior of the sensor field in its local area, which, as we describe in Section 5 can be very important. Furthermore, although the total communication is relatively high for our small lab network, the total bytes sent by the root of the network was just 123 bytes for our algorithm versus 329 bytes when extracting all readings. In the next section, we study the scaling of our algorithm as the network size increases, noting that the cost of distributed regression grows quite slowly with network size, unlike the cost of extracting all readings.

4.2 Convergence rate and robustness to losses

The rate of convergence of our algorithm and its robustness lost messages is best understood by analyzing the asynchronous version of the algorithm shown in Figure 3, which we implemented in TinyOS and evaluated it with TOSSIM. When the nodes first start running, the message matrices are initialized to zero, and the approximation error should be high. As messages are received, the node can update the values of these matrices, and the estimates improve as the information is propagated throughout the network. This ef-

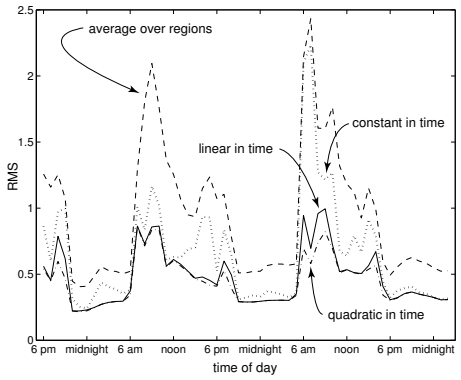

Figure 10: The error of different regression models for the lab data set at different times of day, using a time window size of 2 hours.

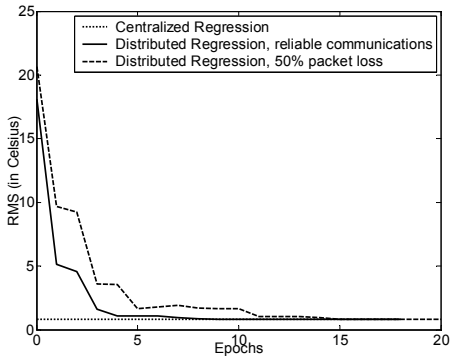

Figure 11: Convergence curves of the initialization phase of the distributed implementation of our algorithm for a model with 5 kernels and 3 basis functions per kernel on temperature measurements made at noon.

fect is seen clearly in Figure 11 that shows the RMS error in the model for the temperatures at noon, using 5 kernels with linear basis over space and no basis functions over time. The coefficients of the basis functions for a region are extracted from one representative in that region. In each epoch, every node sends a message to each of its neighbors in the routing tree. The graph shows that after only 7 epochs the results reach the optimal value, having the same RMS error as off-line regression. This result is quite significant, given that our routing tree has nodes that are 20 hops away from each other. We can also see that the method is very robust to lost messages. Figure 11 also shows that even when packets are lost with 50% (independent) probability, the algorithm converges very fast to the optimal solution.

These results focus in the initialization phase. However, at steady-state, we already have a converged estimate taken earlier in time, and we only need to improve this estimate to take into account the new values observed. Figure 12 shows similar convergence results for a setting where the nodes start from the converged values for the temperature measurements at noon and substitute the measurements with those taken at 6pm. We first note that, as expected, the RMS error starts from a significantly lower value than in the initialization phase. Furthermore, the convergence is faster for both the reliable and the lossy communication cases. These results suggest that, at steady-state, we can achieve the same error levels at lower communication rate than suggested by the worst-case complexity of the algorithm.

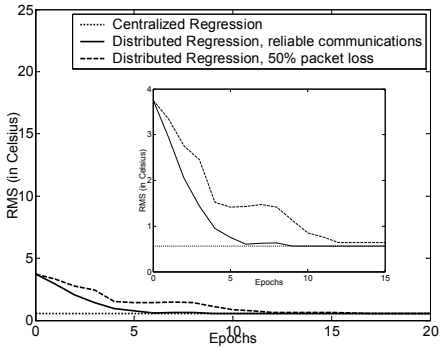

Figure 12: Convergence curves of the steady-state phase of the distributed implementation of our algorithm for a model with 5 kernels and 3 basis functions per kernel on temperature measurements made at 6pm, where the initialization was done at noon. The outer graph is in the same scale as Figure 11, while the inner graph shows a more detailed scale.

