DISTRIBUTED WAVELET TRANSFORM FOR IRREGULAR SENSOR NETWORK GRIDS

Raymond Wagner, Hyeokho Choi, Richard Baraniuk *

Rice University
Houston, Texas, USA

Véronique Delouille

Royal Observatory of Belgium
Brussels, Belgium

ABSTRACT

Wavelet-based distributed data processing holds much promise for sensor networks; however, irregular sensor node placement precludes the direct application of standard wavelet techniques. In this paper, we develop a new distributed wavelet transform based on lifting that takes into account irregular sampling and provides a piecewise-planar multiresolution representation of the sensed data. We develop the transform theory; outline how to implement it in a multi-hop, wireless sensor network; and illustrate with several simulations. The new transform performs on par with conventional wavelet methods in a head-to-head comparison on a regular grid of sensor nodes.

1. INTRODUCTION

Wireless sensor networks have emerged as an important application area for distributed signal processing. Sensor networks consist of nodes that sense phenomena of interest, process the measurements, and share data via a wireless, multi-hop routing network. Nodes have limited on-board power supplies, and since communication power consumption typically dominates over processing power by orders of magnitude, intelligent, in-network signal processing is necessary to reduce the amount of transmitted data. Whenever possible, transmissions outside the network should take the form of summarized *results* and *conclusions* rather than *raw data*. Such processing must be both *distributed* — not requiring all data in a central location — and *localized* — requiring access only to data in a node's immediate vicinity.

The restrictions on signal processing algorithms for sensor networks are considerably complicated by the *irregular node placement* typical of real-world deployments. While most traditional regular-grid signal processing techniques do not translate directly to this setting, much of the literature on distributed signal processing for sensor networks has nonetheless assumed regular sampling grids, a fact highlighted in [1]. To help rectify this discrepancy, we propose in this paper what is to our knowledge the first distributed, two dimensional (2-D), irregular-grid wavelet transform for sensor networks capable of multiscale, piecewise-planar approximation of node measurements (two wavelet vanishing moments). We provide a detailed treatment not only of the transform theory (based on wavelet lifting [2]) but also its implementation issues, developing along the way a new distributed triangulation protocol that extends the work of [3].

Section 2 discusses related work in wavelet processing for sensor networks. Section 3 overviews the lifting approach that we exploit in the later sections. Section 4 develops a multiscale structure on the sensor nodes; Section 5 derives the requisite filters; and Section 6 deals with the iteration of the transform. Section 7 develops a new technique for distributed network triangulation. Section 8 demonstrates the approximation power of our transform coefficients with two distributed compression examples. Section 9 concludes and reviews our ongoing work.

*Supported by NSF, AFOSR, ONR, and the Texas Instruments Leadership University Program. Web: compass.cs.rice.edu, dsp.rice.edu
Email: {rwagner, choi, richb}@rice.edu, v.delouille@oma.be

2. RELATED WORK

Multiscale algorithms are not difficult to motivate for sensor network applications, since the laws of physics often induce in the measured data a natural multiscale structure that can guide the scope and extent of in-network signal processing and communication. In particular, as sensors become more distant from each other, the spatial correlations between their measurements will decay rapidly. This suggests *local processing at fine scales* between neighboring nodes and *global processing at coarse scales* between more far-flung nodes.

DIMENSIONS [4] uses an in-network wavelet transform to facilitate querying and storage of sensor network measurements, but it assumes a regular-grid placement of nodes. The same assumption is shared by the wavelet-based Wisden system [5] for structural monitoring. Similarly, [6] proposes separable application of 1-D regular-grid wavelet transforms to solve the 2-D sensor broadcast problem. The lifting-based, regular-grid distributed wavelet transform in [7] is similar in spirit to the one proposed here. It employs a 1-D wavelet decomposition along a path through the 2-D measurement field; however, no method for determining the optimal path is given. While the technique could be extended to use irregular-grid 1-D wavelets, such an approach is not capable of fully capturing the higher-dimensional spatial dependencies among the measurements. Similar conclusions apply to the 1-D Haar protocol described in [8].

The work in [9], which this paper extends, provides an irregular-grid, fully 2-D, distributed wavelet transform for sensor networks, based on piecewise-constant multiscale approximation and multiscale routing structures. In this paper, we develop a distributed lifting transform capable of piecewise-planar approximation and requiring no *a priori* multiscale network structure. There has been significant prior treatment of centralized irregular-grid lifting in both the computer graphics and statistical estimation communities (see [10, 11] and the references therein); we base our distributed scheme here on a technique suggested in [11].

