Contour Approximation in Sensor Networks

Chiranjeeb Buragohain[1], Sorabh Gandhi[1],
John Hershberger[2], and Subhash Suri[1]

[1] Dept. of Computer Science, University of California, Santa Barbara, CA 93106*
{chiran,sorabh,suri}@cs.ucsb.edu
[2] Mentor Graphics Corp., 8005 SW Boeckman Road, Wilsonville, OR 97070
john_hershberger@mentor.com

Abstract. We propose a distributed scheme called Adaptive-Group-Merge for sensor networks that, given a parameter k, approximates a geometric shape by a k-vertex polygon. The algorithm is well suited to the distributed computing architecture of sensor networks, and we prove that its approximation quality is within a constant factor of the optimal. We also show through simulation that our scheme outperforms several other alternatives in preserving important shape features, and achieves approximation quality almost as good as the optimal, centralized scheme. Because many applications of sensor networks involve observations and monitoring of physical phenomena, the ability to represent complex geometric shapes faithfully but using small memory is vital in many settings.

1 Introduction

We consider the problem of approximating polygonal paths and cycles in the context of a sensor network. The problem of representing complex geometric shapes using small memory is fundamental in many sensor net applications: sensor networks observe, measure, and track physical phenomena, which often involves representing and communicating a geometric shape. The problem arises, for example, in the application of computing contour lines on a field of sensor measurements [8]. Suppose that a geographically distributed set of sensors measures some physical parameter, say temperature, that varies smoothly over the sensor region. An analyst is interested in the rough shape of the temperature distribution, but does not care about the exact values measured by all the sensors. A collection of *isocontours*—cycles along which the measured and interpolated sensor values are constant—can be a useful summary of the distribution.

Contour lines reduce the data to be reported from two dimensions (the full set of sensors) to one dimension (dependent on only those sensors near the contour). However, even this reduction may not be enough. Communication is arguably the most important resource in a sensor net, and a one-dimensional contour whose feature size depends on the spacing of the sensors may contain too much

* The research of C. Buragohain, S. Gandhi and S. Suri was supported by grants from the National Science Foundation (CCF-0514738) and Army Research Organization (DAAD19-03D0004).

data to send through the network back to the analyst. Therefore, it is important to consider methods for simplifying a one-dimensional contour that approximate the original data well and can be computed by a distributed network.

We use "contour approximation" as a guiding application, but our treatment of the problem is at an abstract level: distributed algorithms to compute a bounded-memory approximation of a polygonal curve embedded in a sensor field. Because sensor networks are envisioned as distributed "spatial instruments" that take measurements in a physical space but have limited resources (bandwidth, power, etc.), the ability to represent complex geometric shapes faithfully but using small memory is vital to sensor networks. In particular, significant improvement in system lifetime is possible if the network performs local computation to build compact approximations instead of sending the entire raw data to a centralized location. Indeed, a number of techniques have been proposed recently for "in-network aggregation" of sensor data [8, 12, 16]. The focus of these papers is on numerical summaries, such as min, max, average, or median, while the main focus of our paper is a fundamental form of *spatial summary*. Imagine, for instance, a physical phenomenon, such as a structural fault, that is evolving with time, and an analyst who wants to receive a periodic snapshot of the general shape of the phenomenon. Another possible application is building a compact representation of the boundary of the entire sensor field, which can be broadcast efficiently throughout the network so that each node knows the overall geographical coverage of the system. Awareness of the sensor field's shape can be useful in data storage schemes like Geographical Hash Tables (GHT) that associate data with geometric locations.

The problem of contour approximation was considered by Hellerstein et al. [8] in a sensor net setting. They proposed an algorithm in which a contour is initially approximated by its axis-aligned bounding box, and then the approximation is successively refined. At each step the approximate polygon encloses the original contour. Each refinement step deletes from the current approximation the maximum-area rectangular notch that lies outside the original contour. The refinement stops when the approximating polygon reaches some target complexity (number of vertices). This approach, while a useful heuristic, has several liabilities: (1) the restriction to rectangular approximation imposes an axis-dependence where none is required by the data; (2) the greedy maximization of area removed at each step does not ensure that the approximating polygon is near the original; and (3) the algorithm is a heuristic, with no proof of approximation quality. In [17], Singh, Bakshi, and Prasanna consider the problem of producing topographic maps over a sensor field using a quadtree-based approach, but they do not focus on constructing a compact representation of the map.

