Robust Distributed Estimation in Sensor Networks using the Embedded Polygons Algorithm

Véronique Delouille
ECE Department
Rice University
Houston, Texas, USA
vero@rice.edu

Ramesh Neelamani
Upstream Research Company
ExxonMobil
Houston, TX, USA
ramesh.neelamani@exxonmobil.com

Richard Baraniuk *
ECE Department
Rice University
Houston, Texas, USA
richb@rice.edu

ABSTRACT

We propose a new iterative distributed algorithm for linear minimum mean-squared-error (LMMSE) estimation in sensor networks whose measurements follow a Gaussian hidden Markov graphical model with cycles. The *embedded polygons algorithm* decomposes a loopy graphical model into a number of linked embedded polygons and then applies a parallel block Gauss-Seidel iteration comprising local LMMSE estimation on each polygon (involving inversion of a small matrix) followed by an information exchange between neighboring nodes and polygons. The algorithm is robust to temporary communication faults such as link failures and sleeping nodes and enjoys guaranteed convergence under mild conditions. A simulation study indicates that energy consumption for iterative estimation increases substantially as more links fail or nodes sleep. Thus, somewhat surprisingly, energy conservation strategies such as low-powered transmission and aggressive sleep schedules could actually be counterproductive.

Categories and Subject Descriptors

G.3 [**Probability and Statistics**]: Statistical computing; J.2 [**Physical Sciences and Engineering**]: Engineering

General Terms

Algorithms, Reliability, Theory

Keywords

Sensor networks, distributed estimation, graphical models, hidden Markov models, Wiener filter, matrix splitting

*This work was supported by grants from NSF, DARPA, ONR, AFOSR, and the Texas Instruments Leadership University Program. Web: dsp.rice.edu.

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

1. INTRODUCTION

Sensors, signal processing, and wireless communication technologies have matured to the point where large networks of sensor nodes can now be easily deployed in a wide variety of environments, making them very attractive for large-scale applications like environmental monitoring, security surveillance, and disaster relief [1]. Often battery-powered, sensor nodes are capable of *sensing*, *computing*, and *communicating* information.

In the setting considered here, we assume that N sensors each make noisy scalar measurements of their physical environment, such as the local temperature, wind speed, or concentration of some substance. We model the vector of N measurements y as a Gaussian random field $y = x + \epsilon$, with

$$y_i = x_i + \epsilon_i, \qquad i = 1, \ldots, N. \tag{1}$$

Here x_i is the true value of the random field at sensor i, and ϵ_i is sensor i's measurement noise. We model the vector x as jointly Gaussian with zero mean and correlation matrix Σ, and we assume that the measurement noise ϵ is independent of x and Gaussian with zero mean and diagonal correlation matrix R. Since the variables x_i are Gaussian and unknown (that is, hidden), this model is often referred to as a Gaussian *Hidden Markov Model* (HMM).

A core sensor network problem involves producing accurate estimates of the true x_i's from the noisy y_i's. The linear minimum mean-squared-error (LMMSE) Wiener estimate of x is computed from the *normal equations*

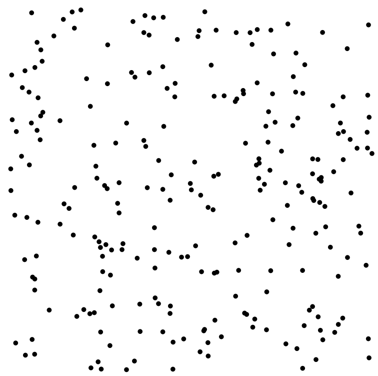
$$V\widehat{x} = R^{-1}y \tag{2}$$

with $V := \Sigma^{-1} + R^{-1}$.

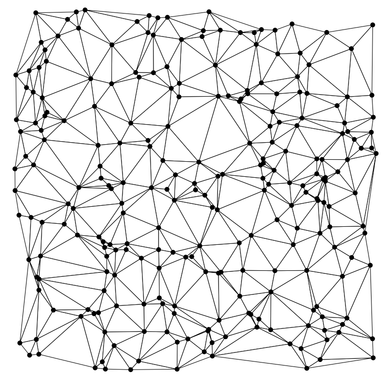
In a sensor network, a naive approach to obtain the LMMSE estimate would first transmit all of the measurements y to a central location and then compute and apply the $N \times N$ matrix inverse V^{-1} to $R^{-1}y$ to obtain $\widehat{x}$. Unfortunately, since communication consumes significant energy and bandwidth [2], this naive centralized approach is extremely wasteful.

Thus, there is a great need for *distributed* estimation algorithms that replace global communication and centralized computation by parallel, *local* communication and computation, effectively distributing the $N \times N$ matrix inverse computation across the network. Such distributed algorithms should also be resilient to communication failures caused by sleeping sensors and faulty transmissions.

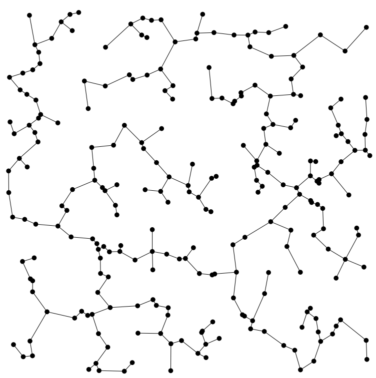
Graphical models (GMs) provide a natural foundation both for modeling the correlations amongst the sensors (the



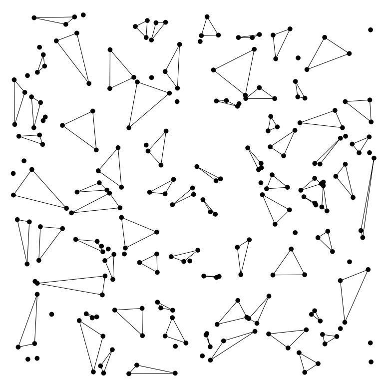
(a) Sensor nodes



(b) Delaunay triangulated network



(c) Minimum spanning tree



(d) Independent embedded triangles

Figure 1: (a) Example sensor network with 250 uniformly distributed sensor nodes. (b) Graphical model for the hidden variables x_i based on the Delaunay triangulation of the sensor locations. (c) Minimum spanning tree embedded in the Delaunay graph. (d) A set of independent triangles and singleton nodes embedded in the Delaunay triangulation. Independence refers to the fact that no two triangles share a common node.

matrix Σ) and for developing efficient distributed estimation algorithms [3]. In a sensor network GM, the nodes correspond to the variables x_i, and the existence of an edge between nodes x_i and x_j indicates a conditional dependency or nonzero *partial correlation* $P_{i,j}$. The correlation matrix is given by $\Sigma = P^{-1}$ [3].