3. WAVELET LIFTING ON IRREGULAR GRIDS

Wavelet lifting [2] replaces the measurement at each sensor network node with a wavelet coefficient representing the inner product of the measured field with a wavelet function in a basis or frame expansion. The process is multiscale, starting from the original node measurements at some finest scale J and iterating to a final, coarsest scale 0. Denote the complete set of nodes by $E(J)$ and the complete measurements by s_J; these comprise the finest-scale set of *scaling coefficients*. We assume that each node knows its location in space; providing this information is an area of active research [12].

Adopting with slight modification the notation of [11] and starting with $j = J - 1$, we say that each scale j of the transform first *splits* the set of nodes $E(j+1)$ into a set of $\#E(j)$ "even" coefficients located at nodes $E(j)$ and a set of $\#O(j)$ "odd" coefficients located at nodes $O(j)$.[1] This divides the set of scaling coefficients into two new sets $s_{j+1,E(j)}$ and $s_{j+1,O(j)}$. The coefficients in $s_{j+1,O(j)}$ give rise to the scale–j *wavelet coefficients* $d_{j,O(j)}$. We say that these are *predicted* using $s_{j+1,E(j)}$ and refer to the nodes in $O(j)$ as *predicted nodes.*The coefficients in $s_{j+1,E(j)}$ are then *updated* to $s_{j,E(j)}$ using $d_{j,O(j)}$ in order to preserve the weighted average of the scaling coefficients; the nodes in $E(j)$ are known as *updated nodes.*

The scale–j set of scaling coefficients $s_{j,E(j)}$ replaces the even scaling coefficients from scale $j+1$, and the transform then iterates over $s_{j,E(j)}$ at coarser scale $j-1$, continuing until a root set of scaling coefficients resides at the nodes in $E(0)$. At that point, the set of terminal scaling coefficients s_0 and the entire set of wavelet coefficients $\{d_j\}_{j\in\{0,\ldots,J-1\}}$ number the same as the original field measurements at $E(J)$ and completely describe them — that is, the entire process is invertible.

The key issues in any lifting decomposition are (1) determining which values are to be *decimated* at each scale (i.e., partitioning nodes into $E(j)$ and $O(j)$), and (2) defining filters to calculate the scaling and wavelet coefficients at each scale. In a regular-grid setting, the grid structure guides both these choices — decimation to a coarser grid is trivial, and the same set of scaling and wavelet filters can be applied at each grid point. With grid irregularity, choosing which nodes to decimate becomes more complicated, and the lack of a predictable spatial neighbor distribution necessitates adapting different filter coefficients for each node.

The technique in [9] exploits the simplicity of piecewise-constant multiscale approximation and leverages hierarchical routing structures (see [13] for an example) to sidestep these two issues. The routing groups nodes into clusters, and the transform assigns within each cluster a scaling coefficient (average) to an elected clusterhead node while assigning wavelet coefficients (differences from the average) to the other nodes. Clusterheads participate in the next transform level. The routing hierarchy thus guides decimation, and simple averaging avoids the need for filters that depend on spatial neighbor distributions. Unfortunately, when the data are smoother than piecewise-constant, this approximation will not completely sparsify the wavelet coefficients, degrading performance in applications such as compression and de-noising. To enable higher-order multiscale approximation and to free ourselves from dependency on multiscale routing structures, we must meet both the challenges of decimation and local filter design head-on.

4. SENSOR NODE DECIMATION

At each scale j we separate predicted nodes from the previous scale into predicted nodes $O(j)$ and updated nodes $E(j)$, where $E(j+1) = E(j) \cup O(j)$. Naturally, $O(j)$ and $E(j)$ must be disjoint. Additionally, we wish the decimated set $O(j)$ to be as large as possible at each scale in order to have as few scales as possible. To accomplish this, we construct a mesh to guide the decimation — in particular (but without loss of generality) a *Delaunay triangulation* (DT) of the nodes at scale j [10, 11]. DT's can be built in a distributed fashion, as detailed in Section 7. The triangulation is calculated once (at the finest level of the transform) and then locally updated as points are removed at each scale. To begin, each node need only discover its location and share this information among nodes within its communication range (see [12] and the references therein).

Using the mesh, we first define the *neighbors* of a node at scale j as those nodes with which it shares an edge in the scale–j mesh. We can then designate which points to remove at each transform level by appealing to a notion of *scale* in the irregular grid. The mesh allows us to assign an *area of support* to each node — a computation detailed in Section 5.3. While in a regular grid each node has the same support area, each node in an irregular grid has a different area and, thus, a unique scale. Since multiscale analysis provides increasingly broad views of a measurement field at each scale, it follows naturally that we should remove the smallest-scale (-area) points at each transform level. Recall, however, that a node predicted at level j cannot have predicted neighbors at that level. To avoid violating these conditions, we propose the following greedy solution to decimating nodes:

1. Mark all nodes on the mesh boundary as updated, placing them in $E(j)$.
2. Mark the node with the smallest area among unmarked nodes as predicted, placing it in $O(j)$.
3. Mark the predicted node's neighbors in the mesh as updated, placing them in $E(j)$.
4. Return to Step 2 while unmarked nodes remain.