Approximating polygons is a fundamental problem that has been considered in many fields, including geographic information systems (GIS), computer vision, and computational geometry. In these settings the computational model favors centralized computation, in which all the input data are available to a single computational engine. Performance is measured in terms of approximation quality (in any of a variety of metrics) and running time/memory usage as a function

of the input size n and the output size k. Typical algorithms include dynamic programming (which can optimize most metrics in roughly $O(n^2k)$ time [10, Chapter 3]) and the Douglas–Peucker algorithm (which provides good practical approximation quality in $O(n \log n)$ time [4, 9]). Because of the centralized computation requirement, however, these algorithms are ill-suited for use in a sensor net setting without significant adaptation.

Our Contribution

We make the following contributions in this paper: (1) We propose a new distributed algorithm, called Adaptive-Group-Merge (AGM), for polygon approximation with a worst-case constant factor *approximation guarantee*. (2) We develop a *distributed* wavelet-based scheme as a natural, simple alternative to AGM. (3) We show through simulation that AGM significantly outperforms the wavelet scheme in approximation quality. (4) Our experiments show that, in fact, AGM performs almost as well as the centralized, dynamic-programming-based optimal scheme. Thus, our new scheme is able to combine the virtues of the two extreme alternatives: it delivers the approximation quality of the centralized optimal scheme, but it incurs a computational and communication cost comparable to the wavelet scheme.

One of the most attractive features of our algorithm is its *locality*, which makes it highly suitable for heterogeneous multi-tiered sensor architectures, such as Tenet [7, 19]. These networks include a small number of high-powered (tier 1) nodes that act as clusterheads for many low-powered, mote-caliber (tier 2) devices. The motes simply collect and send their data to a neighboring clusterhead—the application software runs only on the tier 1 nodes. Using AGM, each tier 1 node can approximate its own portion of the contour *without jeopardizing* the global approximation quality. These partial contour approximations then can be exchanged among the tier 1 nodes to compute the final approximation. By contrast, centralized schemes such as dynamic programming or Douglas-Peucker require global knowledge of the data to decide which portions of the contour to keep, and thus are not amenable to distributed computation.

2 Preliminaries

We make the following assumptions about the sensors in the network: each sensor has a fixed radio range r, it knows its geographical location by using some localization technique [1, 14] and every sensor knows its neighbors' positions (other sensors within a circle of radius r). These assumptions, though somewhat idealized—radio ranges are not disks in practice [11, 20], and localization is nontrivial—are fairly standard in sensornet research, and allow us to focus on the approximation problem of interest. At the same time, we make no assumptions about the distribution of sensors in the field, or the shape of the field, so our results apply to an *arbitrary* collection of sensors.

Many different metrics have been used to measure the quality of a polygonal approximation. Two common choices are the L_p metrics and the Hausdorff metric. Given a polygonal curve S (a *polyline*) whose vertex sequence is $(p_1, p_2, \ldots, p_n)$, let $A = (a_1, a_2, \ldots, a_k)$ be a k-point approximation of S. To measure the approximation using the L_p metric, let $S' = (p'_1, p'_2, \ldots, p'_n)$ be the points on the polyline A closest to the corresponding vertices of S. Define the point coordinates to be $p_i = (x_i, y_i)$ and $p'_i = (x'_i, y'_i)$. Then the L_p approximation error of A is

$$\varepsilon_p \equiv ||S - S'||_p \equiv \left(\sum_i (|x_i - x'_i|^p + |y_i - y'_i|^p) \right)^{1/p}.$$

In particular ε_2 is the Euclidean mean squared error and ε_∞ is the maximum error. To define the Hausdorff approximation error, let $d(p, Q)$ be the minimum Euclidean distance from a point p to a polyline Q. Then the Hausdorff distance between S and A is

$$H(S, A) \equiv \max(\max_{0 \le i < n} d(p_i, A), \max_{0 \le j < k} d(a_j, S)).$$

Given the above definition of the distance $d(p, Q)$ between a point p and a polyline Q, we can think of the Haussdorff error as follows. The Hausdorff error between two polylines is the maximum distance of a point on either of the two polylines from the other polyline.