The sparsity of the matrix P controls the efficiency of GM distributed algorithms. In a fully connected GM (which can exactly model any Σ), distributed algorithms involve intensive computation, communication, and energy consumption. Fortunately, for smoothly varying physical quantities such as temperature or gas concentration, sensor i's measurement can be assumed to be uncorrelated with the rest of the network given the measurements of all close-by sensors. In the associated GM, each node x_i would be connected only to its close spatial neighbors,[1] leading to a sparse P and efficient algorithms. For example, Fig. 1(b) illustrates a GM based on a *Delaunay triangulation* [4] of the node locations from Fig. 1(a).[2]

[1]For example, a neighborhood can be defined as all nodes within a certain radius.

[2]A Delaunay triangulation has the attractive property of linking together the closest nodes.

Sensor network GMs in most practical situations will contain numerous *loops* or *cycles* (see Fig. 1(b), for example). Unfortunately, while there exist simple and efficient distributed estimation algorithms for GMs without loops (Markov chains, for example), the situation is anything but straightforward when loops are present. Proposed distributed estimation approaches for loopy GMs include the non-iterative Gaussian elimination message passing [5] as well as iterative methods such as belief propagation (BP) [6], loopy BP [7], and embedded trees [8, 9]. Each of these algorithms either lacks fault tolerance, lacks parallelism, or converges slowly, which limits their applicability to wireless sensor networks.

In this paper, we develop the *embedded polygons algorithm* (EPA), a new iterative algorithm for distributed LMMSE estimation in GMs that is simple, local, scalable, fault-tolerant, and energy-efficient, and thus well-suited for wireless sensor networks [10].

Given a loopy sensor network GM, our core idea is to first decompose it in terms of a number of linked *embedded polygons*; see Fig. 1(d) for an example of embedded *triangles.* Then, we compute the solution to (2) via an iteration comprising local LMMSE estimation on each polygon (for example, a 3×3 matrix inverse computation for an embedded triangle) followed by an exchange of this information between neighboring nodes and polygons.

Paper overview: After defining GMs and our estimation problem more precisely in Section 2, we review matrix splitting algorithms in Section 3 and the EPA in Section 4. Section 5 overviews the EPA's fault tolerance and convergence properties, while Section 6 compares its power consumption as a function of communication failure rate to the Jacobi algorithm in two simulation experiments. A central contribution is a guarantee that both the EPA and Jacobi algorithm converge when sensors and communication links fail temporarily. Moreover, we demonstrate that networks with nodes that sleep (ostensibly to save energy) can actually consume *more* total energy to converge to the LMMSE estimate due to many additional iterations. We conclude with a discussion and pointers to future work in Section 7.

2. ESTIMATION IN SENSOR NETWORKS

Our aim is to find the LMMSE estimate for the true sensor values x_i from the noisy measurements y_i under the model (1). To do so efficiently, we approximate the correlation structure of x by a *graphical model* (GM). We will use the terms "sensor" and "node" interchangeably below.

2.1 Graphical models

GMs, including Bayesian networks and Markov random fields, represent statistical dependencies between variables by means of a graph [3]. Let $\mathcal{G}$ be an undirected graph defined by a set of nodes $\mathcal{V}$ and a set of edges $\mathcal{E}$. Each node $i \in \mathcal{V}$ is associated with an unobserved or *hidden variable* x_i as well as with a noisy *measurement* y_i. We assume that, given the set of hidden variables $x := \{x_i | i \in \mathcal{V}\}$, the observations $y := \{y_i | i \in \mathcal{V}\}$ are independent of each other. A missing edge (i,j) between nodes i and j implies conditional independence between the variables x_i and x_j given all other hidden variables. That is, if the *neighborhood* of node i is defined as $\mathcal{N}(i) := \{j | (i,j) \in \mathcal{E}\}$, then $p(x_i | x_{\mathcal{V}\setminus i}) = p(x_i | x_{\mathcal{N}(i)})$.

In this paper, we focus on Gaussian HMMs, where the hidden variables x form a jointly Gaussian process. In this case, note that the matrix $P = \Sigma^{-1}$ is related to the *partial correlation* $r_{i,j|\mathcal{V}\setminus\{i,j\}}$ between x_i and x_j [3] by $r_{i,j|\mathcal{V}\setminus\{i,j\}} = -P_{i,j}/\sqrt{P_{i,i}P_{j,j}}$. Note that even when P is sparse, $\Sigma = P^{-1}$ will generally be full; that is, all nodes are correlated with all others. In particular, if x_i and x_j are partially correlated and so are x_j and x_k, then x_i and x_k are correlated through x_j. x_i and x_k are independent only when the value of x_j is given.

2.2 Choice of graphical model

In this paper, we assume that the matrices Σ (equivalently P) and R are given. In practice, however, Σ and R are unknown and must be estimated from the data. The matrix R represents the variance of the noise and can be estimated from repeated measurements at the same location. The matrix $\Sigma_y = \Sigma + R$ can be estimated from the sample covariance matrix of the observations. For *decomposable* graphical models [11], Lauritzen [3] gives an algorithm that computes the maximum likelihood estimate of $V = \Sigma^{-1} + R^{-1}$ from inverses of sub-matrices of the estimate $\widehat{\Sigma}_y$.

In sensor network estimation applications, GMs should balance the trade-off between accurately capturing the correlation structure of the quantities being measured and supporting energy-efficient distributed algorithms. A spatial *triangulation* of the sensor location induces a GM that balances this trade-off. A triangulated GM assumes that a sensor's measurement is uncorrelated with the rest of the network given the close-by measurements. This is clearly reasonable for smoothly varying quantities.

The *Delaunay triangulation* (DT) [4] (see Fig. 1(b)) induces a GM with some additional attractive properties. First, the DT links together the *closest* neighbors in the graph, in the sense that the circumcircle of each triangle[3] does not contain any points of the triangulation. Second, the DT can be established in a distributed fashion [12, 13], and for this reason is successfully used as an overlay topology in the networking field [12]. For the non-zero partial correlation between two connected sensors, we use a decreasing function of the Euclidean distance between them [14, p. 61].

3. DISTRIBUTED ESTIMATION

We can take advantage of a sensor network's computation and communication capabilities to solve the linear system (2) in a distributed way. This saves the significant cost of transmitting all data to a central location.