Step 1 ensures the stability of the predict stage (discussed further in Section 5.2) and guarantees that boundary nodes form the coarsest set $E(0)$. While this protocol is essentially a centralized one, it provides an elegant solution to decimation with relatively low overhead. One could imagine a message-passing scheme where nodes eventually agree on which one has the next smallest area; however, the associated communication overhead could become prohibitive as the network grows. The triangulation algorithm is deterministic given node locations, and so we can easily compute the decimation order outside the network after gathering the nodes' self-localized positions and then inform individual nodes of the transform level at which they are decimated.

Figure 1 illustrates two successive levels of decimation. Predicted nodes at scale j are marked with dark circles in Figure 1(a). Following removal of these points and local re-triangulation, a new set of predicted nodes is designated at scale $j-1$ in Figure 1(b). Experimentally, this technique removes approximately 25% of the nodes per scale.

5. DISTRIBUTED FILTER DESIGN

Once the sets $O(j)$ and $E(j)$ have been designated, we must compute the scale–j wavelet coefficients for the nodes in $O(j)$ and scaling coefficients for the nodes in $E(j)$. This entails assigning predict and update filters to operate on the coefficients at neighboring nodes. When the node locations remain fixed, both the decimation order and filter coefficients need only be computed once.

[1] # denotes here the number of elements in a set.

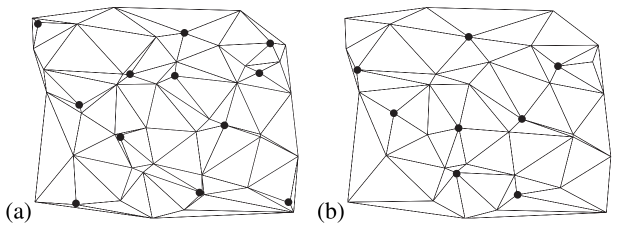


Fig. 1. Triangulated mesh of sensor nodes (a) at scale j and (b) at coarser scale $j-1$ (• denotes vertices to be decimated).

While we suggest that the decimation structure be centrally computed and disseminated to the nodes as outlined in Section 4, the filter coefficients can be efficiently computed in a distributed fashion and stored locally at the nodes for future reference.

5.1. Refinement relations

We first specify how the wavelet and scaling functions are related across scales in our piecewise-planar biorthogonal wavelet transform, which is inspired by Section 7.2.3 of [11]. Label the row vector of scaling functions to be generated at scale j for nodes in $E(j)$ as $\Phi_{j,E(J)}$, and let the scaling coefficients $s_{j,E(j)}$ form a column vector. Similarly, let $\Psi_{j,O(j)}$ represent the row vector of scale–j wavelet functions at nodes in $O(j)$ with an associated column vector $d_{j,O(j)}$ of wavelet coefficients. To relate the scaling functions from the finer scale $j+1$ to the scaling and wavelet functions at scale j, define an $\#O(j) \times \#E(j)$ predict filter P_j and an $\#E(j) \times \#O(j)$ update filter U_j to operate as follows

$$\begin{aligned} \Phi_{j,E(j)} &= \Phi_{j+1,E(j)} + \Phi_{j+1,O(j)}P_j \\ \Psi_{j,O(j)} &= \Phi_{j+1,O(j)} - \Phi_{j,E(j)}U_j. \end{aligned} \tag{1}$$

Similarly, P_j and U_j relate the scaling coefficients from scale $j+1$ to the scaling/wavelet coefficients at coarser scale j as

$$\begin{aligned} d_{j,O(j)} &= s_{j+1,O(j)} - P_j s_{j+1,E(j)} \\ s_{j,E(j)} &= s_{j+1,E(j)} + U_j d_{j,O(j)}. \end{aligned} \tag{2}$$

Without careful design the transform may be far from orthogonal, resulting in coefficients whose impact on reconstruction in the spatial domain is poorly related to their magnitudes in the wavelet domain. Entries in P_j and U_j must therefore be chosen so that the transform is as close to orthogonal as possible. The filters also control how distributed and local the transform will be. An element in a row of P_j or U_j must be nonzero only if the node corresponding to that element's column is a neighbor of the node corresponding to the element's row in the scale–j mesh. Moreover, data gathered in *calculating* these predict and update filter coefficients at each node must come only from neighbors of each node at scale j. With these requirements in mind, we describe how to locally fill in the entries of P_j and U_j and discuss the associated communication traffic.