We will evaluate the efficiency of our algorithms primarily in terms of total communication complexity (also known as *message complexity*). If an algorithm requires N message transmissions, with each message of size m, then the communication complexity of the algorithm is defined to be $O(Nm)$. We will also consider total work (the sum over all processors of the running time they use) and overall running time (the elapsed time between the start and end of an algorithm). Overall running time helps us measure how much of the *computational parallelism* present in a sensor network we are able to exploit.

We assume that the isocontour (or the shape) to be approximated is already available to the network. The problem of determining an isocontour from raw sensor data is a well-studied problem, and many (distributed) algorithms are available. An interested reader may consult [3, 15, 17] for various approaches to constructing the contour boundary. Thus, we assume that a subset of the sensors, namely, $s_1, s_2, \ldots, s_n$, collectively stores the detailed representation of the isocontour, and the goal of our algorithm is to build a provable-quality approximation that fits in a given memory size. There has not been significant previous work on this *data reduction* aspect of isocontour construction.

3 Algorithms for Shape Approximation

We assume that the isocontour to be approximated is a polygonal curve embedded in the two-dimensional plane, and a sequence of sensor nodes $s_1, s_2, \ldots, s_n$

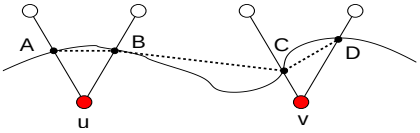

Fig. 1. Boundary estimation from sensor values.

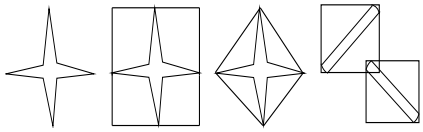

Fig. 2. Low approximation quality using bounding box or convex hull

collectively stores the contour. Specifically, each node s_i stores a consecutive subsequence of the contour polygon so that the concatenation of the chains stored at nodes $s_1, s_2, \ldots, s_n$ results in the full contour. We allow each sensor node to contribute arbitrarily complex portions of the isocontour because, in general, sensors can use complex and collaborative algorithms to compute the contour boundary. As an example (see Fig. 1), the contour detection algorithm may use interpolation to decide that the points A, B, C and D lie on the contour. Points A and B may be stored at node u, while C and D may be stored at node v. In order to keep the presentation of our algorithm simple, however, we will assume that each sensor s_i has only one vertex p_i of the contour. (The location of the contour vertex p_i does not necessarily coincide with the sensor s_i.) However, it will be clear from the description that our algorithm extends easily to the general case where each sensor may store a contiguous portion of the contour boundary.

We assume that adjacent sensor nodes storing the contour boundary are within the communication range of each other; that is, each node s_i is within one hop of s_{i-1} and s_{i+1}. Given a user-specified parameter k, where typically $k \ll n$, we wish to compute a k-vertex approximation of S. Of course, a trivial approach is to communicate all the vertices of S to a central node, and build the approximation there. This scheme, however, has message complexity $\Theta(n^2)$, and we seek more efficient alternatives.

In the following three subsections, we describe contour approximation schemes with which we will compare our new scheme Adaptive-Group-Merge. In Section 3.1, we briefly mention two naïve schemes, which are simple to compute but are too crude to be useful. In Section 3.2, we design a distributed two-dimensional wavelet-based scheme that takes advantage of the signal compression abilities of wavelets. This scheme is easy to implement in the distributed environment of the sensor network, though it lacks good theoretical bounds on the approximation quality. In Section 3.3 we describe a dynamic programming based algorithm that can compute an optimal contour approximation. The dynamic programming algorithm gives optimal approximation, but requires centralized computation, and so is ill-suited for an efficient implementation in sensor networks. It serves, however, as the ultimate benchmark for approximation quality.