A number of algorithms that proceed by passing messages along the edges of a graphical model have been proposed to solve (2). These include belief propagation (BP) [6], loopy BP [7], and extended message passing based on Gaussian elimination [5]. In order to efficiently handle loopy GMs, we consider in this paper an attractive class of iterative methods for solving (2) based on a decomposition of V.

3.1 Matrix splitting algorithms

Let $\overline{y} = R^{-1}y$ denote the normalized observation vector. Equation (2) then becomes

$$V\widehat{x} = \overline{y}. \tag{3}$$

If we rewrite V by "splitting" it into

$$V = J - K, \tag{4}$$

then solving (3) is equivalent to finding a fixed point of the system [15]

$$J\widehat{x} = \overline{y} + K\widehat{x}.$$

For sensor networks, we seek a (J, K) pair for which the solution to this fixed-point problem can be found iteratively by simple computations and information exchanges between neighboring nodes.

We assume that at the start of the algorithm each sensor $i \in \mathcal{G}$ knows its initial estimate $\widehat{x}_i^0$ as well as the partial correlations with its neighbors; that is, it knows row i of the matrix V. Let $\mathcal{N}_K(i) := \{j \in \mathcal{N}(i) | K_{i,j} \neq 0\}$ be the neighbors of i that correspond to non-zero entries of the matrix K in (4).

Starting from the initial guess $\widehat{x}^0$, we solve the fixed point equation by generating a sequence of iterates $\{\widehat{x}^m\}_{m=1}^{\infty}$ according to the following two-step recursion:

Update: Each node i sends its value x_i^{m-1} to its neighbors $\mathcal{N}_K(i)$ and receives from them their current estimates $\widehat{x}^{m-1}_{\mathcal{N}_K(i)}$. The nodes then update their values using

$$\widehat{y}^m = \overline{y} + K\widehat{x}^{m-1}. \tag{5}$$

Solve: The new estimate $\widehat{x}^m$ is found from

$$\widehat{x}^m = J^{-1}\widehat{y}^m. \tag{6}$$

[3]The circle passing through the three vertices of the triangle.

This algorithm converges when the spectral radius of the matrix $J^{-1}K$ is strictly smaller than one [15]; that is

$$\widehat{x}^m \overset{m\to\infty}{\longrightarrow} \widehat{x} \qquad \Leftrightarrow \qquad \rho(J^{-1}K) < 1.$$

Several choices of (J, K) are appropriate for distributed estimation.

3.2 Jacobi algorithm

The simplest splitting algorithm is the well-known *Jacobi iteration* that sets J equal to the diagonal of V

$$J = \begin{bmatrix} V_{1,1} & 0 & 0 \\ 0 & \ddots & 0 \\ 0 & 0 & V_{N,N} \end{bmatrix}, \tag{7}$$

where $V_{kk}, k = 1, \ldots, N$ denote the diagonal elements of V. Iteration m proceeds as follows:

Jacobi Update: Node i sends its current estimate $\widehat{x}_i^{m-1}$ to all of its neighbors $j \in \mathcal{N}(i)$ and receives the current estimates $\widehat{x}_j^{m-1}$ from those same neighbors. The nodes then compute

$$\widehat{y}_i = \overline{y}_i - \sum_{j\in\mathcal{N}(i)} V_{i,j} x_j^{m-1} \qquad \forall i \in \mathcal{G}.$$

Jacobi Solve: $\widehat{x}_i^m = V_{i,i}^{-1}\widehat{y}_i \qquad \forall i \in \mathcal{G}.$

The Jacobi algorithm converges slowly in general, but it has the advantage of being eminently *local*: each node i needs to know only its neighbors' values $x_{\mathcal{N}(i)}$ in order to proceed.

3.3 Embedded trees algorithm

In the *embedded trees algorithm*, the matrix $J := J_T$ corresponds to a spanning tree $\mathcal{G}_T$ embedded in the loopy graph $\mathcal{G}$ [8, 9]. For example, Fig. 1(c) illustrates a minimum-spanning tree for the sensor network in Fig. 1(b). With embedded trees, the inversion of J_T can be performed exactly using a message passing algorithm such as belief propagation. Conditions for convergence are provided in [9]. Iteration over multiple different trees is also possible and in certain case significantly improves the rate of convergence [9]. The method can further be extended to exactly compute the variance of $\widehat{x}$. Unlike the Jacobi method, the embedded trees algorithm is *global*; in the Solve step, messages must be passed throughout the entire network before node i can compute the next value of $\widehat{x}_i^m$.

4. EMBEDDED POLYGONS ALGORITHM

We now present an algorithm that combines the best of the Jacobi and embedded tree methods in the sense that it is both *local* and *fast* to converge.

4.1 Embedded polygons

When a graphical model contains many loops, it becomes efficient to place in J the elements of V corresponding to small connected subsets of nodes from $\mathcal{G}$. Hence, we propose to place in J a set of subgraphs of $\mathcal{G}$ that we term *independent embedded polygons*. As shown in Fig. 2, examples of embedded polygons include singletons, dipoles, triangles, quadrangles, and so on. Two polygons are independent if they have no common vertices. Let $\tau := \{\tau_k\}_{k=1}^M$ be a set of independent polygons of size L_k embedded in a loopy graph

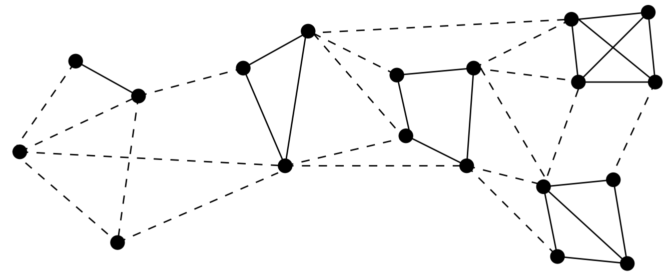


Figure 2: A graphical model (solid and dashed lines) and a decomposition into embedded polygons (solid lines); from left to right two singletons, a dipole, triangle, quadrangle, chordal quadrangle, and fully connected quadrangle.