5.2. Predict filters

The predict filter has the most straightforward design. We desire the transform to have two vanishing moments; that is, if the data are planar, then the wavelet coefficients should be zero. To compute the wavelet coefficient at a predicted point $n_p \in O(j)$, we regress a plane through its neighbors in the scale-j mesh. Denote these neighbors as $\mathcal{N}(n_p) \in E(j)$. Per (2), the value of the wavelet coefficient is the difference between the scale-$(j+1)$ scaling coefficient at n_p and the value predicted by the plane at the location of n_p. The $1 \times \#\mathcal{N}(n_p)$ vector $\underline{p}_{j,n_p}$ of predict coefficients[2] at n_p is given by

$$\underline{p}_{j,n_p} = [1, x(n_p), y(n_p)](X'X)^{-1}X', \tag{3}$$

where $x(\cdot)$ and $y(\cdot)$ give the x and y coordinates of a node and the $\#\mathcal{N}(n_p) \times 3$ matrix $X = [\mathbf{1}, x(\mathcal{N}(n_p)), y(\mathcal{N}(n_p))]$ (where $\mathbf{1}$ is a column vector of ones). Properly sorted, the coefficients in $\underline{p}_{j,n_p}$ fill sensor n_p's row in P_j at the positions corresponding to neighbors $\mathcal{N}(n_p)$. The remainder of the elements in the row, corresponding to more distant nodes, are zero, and thus the entire process is local. Figure 2(a) depicts the one-time local data flow at scale j during computation of predict filter coefficients, with neighbors of predicted nodes transmitting their (x, y) coordinates for use in (3).

This predict scheme illuminates why care is taken in Section 4 to exclude boundary nodes when selecting nodes for prediction. When the angular distribution in polar coordinates of neighbors around a predicted node does not cover most of the range $[0, 2\pi)$, planar regression becomes extrapolation in the region with no data points, leading to numerical instability. Several intricate, centralized techniques are mentioned in [11] to deal with this instability. To counter this problem in a distributed fashion, we simply choose never to decimate nodes on the boundary of the finest-level mesh, since these are precisely the nodes that do not have neighbors throughout the surrounding $[0, 2\pi)$ space.

5.3. Scaling function support areas

Computing the update coefficients, to follow shortly, requires maintaining integrals of the scaling functions at scale j in terms of integrals of the scaling functions at scale $j+1$. If we consider the scaling functions at the finest scale to be indicator functions over pseudo-Voronoi regions surrounding the nodes, then this amounts to calculating and updating the scaling function support areas. The first set of integrals is extracted from the finest-scale mesh, with each node assigning to itself $1/3$ the area of each triangle to which it belongs [10].

Denote by A_{j,n_u} the integral of scaling function $\phi_{j,n_u} \in \Phi_{j,E(j)}$ at updated node $n_u \in E_j$. Integrating the first equation of (1) at n_u yields

$$A_{j,n_u} = A_{j+1,n_u} + \sum_{n_k \in \mathcal{N}(n_u)} \underline{p}_{j,n_k}(n_u) A_{j+1,n_k}, \tag{4}$$

where $\mathcal{N}(n_u)$ describes the neighbors of n_u whose wavelet coefficients are *predicted* at scale j; $\underline{p}_{j,n_k}(n_u)$ gives the predict coefficient node n_k applies to the scale–$(j+1)$ scaling coefficient of node n_u. Essentially, to compute the integral of the scale–j scaling function for an updated node n_u, we add the integral of its scale–$(j+1)$ scaling function to a weighted sum of the scale–$(j+1)$ scaling function integrals of its predicted neighbors. The prediction coefficients for n_u at those neighbors provide the sum weights. Figure 2(b) illustrates the one-time communication traffic

[2]Underlining is used here to distinguish *locally* computed filter coefficient vectors.