4.3 Scaling the network

To study the scaling of our algorithm, we varied the number of motes in the network, keeping the number of motes per kernel constant (at 20 – the approximate value in our lab deployment) and using 4 coefficients per kernel (e.g., linear-space, linear-time). As in our lab deployment, we assume each sensor is in no more than two kernels. We compared the cost of running our distributed regression algorithm and extracting all of the basis function coefficients from every kernel to the cost of extracting a complete set of readings from all nodes. We studied the maximum number of messages sent by any node in the network. In both the distributed regression and data extraction cases, the node where the data is extracted (i.e., the root) sends the most messages. We chose to study maximum communication because this dictates the maximum sample rate of the network as well as the time until failure of the root. The number of messages sent at the root is also independent of network topology, allowing us to avoid having to make (likely unrealistic) assumptions about the topology and node distribution in a large, synthetic network.

Figure 13 shows the results of this scaling experiment, plotting the number of bytes sent at the root node for network sizes up to 1000 nodes. The three lines represent the cost of extracting all data at every second to the cost of extracting all data every 5 seconds and the cost of running our distributed regression algorithm every 5 seconds (although the regression is still run on data collected every second). Figure 13 also shows the maximum number of bytes that a current generation Mica2 mote can send in a second, based on the typical performance of its ChipCon CC1000 radio running TinyOS – about 20, 36 byte messages per second. Notice that distributed regression places a substantially lower communications burden on the root, allowing the performance to scale to much larger networks or higher update rates before the root link is saturated.

5. CONCLUSIONS AND FUTURE WORK

In this paper, we discussed the application of kernel-based regression for modeling sensor fields. We showed that, using our a distributed regression algorithm based on junction trees, it is possible to compute the coefficients of kernel-based regression models efficiently within a sensor network. Our

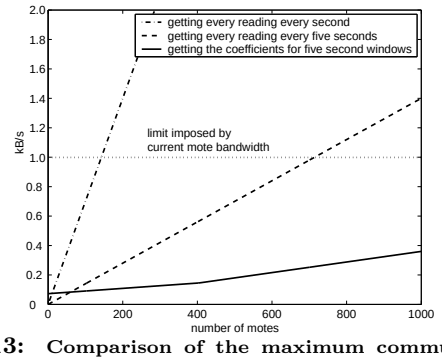

Figure 13: Comparison of the maximum communication requirements at the root of a sensor network for distributed regression versus extracting every sample from every node.

experimental results demonstrate that the approach is capable of accurately summarizing and predicting values of sensor fields using small amounts of communication, and that this algorithm scales well to large sensor networks of hundreds of sensors. We also show, on a simulated distributed implementation, that our algorithm converges to the globally optimal solution at a fast rate, and that our approach is very robust to packet losses.

Besides the enabling a range of basic aggregation and summarization techniques, such as building contour plots, there are a number of sophisticated in-network applications of our algorithm that we believe are important future directions of this work. We conclude by summarizing a few of these that have we believe have significant practical implications for sensor networks:

Adaptive Sampling. Rather than delivering samples at a continuous rate, as most sensor-network query systems propose (Mad03; YG03), our approach allows nodes to locally suppress messages that are very near to the value predicted by the regression model. Since, after running the junction tree algorithm each node has a local image of the basis functions to which it contributes (*i.e.*, where its kernel weight is non-zero), this can be done at no additional communications cost. Specifically, a node can measure the RMS and choose to incrementally update its basis coefficients only when the RMS varies by some query-specific threshold.

Outlier and faulty sensor detection. Our distributed regression algorithm also allows sensors to determine if a particular reading is an outlier using only their local basis functions. The mechanism for doing this is similar to that used in adapting sample rates: when a new reading substantially affects the coefficients of the basis functions, or lies far from the value that the regression predicts, it is an outlier. In addition to adapting the model, however, it may be useful to tag such readings as outliers and transmit them (independently of the model) to the user. These outliers typically indicate that the sensor is misbehaving or that something interesting has happened (*e.g.*, the sensor became hot because a fire started nearby) – both cases of which the user should be aware.