$\mathcal{G}$, and let A_{τ_k} denote the $L_k \times L_k$ sub-matrix of V corresponding to the polygon τ_k. For an appropriate ordering of the nodes, J becomes the block diagonal matrix

$$J = \begin{bmatrix} A_{\tau_1} & 0 & 0 \\ 0 & \ddots & 0 \\ 0 & 0 & A_{\tau_M} \end{bmatrix}. \tag{8}$$

We refer to the resulting matrix splitting method as the *embedded polygons algorithm* (EPA). The Jacobi matrix J in (7) is a special case of (8) where the sub-matrices A_{τ_k} degenerate to the diagonal elements V_{kk} of V (that is, all polygons are singletons).

4.2 Parallel block Gauss-Seidel algorithm

With J defined as in (8), the recursion (5), (6) corresponds to a *block-Jacobi iteration*. To maximize parallelism in the EPA, we slightly modify iteration m to proceed as follows:

EPA Update: Each node i computes the update $\widehat{y}_i^m$ as in (5) by communicating with all of its neighbors except the ones belonging to the same embedded polygon.

EPA Solve: Within each embedded polygon τ_k, the nodes $i \in \tau_k$ exchange their updated values $\widehat{y}_i^m$ amongst themselves. Each polygon then computes $\widehat{x}_{\tau_k}^m = A_{\tau_k}^{-1}\widehat{y}_{\tau_k}^m$ in parallel. (Note that singleton nodes do not communicate in this step.)

The EPA is simple to parallelize because the embedded polygons are independent, that is, they have no nodes in common. The inversion of each $L_k \times L_k$ matrix A_{τ_k} in the EPA Solve step above is performed non-iteratively. In the initialization phase, each node belonging to τ_k gathers information about A_{τ_k} from its neighbors $i \in \tau_k$ and computes and stores the inverse $A_{\tau_k}^{-1}$.

In order to increase the rate of convergence, we can also slightly reduce the parallelism and introduce a Gauss-Seidel [15] aspect to the algorithm, which now proceeds in two stages. In the first stage, all nodes belonging to embedded polygons of size $L_k \geq 2$ perform the EPA Update and EPA Solve steps to obtain their new value $\widehat{x}^m$ at iteration m. In the second stage, each singleton node i gathers the most recent estimates from its neighbors $j \in \mathcal{N}(i)$. Node i then uses these values to compute the update $\widehat{y}_i^m$ and to solve $\widehat{x}_i^m = \widehat{y}_i^m / V_{i,i}$.

4.3 Size of embedded polygons

The choice of the sizes of the embedded polygons τ_k is application-dependent. If the size $L_k > 5$, then an iterative method is needed to invert the $L_k \times L_k$ matrix during the EPA Solve step. However, larger blocks A_{τ_k} also increase the algorithm's rate of convergence [15, p. 194]. Hence a trade-off of computation/communication versus faster convergence will dictate the optimal polygon size L_k.

Since any planar graph can be decomposed into a set of embedded triangles, dipoles, and singletons, in the simulations of Section 6 we focus on triangulated GMs and study the performance of the resulting *embedded triangles algorithm* (ETA) [10].

4.4 Scalability and self-organization

Since the EPA operates in parallel over all independent polygons, the amount of time spent on each iteration does not change as we increase the number of nodes N in the network. In contrast, since the embedded trees [9], Gaussian elimination [5], and loopy BP [7] algorithms rely on sequential message passing either on a tree (for the first two methods) or on the entire graph (for loopy BP), their time-per-iteration will increase with the diameter of the network,[4] which grows with N [16].

Moreover, the EPA and ETA can begin iterating without expensive up-front route finding. Indeed, simple protocols exist to associate the sensors into independent triangles, for example. Suppose that in a triangulated GM each sensor knows the locations of its neighbors. At initialization, all nodes are singletons and free to form a triangle. A *leader* node ℓ is designated to perform the following operations:

Query to form a triangle: The leader ℓ sends a message to its neighbors to request permission to form a triangle τ. The neighbors respond *yes* if they are singletons and *no* if they are already part of a triangle.

Triangle formation: If ℓ receives two positive answers that allow it to form a triangle τ, then it sends back a message to those two nodes that become part of the triangle τ. If ℓ cannot form a new triangle, then the algorithm terminates.

Leadership hand-off: Node ℓ looks for singleton nodes in the neighborhood of its triangle. If there are no singletons, then the algorithm terminates. Otherwise, it hands off leadership to a randomly chosen neighboring singleton.

This algorithm results in an expanding wavefront of crystalizing independent embedded triangles.

5. FAULT-TOLERANCE

In this section, we first discuss how the Jacobi algorithm and the EPA adapt to temporary inter-node communication failures in the network. Then we prove that both the algorithms are guaranteed to converge under mild assumptions.

[4]The diameter of a graph is the length of the longest shortest path between two nodes.

5.1 Sleeping nodes and failing links

In practical sensor networks, inter-node communication can fail due to two main causes. Define a *sleeping node* as one that powers-down temporarily to conserve energy and hence cannot interact with its neighbors [17, 18]. A *temporary link failure* occurs when the transmitted information is dropped due to temporary inter-node channel fluctuations (for example, due to multi-path fading).

To increase the Jacobi and EPA's robustness to temporary link failures and sleeping nodes, we modify them slightly by allowing each sensor node to store its neighbors' most recent, successfully communicated values in its *local memory.* Let I^m be the set of active nodes at iteration m. A sleeping node $i \notin I^m$ updates neither its estimate nor its memory during iteration m (that is, $\widehat{x}_i^m := \widehat{x}_i^{m-1}$) and, moreover, does not communicate with its neighbors. A failing communication link between sensors j and i precludes node i from receiving any data from node j. In place of this missing value, sensor i recalls from its local memory the last successfully received value from sensor j. Let $s_{j\to i}(m)$ denote the iteration count corresponding to this most recently received value. In Robust EPA, active nodes $i \in I^m$ perform the following steps. The Robust Jacobi algorithm is defined similarly, since Jacobi is a particular case of EPA with singleton polygons.

Robust EPA Update: For all $j \in \mathcal{N}_K(i)$, sensor i either receives the value $\widehat{x}_j^{m-1}$ from sensor j (successful communication) or recalls from its memory the value most recently received from j (failed communication). Then $\widehat{y}_i^m$ is updated as in (5) with $\widehat{x}_j^{m-1}$ or $\widehat{x}_j^{s_{j\to i}(m)}$ for $j \in \mathcal{N}_K(i)$. In other words, sensors substitute values from memory for the values not received and perform the usual update (5).