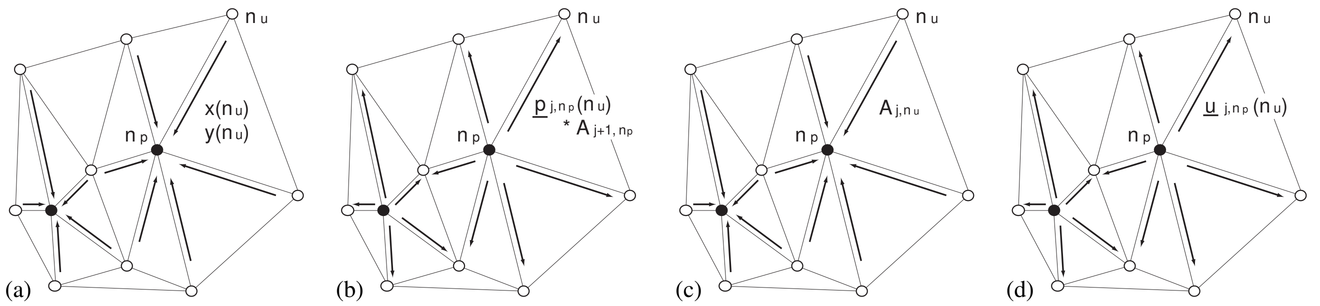


Fig. 2. One-time communication at scale j between predicted ($\bullet$) and updated ($\circ$) nodes during filter coefficient calculation. Messages are labelled between a predicted node n_p and an updated neighbor n_u. (a) n_u sends its (x, y) coordinates to each node it predicts; (b) n_p sends its weighted area $\underline{p}_{j,n_p}(n_u)A_{j+1,n_p}$ to n_u; (c) n_u sends its new area A_{j,n_u} to n_p; (d) n_p sends its calculated updated coefficient for n_u, $\underline{u}_{j,n_p}(n_u)$, to n_u.

generated during area re-calculation, with predicted nodes transmitting their areas weighted by predict coefficients to each updated node participating in the prediction.

5.4. Update filters

To ensure transform stability, the update operator should keep the sum of scaling coefficients weighted by the scaling function integrals constant across scale. Wavelet biorthogonality relations equate this requirement to providing each wavelet function with a zero integral. Integrating the second equation of (1), setting the left-hand side to zero, and solving for the update matrix U_j indicates that we must satisfy $A_{j+1,O(j)} = A_{j,E(j)}U_j$. While [11] recommends a centralized solution of this equation, it can be solved column-wise with virtually no reduction in approximation performance.

We apply to each column of U_j the *minimum-norm* approach first suggested in [10], which has a stabilizing effect on the overall transform. For each predicted node $n_p \in O(j)$ we must find the minimum-norm update coefficient column vector $\underline{u}_{j,n_p}$ such that $A_{j+1,n_p} = A_{j,\mathcal{N}(n_p)}\underline{u}_{j,n_p}$ (where $A_{j,\mathcal{N}(n_p)}$ is a row vector of neighbor areas). This can be solved by a simple pseudo-inverse, so that

$$\underline{u}_{j,n_p} = \frac{A'_{j,\mathcal{N}(n_p)}A_{j+1,n_p}}{\sum_{n_k\in\mathcal{N}(n_p)} A^2_{j,n_k}}. \tag{5}$$

Since $\underline{u}_{j,n_p}$ only applies to neighbors of n_p in the level–j mesh, calculation of update coefficients is again local. Note that the properly sorted elements of $\underline{u}_{j,n_p}$ represent non-zero entries in the *column* of U_J corresponding to node n_p, since the predicted node is computing these update coefficients.

Figure 2(c) depicts the one-time data flow required for computing the update filter coefficients, with scale-j areas of updated nodes passing to predicted nodes for update coefficient calculation. The appropriate update coefficient is then sent in Figure 2(d) back to each updated node for use in subsequent transformations.

5.5. Managing coefficient calculation

Assuming adequate synchronization among nodes (see [14], for example), no central management is required to calculate the predict and update filters. The distributed computation at scale j proceeds as follows: Predicted nodes first wait for an interval appropriate for the scale to ensure that they learn of all neighbors in the scale–j mesh. Then, they exchange data with these neighbors and compute the transform coefficients for both themselves and their updated neighbors. Finally, they re-triangulate for the next level of the transform and inform their updated neighbors of the new connectivities.

This procedure, of course, assumes ideal communication links. Since the network and the decoder must agree on the neighbors of each node, extra care will be necessary to ensure encoder/decoder synchronization if the communication links are unreliable. While such a protocol is beyond the scope of this paper, we note that its overhead can be amortized over several transform calculations, since the filter coefficients are calculated only once.

6. DISTRIBUTED TRANSFORM COMPUTATION

Once the predict and update filter coefficients have been locally calculated and stored for each level $j \in \{J-1, ..., 0\}$ of the transform, multiple wavelet transformations of measured data can proceed using these filter coefficients. Following (2), we first compute a wavelet coefficient d_{j,n_p} for each predicted sensor n_p and then compute a scaling coefficient s_{j,n_u} for each updated sensor n_u using the new set of wavelet coefficients.

As before, denote the set of neighbors around a predicted node n_p as $\mathcal{N}(n_p)$, and let $\underline{p}_{j,n_p}$ be the row vector of predict coefficients computed at n_p in (3). Using its scale–$(j+1)$ scaling value and those from its neighbors, n_p computes its scale–j wavelet value as

$$d_{j,n_p} = s_{j+1,n_p} - \underline{p}_{j,n_p}s_{j+1,\mathcal{N}(n_p)}, \tag{6}$$

where $s_{j+1,\mathcal{N}(n_p)}$ is a column vector of scaling coefficients. Figure 3(a) depicts the required local data flow, with neighbors simply transmitting their scale–$(j+1)$ scaling coefficients to node n_p.