Autonomy, redundancy, and compression. After distributed regression, each node has the coefficients of the kernels to which it belongs; this knowledge is not (necessarily) centralized. This allows the user to connect to any node and quickly have questions answered about the sensor field around that point. For a small cost, the regression coefficients of the entire network can be collected at that point, allowing any query to answered from any point in the network. Furthermore, because the coefficients of each kernel are distributed to all of the nodes in the kernel, the failure of one or more nodes does not affect the ability of the network to answer these queries. Finally, regression coefficients provide a compact summary of readings at a given point in time; by storing historical regression coefficients, the network can answer queries about the distant past. These properties compare well with other in-network storage approaches(RKY+02), which store all data (limiting the extent of history that can be realistically kept) and do not provide redundancy without the addition of explicit and difficult to implement replication.

Adaptive data modeling. We have assumed that the location of kernels and the set of basis functions are given by the user. Figure 10 suggests that, when the temperature is changing rapidly, we may need more basis functions, while at night, a simple model is sufficient to represent the data. Once the local coefficients are obtained, the nodes in a region may be able use this model locally to estimate the advantage of adding or removing basis functions or kernels. Such procedure could allow us to adapt the model (and the required communication) as environmental conditions change.

References

[BB72] U. Bertele and F. Brioschi. *Nonserial Dynamic Programming*. Academic Press, New York, 1972.

[CDLS99] R. Cowell, P. Dawid, S. Lauritzen, and D. Spiegelhalter. *Probabilistic Networks and Expert Systems*. Spinger, 1999.

[GV89] G. Golub and C. Van Loan. *Matrix Computations*. Johns Hopkins, 1989.

[HSW+00] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. Culler, and K. Pister. System architecture directions for networked sensors. In *Proceedings of ASPLOS*, pages 93–104, November 2000.

[IEGH02] C. Intanagonwiwat, D. Estrin, R. Govindan, and J.Heidemann. Impact of network density on data aggregation in wireless sensor networks. In *ICDCS*, July 2002.

[IGE00] C. Intanagonwiwat, R. Govindan, and D. Estrin. Directed diffusion: A scalable and robust communication paradigm for sensor networks. In *MobiCom*, August 2000.

[LLW+03] P. Levis, N. Lee, A. Woo, M. Welsh, and D. Culler. TOSSIM: Accurate and scalable simulation of entire tinyos applications. In *SenSys*, November 2003.

[Mad03] S. Madden. *The Design and Evaluation of a Query Processing Architecture for Sensor Networks*. PhD thesis, UC Berkeley, 2003.

[Mic03] M. Hamilton et al. Habitat sensing array, first year report. http://cens.ucla.edu/Research/Applications/habitat_sensing.htm, 2003.

[MN83] P. McCullagh and J. Nelder. *Generalized linear models*. Chapman and Hall, 1983.

[MPS+02] A. Mainwaring, J. Polastre, R. Szewczyk, D. Culler, and J. Anderson. Wireless sensor networks for habitat monitoring. Technical Report IRB-TR-02-006, Intel Research, June 2002.

[NM03] R. Nowak and U. Mitra. Boundary estimation in sensor networks: Theory and methods. In *IPSN*, 2003.

[PG03] M. Paskin and C. Guestrin. Distributed inference in sensor networks. Technical Report IRB-TR-03-039, Intel Research, 2003.

[PL03] M. Paskin and G. Lawrence. Junction tree algorithms for solving sparse linear systems. Technical Report UCB/CSD-03-1271, U.C. Berkeley, 2003.

[PK00] G. Pottie and W. Kaiser. Wireless integrated network sensors. *Communications of the ACM*, 43(5):51 – 58, May 2000.

[PMBT01] N. Priyantha, A. Miu, H. Balakrishnan, and S. Teller. The Cricket Compass for context-aware mobile applications. In *Proceedings of ACM MOBICOM*, Rome, Italy, July 2001.

[RKY+02] S. Ratnasamy, B. Karp, L. Yin, F. Yu, D. Estrin, R. Govindan, and S. Shenker. Ght: A geographic hash table for data-centric storage in sensornets, 2002.

[YG03] Y. Yao and J. Gehrke. Query processing in sensor networks. In *CIDR*, 2003.