Robust EPA Solve: If sensor i belongs to an independent polygon τ_k, then it sends $\widehat{y}_i^m$ to its neighbors in τ_k. If sensor i does not receive the value $\widehat{y}_j^m$ from one of its neighbors $j \in \tau_k$, then it recalls from its memory the last successfully received value $\widehat{y}_j^{s_{j\to i}(m)}$. Sensor i solves for $\widehat{x}_i^m$ by inverting the matrix A_{τ_k} as in the EPA Solve step from Section 4.2. If node i is a singleton, then it solves according to the Jacobi Solve step from Section 3.2. (Note that the singleton nodes do not communicate in this step.)

Unlike the Jacobi algorithm, EPA, and ETA, it is unclear how to make the embedded trees algorithm resilient to communication failures. Embedded trees employ belief-propagation (sum-product algorithm) in the Solve step. During this step, messages must be transmitted successfully through the entire tree to compute the new $\widehat{x}^m$. If a link (i, j) belonging to the tree fails, then it is unclear how to proceed with the iteration until the tree is healed (perhaps by some higher-level networking protocol).

5.2 Robust convergence

Mild conditions on the occurrence of link failures and sleeping nodes are sufficient to ensure convergence of both the Robust Jacobi algorithm and Robust EPA: As the iterations proceed, all nodes must eventually receive updated values from their neighbors, and each node must eventually be updated. In other words, links can fail and nodes can sleep, but only temporarily.

THEOREM 1. *Consider the matrix V in (2) corresponding to a Gaussian HMM with all elements of Σ positive. If during the Robust EPA Update and Robust EPA Solve steps of Section 5.1 the delays $d_{j\to i}(m) := m - s_{j\to i}(m)$ are bounded for all $(i,j) \in \mathcal{G}$ and if each node is updated within a finite number of iterations, then both the Robust Jacobi algorithm and the Robust EPA converge.*

For the proof, see the Appendix.

6. SIMULATIONS

To complete the paper, we now conduct several simulations to study the rates of convergence of the *Jacobi* and *embedded triangles algorithm* (ETA) estimates as a function of energy consumption for both unidirectional and omnidirectional communication.

6.1 Data generation

We assume that 250 sensor nodes are randomly distributed in the square $[0,1] \times [0,1]$. We generate observations $y = x + \epsilon$ with $\epsilon \sim N(0, \sigma^2 I)$, variance $\sigma^2 = 4$, and I the identity matrix. To generate the hidden variables x, we set up the partial correlation coefficients $r_{i,j|\mathcal{V}\backslash\{i,j\}} = c(1 - \mathrm{dist}(i,j))$, with $\mathrm{dist}(i,j)$ the Euclidean distance between nodes i and j and c the largest constant for which the precision matrix P is still positive definite. For the diagonal entries of P we use random variables uniformly distributed between 1 and 2. This permits us to compute the values $P_{i,j}; i \neq j$ as described in Section 2.2 and to generate $x \sim N(0, P^{-1})$. This specification of P ensures that the elements of $\Sigma = P^{-1}$ are positive (see the Appendix). We assume that $V = \Sigma^{-1} + R^{-1} = \Sigma^{-1} + \sigma^{-2} I$ (see (6)) is known. Note that if V were estimated from sample data (see Section 2.2), then the solution $\widehat{x}$ of (2) would contain some bias due to the error in estimating V. Our discussions below address only the error due to the iterative nature of the matrix splitting algorithm (5), (6).

6.2 Comparison metrics

To evaluate the different algorithms, we compare the decay of their *residual normalized mean-squared error* (termed residual error henceforth) as the total energy consumed by inter-node communication increases.

The residual error is computed at each iteration m as $e^m := \|\overline{y} - V\widehat{x}^m\| / \|\overline{y}\|$ [9] with $\overline{y} = y/\sigma^2$ and $V = \Sigma^{-1} + \sigma^{-2} I$. In our experiments, we compute the inter-node communication energy for the two different scenarios.

With *unidirectional transmitters*, a node communicates with each of its neighbors one at a time, even to convey a common message. Examples are nodes with laser-based line-of-sight transmitters. We assume that the energy required by sensor i to convey the same floating point number to all of its neighbors $j \in \mathcal{N}(i)$ is proportional to [2]

$$\sum_{j\in\mathcal{N}(i)} (\mathrm{dist}(i,j))^2 .$$

With *omnidirectional transmitters,* a node can broadcast a common message to all of its neighbors simultaneously. Examples are nodes with omnidirectional radio transmitters. We assume that the energy required by sensor i to broadcast the same floating point number to all of its neighbors $j \in \mathcal{N}(i)$ is proportional to [2]

$$\max_{j\in\mathcal{N}(i)} (\mathrm{dist}(i,j))^2 .$$

6.3 Fault-free case

We first compare the experimental performance of the Jacobi algorithm, embedded trees algorithm, and the ETA when all nodes and links stay awake throughout the estimation. We averaged the residual error over 50 different realizations of the noise ϵ (see (1)).

Figure 3(a) illustrates the superior performance of the ETA when the sensors are equipped with unidirectional transmitters; its residual error decays faster with increasing energy consumption. With omnidirectional communication, according to Fig. 3(b), ETA's improvement is less substantial.

6.4 Fault-prone case

We now investigate the robustness of the Jacobi algorithm and the Robust ETA to sleeping nodes and communication failures. Since it is unclear how the modify the embedded trees algorithm to handle such faults, we exclude it from this comparison.

In practical scenarios, sensors might need to sleep in order to conserve energy or recharge their batteries [17, 18]. We model this scenario by assuming that, at every iteration, each node independently toggles between the "awake" and one of q "sleep" states according to the finite state model of Fig. 4. According to the model, if a node is currently awake, then at the next iteration it either goes to the state sleep$_1$ with probability p_1 or continues staying awake. If a node has slept for k consecutive cycles, $k < q$, then at the next iteration it can either continue to sleep with probability p_2 by going to state sleep$_{k+1}$ or wake up. A node can sleep for at most q iterations. In the steady state, a node that conforms to such a finite state model will on average stay awake with probability $\frac{1-p_2}{1-p_2+p_1\left(1-p_2^q\right)}$.

To model temporary inter-node communication link failures due to, say, fading, we assume that the each link toggles between the "no-failure" and "failure" states of an analogous finite state model.