Similarly, let $\mathcal{N}(n_u)$ again describe the predicted neighbors in the scale–j mesh of an updated node n_u. Place the update coefficients calculated for n_u by its predicted neighbors in the row vector $\underline{u}_{j,n_u}$. Note that this *row* vector contains the properly sorted nonzero elements of n_u's row in U_j; it should not be confused with any of the *column* vectors of update coefficients calculated at *predicted* sensors in (5). Using its scale–$(j+1)$ scaling value and the scale–j wavelet values from its neighbors in $\mathcal{N}(n_u)$, n_u computes its scale-j scaling value as

$$s_{j,n_u} = s_{j+1,n_u} + \underline{u}_{j,n_u}d_{j,\mathcal{N}(n_u)}, \tag{7}$$

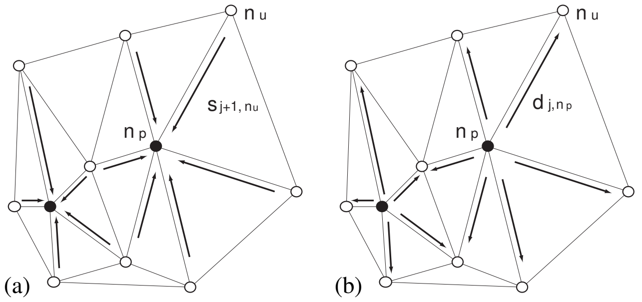


Fig. 3. Communication at scale j between predicted (•) and updated (○) nodes repeated during each wavelet transform. Messages are labelled between a predicted node n_p and an updated neighbor n_u. (a) n_u sends its scale–$(j+1)$ scaling coefficient s_{j+1,n_u} to each predicted neighbor; (b) n_p sends its scale-j wavelet coefficient d_{j,n_p} value to each updated neighbor.

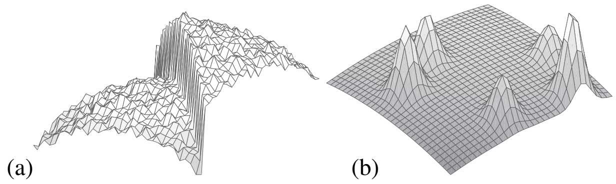


Fig. 4. Test data fields: (a) noisy quadratic bump with discontinuity, and (b) Gaussian bumps on a smooth, quadratic field.

where $d_{j,\mathcal{N}(n_u)}$ is a column vector of wavelet coefficients. Figure 3(b) illustrates the associated traffic pattern, with each updated node passing its scale–j wavelet coefficient to each of its neighbors to enable their scaling values calculations.

Again, no centralized administration is necessary to manage this process, apart from all nodes in the network agreeing to begin a new transform calculation — a process that can proceed at a time-scale appropriate for the measured phenomenon and/or operator interest. When the measured field is transformed to scale j, each predicted node n_p merely waits for nodes in $\mathcal{N}(n_p)$ to send their scale–$(j+1)$ scaling coefficients before computing its scale–j wavelet coefficient. Similarly, each n_u waits for the scale-j wavelet coefficients from its predicted neighbors before computing its new scaling coefficient. The transform proceeds through scales in a completely asynchronous and distributed fashion, never progressing to the next, coarser scale until all computations at the previous, finer scale are complete.

7. DISTRIBUTED TRIANGULATION

As detailed in Section 3, a mesh is required at each scale of the transform to provide connectivity for the predict and update calculations, and areas extracted from the mesh at the starting scale are required to initialize the scaling function integral re-calculation of Section 5.3. For the lifting transform to be truly distributed, construction and maintenance of the mesh must be distributed.