3.1 Bounding boxes and convex hulls

One of the simplest representations of any polygon is its *bounding box*, the smallest axis-aligned rectangle containing the polygon. The bounding box of S can be

computed with $O(n)$ message complexity and time. Another simple representation is the convex hull of the polygon vertices, which can be computed exactly with $O(n^2)$ message complexity, or approximated to within any fixed relative error with $O(n)$ message complexity (using an approximation technique due to Dudley [2, 5]). These approximations can be very poor, as shown in Fig. 2: they are too coarse, fail to highlight significant boundary features, and may lose important topological properties—the approximations of widely separated contours may intersect (Fig. 2, right side).

3.2 Wavelets

Wavelet transforms [13] have been used extensively in signal processing, image analysis and database operations. They represent a signal as a linear combination of normalized wavelet basis functions. A wavelet transform takes a one-dimensional signal sampled at n points $\{f_1, f_2, \ldots, f_n\}$ and outputs n coefficients $\{c_1, c_2, \ldots, c_n\}$ for a given set of basis functions. Given a parameter $k < n$, we construct a size-k approximation of the signal by retaining just the k coefficients with largest absolute magnitudes, and truncate the rest to zero. Let $\tilde{\boldsymbol{c}}$ and $\tilde{\boldsymbol{f}}$, respectively, denote the approximate wavelet coefficient vector and the reconstructed signal. Then the L_2 error of the approximation is given by

$$\sum_i (f_i - \tilde{f}_i)^2 = \sum_i (c_i - \tilde{c}_i)^2 = \sum_{i \in \text{truncated}} c_i^2.$$

We now describe a natural way to use wavelets for approximating a polygon embedded in the two-dimensional plane, and a distributed scheme to implement it. Suppose the coordinates of a point p_i are given by (x_i, y_i). We decompose S into two vectors S_x and S_y such that $S_x = (x_1, x_2, \ldots, x_n),\ S_y = (y_1, y_2, \ldots, y_n)$. We carry out independent wavelet transforms on S_x and S_y, and achieve a compact representation of the curve by keeping only the k most important wavelet coefficients. We can implement this computation in a distributed fashion, with every sensor forwarding a single message to its neighbor in the sequence. The message from s_i to s_{i+1}, which has size $O(k + \log i)$ for the specific case of Haar wavelets [6], contains the top k wavelet coefficients of the sequence $p_1, p_2, \ldots, p_i$. Sensor s_{i+1} integrates its own coordinates into the wavelet transform and forwards the new message to s_{i+2} and so on. Sensor s_1 initiates the computation and when the message reaches s_n the algorithm terminates. Summing up the message sizes $\sum_i (k + \log i)$, we see that the total communication complexity of the DISTRIBUTED-WAVELET algorithm is $O(n(k + \log n))$.

Unfortunately, this algorithm does not exploit the parallelism available in the sensor network. In the full version of this paper, we describe a pipelined version of the distributed wavelets algorithm that completes the computation in optimal $O(n)$ time. We state this as a theorem.

Theorem 1. *There is a distributed implementation of the two-dimensional Haar wavelet approximation that takes $O(n)$ time, with total communication complexity $O(nk + n \log n)$.*

Two key disadvantages of the wavelet representation of a polygon are that it tries to minimize L_2 error, rather than the more important Hausdorff error, and it uses a fixed, nonadaptive set of basis functions. In Section 5, we will show some examples where these disadvantages lead to very poor approximations. This motivates us to consider approximation schemes that attempt to minimize the Hausdorff error.

3.3 Optimal approximation using dynamic programming

Our goal is to partition the polygonal curve $S = \{p_1, p_2, \ldots, p_n\}$ into k fragments $S_1, S_2, \ldots, S_k$, with associated approximating line segments $A_1, A_2, \ldots, A_k$. Each fragment consists of a subsequence $\{p_i, p_{i+1}, \ldots, p_j\}$ of S, with consecutive fragments sharing a common vertex. Each fragment and its approximating segment have an associated error value, and the error of a partition is the maximum error over all fragments in the partition. An optimal partition $OPT_k(S)$ is defined as a partition $Q(S)$ such that the error is minimum over all possible partitions of S. If the optimum approximating segment for a fragment depends only on the points of the fragment, then an optimal partition $OPT_k(S)$ can be computed using dynamic programming as follows. Let T be a $k \times n$ table, where $T(\alpha, j)$ contains the optimal (minimum) error for approximating the polygonal curve $\{p_1, p_2, \ldots, p_j\}$ using α segments, where $\alpha \leq k$. We wish to compute $T(k, n)$. The key insight is that the optimal α-segment approximation of $\{p_1, \ldots, p_j\}$ consists of two pieces: the optimal $(\alpha - 1)$-segment approximation of a prefix curve $\{p_1, \ldots, p_i\}$ for some $i < j$, and a single approximating segment for the fragment $\{p_i, p_{i+1}, \ldots, p_j\}$. This leads to the following recurrence:

$$T(\alpha, j) = \min_{1 \leq i < j} \max\left(T(\alpha - 1, i),\ e(i, j)\right), \tag{1}$$

where $e(i, j)$ is the error of the optimum single-segment approximation for $\{p_i, p_{i+1}, \ldots, p_j\}$.

We fill in the entries of T in increasing order of α, and for each α in order of increasing j. Since the table has nk entries and computing each entry using Eqn. 1 takes $O(n)$ time, the dynamic program runs in $O(n^2 k)$ time once the $e(i, j)$ values are known. The general recurrence of Eqn. 1 can be used to compute optimal approximations under several different error metrics. We use the following two in this paper:

1. *Fixed-Segment Error*: A fragment $S_\alpha = \{p_i, \ldots, p_j\}$ is approximated by the line segment $\overline{p_i, p_j}$. The error $e(i, j)$ is defined to be the maximum distance of any point in the fragment from $\overline{p_i, p_j}$, which is nothing but the Hausdorff error.

2. *Floating-Segment Error*: A fragment $S_\alpha = \{p_i, \ldots, p_j\}$ is approximated by the bisector of the *minimum bounding rectangle* (MBR) of the points in the fragment. The error $e(i, j)$, the maximum distance between any point of S and the approximating segment, is half the width of the MBR, which is also the width of S_α.

The floating segment model allows the approximating polygon to use arbitrary points in the plane as vertices, which can potentially improve the approximation quality. However, the approximating segments for neighboring fragments

do not necessarily meet at a common point, and so additional segments may be needed to patch them into a connected polyline.

A third approximation model, which we may call the *Min-Link* model, allows the approximating polygon to use arbitrary vertices (not just vertices of S), but requires the approximating segments for neighboring fragments to share a common vertex. The optimum approximation for the Min-Link model *cannot* be computed by dynamic programming, because the optimum approximating segment for a fragment depends on points outside the fragment. Nevertheless, the optimum k-segment approximation under the floating-segment model has error no larger than the optimum k-segment Min-Link approximation (which has half as many vertices).

4 Adaptive-Group-Merge (AGM): Provable-quality contour approximation

We now describe the main result of this paper: a new, efficient, distributed contour approximation algorithm that delivers a worst-case guarantee on the approximation quality. In particular, we show that whatever approximation quality the optimal (centralized) scheme achieves with k segments, our algorithm is able to achieve that with at most $2k$ segments.

We prove this guarantee using the Floating-Segment model of error, described in the previous section. That is, given an input polyline S, we consider an approximation A consisting of k *possibly-disconnected* segments. The polyline S is partitioned into k polyline *fragments* $S_1, \ldots, S_k$, each associated with an approximating segment A_i. The Hausdorff distance between S_i and A_i is ε_i, and the maximum ε_i over all i is the error ε of the approximation A. Because the segments of A are independent of each other, the error ε_i depends only on S_i. By choosing A_i to be the bisector of the MBR of S_i, we achieve error ε_i equal to half the width of the fragment S_i. (The width of a set is the minimum separation of two parallel lines that sandwich the set between them. The approximating segment A_i lies parallel to and halfway between these lines.)

Let us define the *width* of a partition of S into fragments to be the maximum width of a fragment S_i. Let us denote a partition of S by $Q(S)$, and its width by $\mathit{width}(Q(S))$. We call a partition optimal if it has the minimum width among all partitions of size k and denote it by $OPT_k(S)$.

In order to reason about the approximation quality of a partition, we define the *min-merge property*. A partition $Q(S)$ has the min-merge property if merging any two adjacent fragments results in a fragment with width at least as large as $\mathit{width}(Q(S))$.