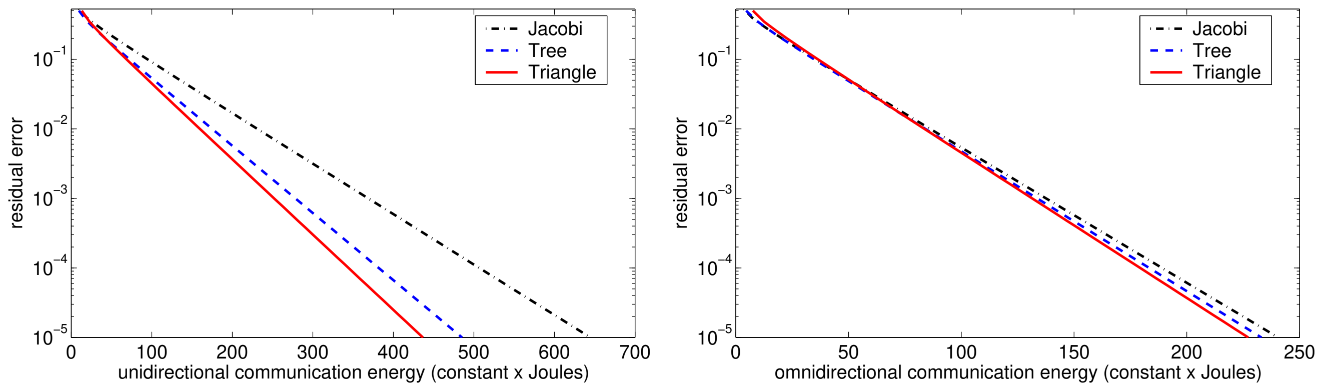
In case of definitive link failures or dead sensors, the GM must be updated. Methods to handle such faults, including imputation for missing data [11, p. 200], are beyond the scope of this paper.

Figures 5(a) and (b) present the Jacobi algorithm and the ETA residual error decay versus increasing unidirectional and omnidirectional communication energy when $p_1 = 0$, 0.1, 0.3 for both sleeping and failing; we set $p_2 = 0.5$ and $q = 10$. These parameters for p_1 translate to each node or link staying awake for 100%, 83%, and 63% of the iterations on average, respectively. We averaged the residual error over 50 realizations, each with different noise and different sequences of link failures and sleeping nodes.

From Figures 5(a) and (b), we can infer that both the Jacobi algorithm and the ETA are robust since they continue to converge.

Note that a sleeping node i can be modeled with failed links between i and all of its neighbors $\mathcal{N}(i)$. Hence in the sequel we will focus our discussion on failing links.

In the unidirectional case, the ETA enjoys faster error decay than the Jacobi algorithm for all p_1's. To attain a residual error of 10^{-5}, the ETA takes twice as much energy



(a) Residual error versus *unidirectional* communication energy

(b) Residual error versus *omnidirectional* communication energy

Figure 3: Simulation results for distributed estimation with the sensor network from Fig. 1(b) using the Jacobi, embedded trees, and embedded triangles algorithms. In this case, no nodes sleep and no communication links fail.

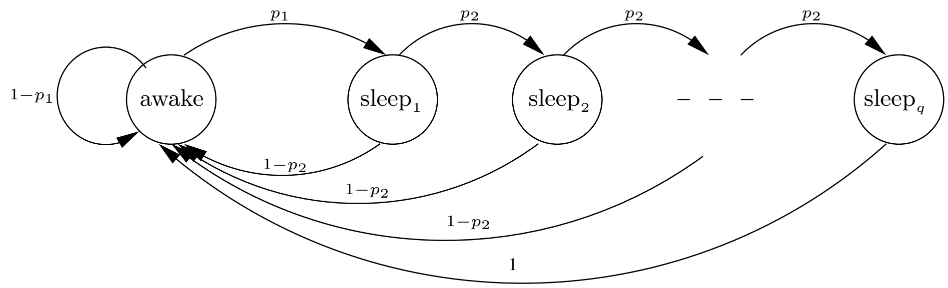


Figure 4: Finite state model for sleeping nodes.

when $p_1 = 0.3$ as compared to when $p_1 = 0$. As p_1 increases, the performance differences between the two algorithms becomes negligible.

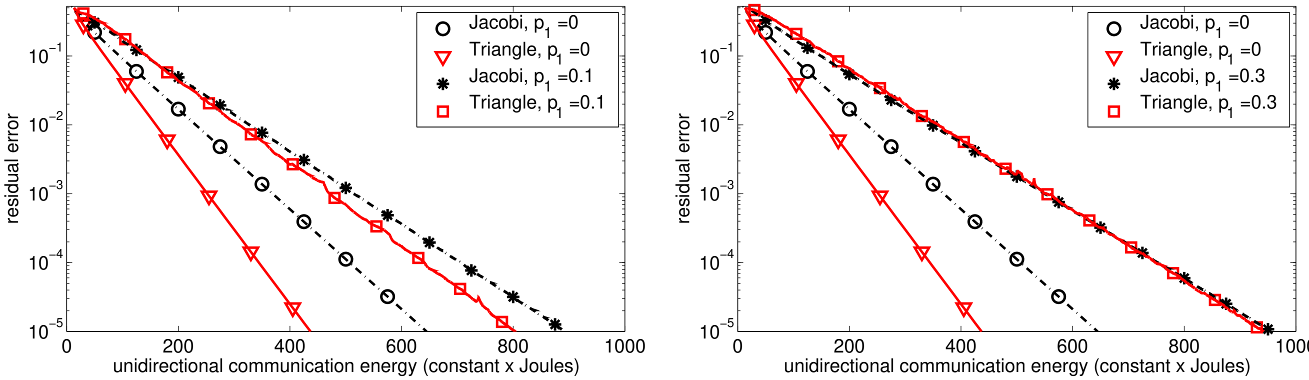
The reason is that the Robust ETA requires inter-node communications for *both* the Robust Update step and the Robust Solve step. In contrast, Jacobi communicates only during the Update step. To illustrate the implications, consider the situation for sensor i and its neighbors j and k at iteration m when the link from j to k is wonky. Suppose that j and k belong to the same embedded triangle δ, but i does not. In both the Jacobi and ETA Update steps, the value $\widehat{x}_i^{m-1}$ is used to compute $\widehat{y}_j^m$. In the Jacobi Solve step, $\widehat{y}_j^m$ is used to compute $\widehat{x}_j^m$, and no communication is required. In contrast, in the Robust ETA Solve step, $\widehat{y}_j^m$ must be transmitted to node k. If the transmission from j to k fails, then node k will compute its new value $\widehat{x}_k^m$ using the last successfully received value from j, denoted $\widehat{y}_j^{s_{j\to k}(m)}$. Obviously, $\widehat{y}_j^{s_{j\to k}(m)}$ does not contain the most recent information transmitted by $\widehat{x}_i^{m-1}$, even though this value was successfully transmitted in the ETA Update step from i to j. Thus the ETA's performance will degrade relative to the Jacobi algorithm as more nodes sleep or links fail.