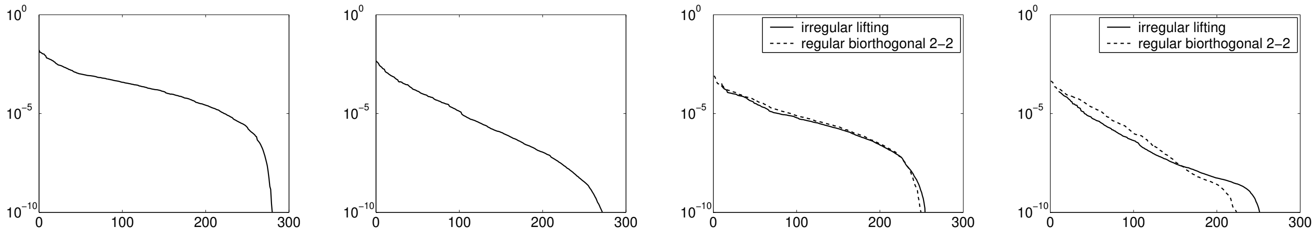
With some modifications, the algorithm from [3] suffices to compute a triangulated mesh. The algorithm builds a Planar Localized Delaunay triangulation (PLDel). Given a node communication radius, the resulting PLDel contains the edges in a centralized DT whose lengths do not exceed the communication radius. First, nodes broadcast their locations, and each node collects the locations of neighbors within its communication radius. Each node then locally computes a Delaunay triangulation based on its collected points and broadcasts the associated triangles to its neighbors within communication range. Once a node has received these triangle messages, it consolidates message triangles with locally computed triangles to obtain a final set of valid Delaunay triangles with vertices at that node. The triangle edges with the node as an endpoint, along with any non-included *Gabriel* edges,[3] form the set of PLDel edges rooted at that node. Edge discovery is guaranteed to be two-way, meaning that each edge is discovered by both of its vertices.

We require more than just connectivity, however. A node must also know about the triangles rooted at itself for area computation; unfortunately, the triangulation may produce edges whose lengths exceed the nodes' communication radius by up to a factor of 2. This means that, while a valid triangle could be discovered at one vertex, the other pair of vertices in the triangle may be out of radio range of each other and hence will not themselves discover the triangle. Allowing a two-hop communication path between these vertices does not present a problem in our application; indeed, paths between vertices will generally become multi-hop as the transform scale becomes coarser. Therefore, we propose adding an additional communication step to the method of [3]: Once a node has identified its set of valid triangles, it informs the other pair of nodes in each triangle of their membership in that triangle. Using found and received triangles, a node can then incorporate any non-included Gabriel edges, forming additional triangles. While this requires some additional negotiation with neighbors, the number of Gabriel edges remaining after the second round of message passing is typically very small in practice.

Ideally, the node density and communication radius will both be sufficiently high so that most edges of the perfect DT have lengths within twice that radius. When this is the case, the distributed triangulation will produce a mesh such as Figure 1(a), where all but the exceptionally long edges at the boundary of the DT are captured in the distributed mesh. When interior links of the DT exceed this bound, the distributed mesh will exhibit non-triangular polygons in its interior, whose perimeters create interior "boundaries" that must be treated similarly to exterior boundaries to ensure stability of the predict operator of Section 5.2.

When the node density and communication radius allow the distributed triangulation to capture all but the longest edges on the boundary of the DT, re-triangulation of the mesh after decimation follows easily using the technique of [15]. Since only interior vertices are subject to decimation, such nodes can locally compute new Delaunay triangles and inform their previous neighbors of their new connections with each other, maintaining a Delaunay mesh throughout scale. In the case that the distributed triangulation differs on its interior from the DT, mesh refinement can proceed with the predicted point linking its closest neighbor to the others.

[3] A Gabriel edge is one for which the circle whose diameter coincides with the edge does not contain any other nodes of the triangulation.



(a) (b) (c) (d)

Fig. 5. Results of nonlinear approximation experiments. Mean square error (log-scale) is plotted on the y-axis with the number of approximation coefficients used in the reconstruction on the x-axis. We note the rapid decay of the approximation error of (a) the noisy, discontinuous quadratic field and (b) the smooth Gaussian bump field. Performance is comparable to more traditional biorthogonal CDF(2,2) wavelets on square-grid sampling for (c) the noisy, discontinuous quadratic and (d) smooth Gaussian bump fields.

8. DISTRIBUTED APPROXIMATION EXAMPLES

We present two brief evaluations showcasing the effectiveness of our proposed transform.

First, echoing an example of [1], we simulate in-network nonlinear approximation (compression without coefficient quantization) of the transform coefficients prior to transmission to an external data sink. In this scenario, the sink floods the network with a magnitude threshold below which wavelet coefficients are considered insignificant. Each node with a significant coefficient passes its coefficient value and node ID to the sink, which reconstructs the measurement field via the inverse wavelet transform. We include successively smaller coefficients in the reconstruction to generate nonlinear approximation curves of increasing fidelity. We consider the two fields shown in Figure 4; the first is a noisy quadratic bump with a discontinuity, and the second is a set of random Gaussian bumps populating a smoothly varying quadratic field. Both examples contain super-planar features beyond the scope of piecewise-planar approximation. For each field, we created 300 sample points from uniformly distributed random node locations. Figures 5(a) and (b) display the nonlinear approximation results for each, plotting along the y-axis the mean-square approximation error on a log scale and along the x-axis the number of wavelet coefficients used in the approximation. Both curves exhibit the smooth, rapidly decreasing decay we expect from a wavelet transform on piecewise-smooth data.