In the omnidirectional case, the Jacobi algorithm is well-suited to exploit the transmitters since during each Jacobi iteration, a common message must be broadcast from each node to all of its neighbors. In contrast, the ETA expends energy communicating two different messages per sensor belonging to an embedded triangle at each iteration m: the value $\widehat{x}_i^{m-1}$ in the Update step and the value $\widehat{y}_j^m$ in the Solve step. Consequently, Jacobi soon outperforms the ETA as p_1 increases from zero.

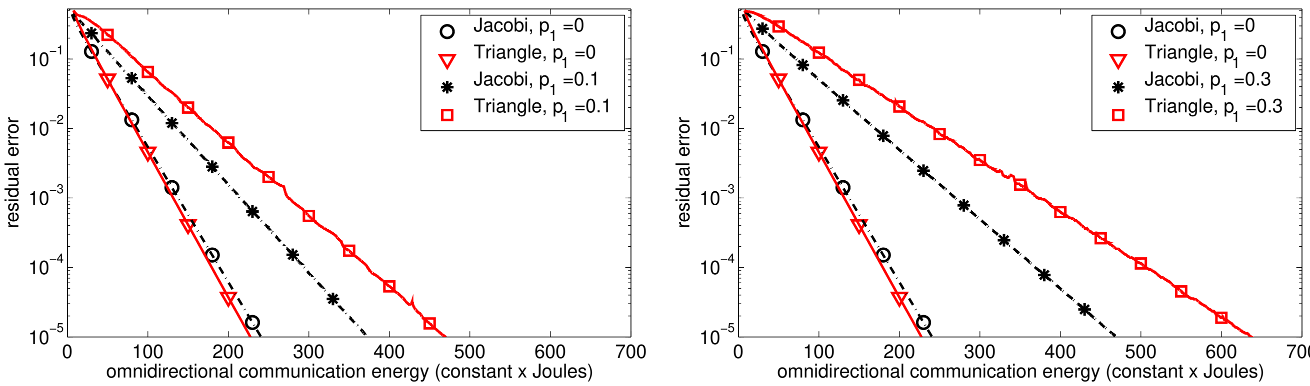
7. CONCLUSIONS

The embedded polygons algorithm (EPA) (and the special-case embedded triangles algorithm (ETA)) are new distributed estimation schemes for loopy Gaussian hidden Markov graphical models. Our iterative approach decomposes a GM into independent embedded polygons, performs LMMSE estimation on each polygon, and then updates this estimate by collaborating with neighboring nodes and polygons. The EPA can be interpreted as an extension of the block Gauss-Seidel approach to matrix inversion. The EPA is robust to temporary local faults such as sleeping nodes and failing communication links.

Our experimental results provide some interesting insights into the tradeoff between a sensor network's energy consumption and iterative algorithm convergence. It is conventional wisdom that sensor nodes should be powered down periodically to conserve energy [17, 18]. However, our results indicate that if the sensors are switched off during the iterative estimation process, then the network could end up consuming significant additional energy to achieve a specified estimate error tolerance. In the limit as nodes and links



(a) Residual error versus *unidirectional* communication energy



(b) Residual error versus *omnidirectional* communication energy

Figure 5: Simulation results for distributed estimation with the sensor network from Fig. 1(b) using the Jacobi and embedded triangles algorithms. In this case, nodes sleep and communication links fail according to the finite state machine model of Fig. 4. The values $p_1 = 0.1$ (left plots) and $p_1 = 0.3$ (right plots) correspond to nodes sleeping or links failing 17% and 37% of the iterations, respectively, in steady state.

sleep and fail for longer amounts of time, we conjecture that the Jacobi algorithm will be hard to beat.

There are many opportunities for future research and performance enhancements. First, we are studying how to best cycle through a range of different embedded polygon decompositions in order to speed up convergence (by analogy to [9]). Second, we are investigating the optimal size of the embedded polygons. As their size increases, the amount of residual error decay per iteration increases, but so does their sensitivity to sleeping and communication faults. Third, we are developing estimation algorithms for parameters of the Gaussian HMM from sample data (which nodes are connected, and their partial correlations). An expectation-maximization (EM) type approach could prove useful to address this issue. Other future avenues include efficiently estimating the variance of our estimate, extending the algorithm to track changes over time (Kalman filtering), and optimizing the interaction between distributed estimation schemes and ad hoc wireless network routing protocols.[5]

APPENDIX

A. PROOF OF THEOREM 1

In this section, we prove a slightly more general result than Theorem 1. The proof proceeds in two parts. First, we show that iterations of the robust algorithm described in Section 5.1 are *asynchronous iterations*. Asynchronous iterations are known to converge if the iteration operator's spectral radius is < 1 [19]. Second, we prove that the spectral radius of our algorithm's iteration operator is indeed < 1.

As in Section 5.1, let I^m be the set of active (non-sleeping) nodes at iteration m, and let $s_{j\to i}(m)$ denote the iteration count corresponding to the most recent value that sensor i successfully received from j.

At iteration m, an active sensor $i \in I^m$ will either receive from the sensors $j \in \mathcal{N}(i)$ their values $\widehat{x}_j^{m-1}$ (working link) or it will recall from its memory the last value received at the iteration count $s_{j\to i}(m)$, $s_{j\to i}(m) < m-1$ (failing link). This scenario is well-studied in parallel computing and is referred to as *asynchronous iterations* [19].

[5]Many thanks to Dan Sorensen for several fruitful discussions and valuable pointers.

Definition: Asynchronous iteration. For $m \in \mathbb{N}$, let $I^m \subseteq \{1, \ldots, n\}$ and the iterations counts $s_{j\to i}(m) \in \mathbb{N}_0, (i,j) \in \mathcal{G} = (\mathcal{E}, \mathcal{V})$ be such that

$$\begin{array}{lccl} (A) & s_{j\to i}(m) \le m-1 & \text{for} & i,j \in \{1,\ldots,N\}, \\ (B) & \lim_{m\to\infty} s_{j\to i}(m) = \infty & \text{for} & i,j \in \{1,\ldots,N\}, \\ (C) & \#\left(\{m \in \mathbb{N} | i \in I^m\}\right) = \infty & \text{for} & i \in \{1,\ldots,N\}, \end{array}$$

with $\#(\cdot)$ denoting the number of elements in a set. Given an initial guess $\widehat{x}^0$ and the above three conditions, the following iteration is termed *asynchronous*

$$\widehat{x}_i^m = \begin{cases} \widehat{x}_i^{m-1} & \text{for } i \notin I^m \\ H_i(\widehat{x}_1^{s_{1\to i}(m)}, \ldots, (\widehat{x}_n^{s_{n\to i}(m)}) & \text{for } i \in I^m \end{cases} \tag{9}$$

where H_i is the i-th component of an iteration operator H.

The first condition (A) on $s_{j\to i}(m)$ requires that only values computed earlier are used in the current approximation $\widehat{x}_j^m$ of the solution. The second condition (B) requires that as the algorithm proceeds, new information is continually transmitted to each sensor. The third condition (C) stipulates that a sensor cannot sleep forever.

Under the assumptions of Theorem 1, the Robust EPA Update and Solve steps in Section 5.1 constitute an *asynchronous iteration*. Indeed, the Robust Update and Solve steps obviously satisfy condition (A). The assumption requiring bounded delays $d_{j\to i}(m) = m - s_{j\to i}(m)$ implies condition (B). The condition on each node being updated within a finite number of iterations effectively means that a node cannot be always asleep, and this is given by (C).

When solving the linear system $Vx = \overline{y}$ from (2) using a matrix splitting $V = J - K$, the iteration operator H is given by

$$H : \mathbb{R}^n \to \mathbb{R}^n, \quad x \to J^{-1}(Kx + \overline{y}). \tag{10}$$

Theorem 4.1 from [19] then states that if the spectral radius $\rho(|H|) < 1$, then the asynchronous iteration (9) will converge to $\widehat{x}$, the solution of $Vx = \overline{y}$. Theorem 1 is then an immediate corollary of the following general result. Below, we use the notation $U > 0$ (or $U \ge 0$) when a matrix U has all positive (or non-negative) elements.

Theorem 2. *Consider a matrix V as in* (2) *with negative off-diagonal elements. Let $V = J - K$, where J is obtained by setting some of the off-diagonal elements of V equal to zero. Then the iteration operator H in* (10) *is such that $\rho(|H|) < 1$. Hence the asynchronous iteration* (9) *converges.*

Proof. First, note that V is an *M-matrix*, that is, it has negative off-diagonal elements, is nonsingular, and its inverse V^{-1} has all positive elements [15, p. 85]. Indeed, in our Gaussian HMM, V is symmetric and positive definite (since it is the sum of two symmetric positive definite matrices). Corollary 3 in [15, p. 85] yields that $V^{-1} > 0$, that is, that V is an $M-$matrix.

Second, by Theorem 3.12 in [15], J is also an M-matrix. Hence $J^{-1} > 0$ and $J^{-1}K \ge 0$ since by assumption $K \ge 0$. The splitting $V = J - K$ is then termed *regular*. For regular splitting, Theorem 3.13 in [15] tells us that if $V^{-1} \ge 0$, then the spectral radius $\rho(J^{-1}K) < 1$. □

REFERENCES

[1] D. Estrin, R. Govindan, J. Heidemann, and S. Kumar, "Next century challenges: Scalable coordination in sensor networks," in *Proc. ACM/IEEE MobiCom'99*, pp. 263–270, Aug. 1999.

[2] A. Wang and A. Chandrakasan, "Energy-efficient DSPs for wireless sensor networks," *IEEE Signal Processing Mag.*, pp. 68–78, July 2002.

[3] S. L. Lauritzen, *Graphical Models*. Oxford, 1996.

[4] C. B. Barber, D. P. Dobkin, and H. T. Huhdanpaa, "The Quickhull algorithm for convex hulls," *ACM Trans. Math. Software*, vol. 22, no. 4, pp. 469–483, 1996.

[5] K. Plarre and P. R. Kumar, "Extended message passing algorithm for inference in loopy Gaussian graphical models," *Ad Hoc Networks*, 2002.

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

[7] Y. Weiss and W. T. Freeman, "Correctness of belief propagation in Gaussian graphical models of arbitrary topology," *Neural Computation*, vol. 13, pp. 2173–2200, 2001.

[8] M. J. Wainwright, *Stochastic Processes on Graphs with Cycles: Geometric and Variational Approaches*. PhD thesis, ECE Dept., MIT, Jan. 2002.

[9] E. Sudderth, M. J. Wainwright, and A. S. Willsky, "Embedded trees: Estimation of Gaussian processes on graphs with cycles," tech. rep., MIT, 2003.

[10] V. Delouille, R. Neelamani, V. Chandrasekaran, and R. G. Baraniuk, "The embedded triangles algorithm for distributed estimation in sensor networks," in *IEEE Statistical Signal Processing Workshop*, (St. Louis), pp. 357–360, 2003.

[11] R. G. Cowell, A. P. Dawid, S. L. Lauritzen, and D. J. Spiegelhalter, *Probabilistic Networks and Expert Systems*. Springer, 1999.

[12] J. Liebeherr, M. Nahas, and W. Si, "Application-layer multicast with Delaunay triangulations," tech. rep., CS Dept., University of Virginia, 2001.

[13] X.-Y. Li, P.-J. Wan, and O. Frieder, "Coverage in wireless ad-hoc sensor networks," *IEEE Trans. Computers*, vol. 52, pp. 727–741, June 2003.

[14] N. A. C. Cressie, *Statistics for Spatial Data*. Wiley, 1993.

[15] R. S. Varga, *Matrix Iterative Analysis*. Prentice-Hall, 1962.

[16] A. Abdalla, N. Deo, and P. Gupta, "Random-tree diameter and diameter-constrained MST," *Congressus Numerantium*, vol. 144, pp. 161–182, 2000.

[17] J.-F. Chamberland and V. V. Veeravalli, "The art of sleeping in wireless sensing systems," in *IEEE Statistical Signal Processing Workshop*, (St. Louis), pp. 9–12, 2003.

[18] Y. Xu, S. Bien, Y. Mori, J. Heidemann, and D. Estrin, "Topology control protocols to conserve energy in wireless ad hoc networks," Tech. Rep. 6, UCLA, Center for Embedded Networked Computing, Jan. 2003.

[19] A. Frommer and B. Szyld, "On asynchronous iterations," *J. Comput. Appl. Math.*, vol. 123, pp. 201–216, 2000.