Second, as a sanity check, we also compare the performance of our transform to a biorthogonal wavelet transform with the same number of vanishing moments (CDF(2,2) [4]) on a *regularly spaced* square grid of 256 grid points. We compare the nonlinear approximation curves, as shown in 5(c) and (d). The solid lifting approximation curve is nearly coincident with the CDF dashed curve, indicating that our distributed lifting transform is competitive with traditional wavelet techniques.

9. CONCLUSIONS AND FUTURE WORK

We have developed a fully distributed, irregular-grid wavelet transform and protocol for sensor networks that is capable of piecewise-planar multiscale approximation. We have presented distributed solutions to implementation issues included mesh building, filter coefficient calculation, and transform coefficient calculation. The transform coefficients have desirable performance under threshold-based processing such as distributed compression and even perform competitively with centralized, regular-grid wavelet techniques. In current work, we are developing a suite of distributed processing algorithms based on this transform and exploring issues of higher order approximation, transform stability, transform coefficient quantization, and extension to 3-D grids.

10. REFERENCES

[1] D. Ganesan, S. Ratnasamy, H. Wang, and D. Estrin, "Coping with irregular spatio-temporal sampling in sensor networks," *SIGCOMM Comput. Commun. Rev.*, vol. 34, no. 1, pp. 125–130, 2004.

[2] W. Sweldens, "The lifting scheme: A construction of second generation wavelets," *SIAM J. on Math. Anal.*, vol. 29, no. 2, pp. 511–546, Mar. 1998.

[3] F. Araújo and L. Rodrigues, "Fast localized Delaunay triangulation," in *Proc. 8th Int. Conf. on Principles of Dist. Sys. (OPODIS)*, Grenoble, France, Dec. 2004.

[4] D. Ganesan, B. Greenstein, D. Estrin, J. Heidemann, and R. Govindan, "Multiresolution storage and search in sensor networks," *ACM Trans. on Storage*, vol. V, no. N, Apr. 2005.

[5] N. Xu, S. Rangwala, K. Chintalapudi, D. Ganesan, A. Broad, R. Govindan, and D. Estrin, "A wireless sensor network for structural monitoring," in *Proc. ACM SenSys Conference*, Nov. 2004.

[6] S.D. Servetto, "Distributed signal processing algorithms for the sensor broadcast problem," in *Proc. 37th Annual Conf. on Information Sciences and Systems (CISS)*, Mar. 2003.

[7] A. Ciancio and A. Ortega, "A distributed wavelet compression algorithm for wireless multihop sensor networks using lifting," in *Proc. IEEE Int. Conf. on Acoustic and Speech Sig. Proc. (ICASSP)*, Mar. 2005, pp. IV–825—IV–828.

[8] J. Aćimović, R. Cristescu, and B. Beferull-Lozano, "Efficient distributed multiresolution processing for data gathering in sensor networks," in *Proc. IEEE Int. Conf. on Acoustic and Speech Sig. Proc. (ICASSP)*, Mar. 2005, pp. IV–837—IV–840.

[9] R. Wagner, S. Sarvotham, and R. Baraniuk, "A multiscale data representation for distributed sensor networks," in *IEEE Int. Conf. on Acoustics, Speech, and Sig. Proc. (ICASSP)*, Mar. 2005.

[10] M. Jansen, G. Nason, and B. Silverman, "Scattered data smoothing by empirical bayesian shrinkage of second generation wavelet coefficients," in *Wavelet App. in Sig. and Image Processing IX, Proc. of SPIE*, 2001, vol. 4478, pp. 87–97.

[11] V. Delouille, *Nonparametric Stochastic Regression Using Design-Adapted Wavelets*, Ph.D. thesis, Universite Catholique de Louvain, 2002.

[12] L. Hu and D. Evans, "Localization for mobile sensor networks," in *Proc. 10th Ann. Int. Conf. on Mobile Comp. and Net. (MobiCom)*, 2004, pp. 45–57.

[13] S. PalChaudhuri, R. Kumar, R.G. Baraniuk, and D.B. Johnson, "Design of adaptive overlays for multi-scale communication in sensor networks," in *Proc. Int. Conf. on Distributed Computing in Sensor Systems (DCOSS)*, Jun. 2005.

[14] J. Elson, L. Girod, and D. Estrin, "Fine-grained network time synchronization using reference broadcasts," *SIGOPS Oper. Syst. Rev.*, vol. 36, no. SI, pp. 147–163, 2002.

[15] O. Devillers, "On deletion in Delaunay triangulations," in *Proc. 15th Annl. Symp. on Computational Geometry (SCG)*, 1999, pp. 181–188.