One algorithm that produces a partition with the min-merge property is Greedy-Merge: starting with the trivial partition of S into n segments (all fragments with zero width), repeatedly merge the adjacent pair of fragments whose merge product has minimum width, until the partition consists of k fragments. It is easy to prove by induction that this algorithm produces a partition with the min-merge property. Likewise, applying Greedy-Merge to a partition

with the min-merge property preserves the property. However, GREEDY-MERGE is not the only way to produce a partition with the min-merge property, as we will see.

We now argue that *any* partition into $2k$ fragments with the min-merge property has width no greater than that of $OPT_k(S)$.

Lemma 1. *Let $Q(S)$ be a partition of the path S into $2k$ fragments that has the min-merge property. Then $width(OPT_k(S)) \geq width(Q(S))$.*

Proof. Any partition of S into k fragments splits at most $k-1$ fragments of a $2k$ fragment partition. Therefore $Q(S)$ will have at least $k+1$ of its fragments unsplit. By the pigeonhole principle, there exists some fragment S_i of $OPT_k(S)$ that contains at least two unsplit fragments of $Q(S)$. By definition, $width(OPT_k(S)) \geq width(S_i)$, which is in turn at least as large as the the width of the union of the two unsplit fragments. By the min-merge property, this is at least $width(Q(S))$.

The preceding lemma assumes that S is a path, with distinct endpoints p_1 and p_n. If S is in fact a cycle, as in an isocontour application, then the proof can be modified to show that $width(OPT_{k-1}(S)) \geq width(Q(S))$. This difference in approximation quality between paths and cycles is minor, and we will ignore it in the remainder of this paper.

The GREEDY-MERGE algorithm maintains the min-merge property, as noted above. However, implementing GREEDY-MERGE in a distributed setting would require global minimization at each step, and thus would suffer from a serialization bottleneck. We propose an alternative hierarchical merging algorithm, and prove that it also preserves the min-merge property.

In the ADAPTIVE-GROUP-MERGE algorithm we divide the original curve S into n/k *groups*, each with k fragments of size 1 each. The total number of fragments is n. The algorithm proceeds in rounds that reduce the number of groups, maintaining the invariant that each group contains k fragments. In each round we split the current sequence of g groups into $\lfloor g/2 \rfloor$ disjoint pairs of adjacent groups (possibly with one group left over unpaired). For each pair we combine the two groups into one group of $2k$ fragments, then run GREEDY-MERGE on the combined group until the total number of fragments is k. We repeat this for $\log(n/k)$ rounds until the total number of fragments is k.

For this algorithm to work we need to argue that each of the groups it produces has the min-merge property. This is true for the initial groups of segments; the following lemma establishes the fact inductively.

Lemma 2. *Let Q and Q' be two adjacent groups of fragments of S, each containing k fragments and each with the min-merge property. If we apply* GREEDY-MERGE *to the union of Q and Q' until k fragments remain, the resulting group has the min-merge property.*

Proof. Without loss of generality assume $width(Q) \geq width(Q')$. It follows that $width(Q \cup Q') = width(Q)$. As long as the min-merge property does *not* hold, GREEDY-MERGE produces fragments with width less than $width(Q)$.

Thus the min-merge property starts to hold *just before* GREEDY-MERGE first produces a fragment with width at least $width(Q)$. In particular, if GREEDY-MERGE produces a fragment that includes two original fragments of Q, the min-merge property must have held prior to that round of GREEDY-MERGE. After $k+1$ rounds of GREEDY-MERGE, at least $k+2$ fragments of $Q \cup Q'$ are contained inside GREEDY-MERGE products, including at least two fragments from Q. Thus the min-merge property holds after k rounds of GREEDY-MERGE, if not before.

If n is not a multiple of k, at least one of the original fragment groups does not have k members, violating the precondition of Lemma 2. However, this is easy to overcome: to take up the slack we create one group of segments with size s in the range $k \leq s < 2k$, and greedily merge it to size k before the main AGM algorithm begins.

To implement this algorithm in a distributed fashion, we need to keep track of the widths of the new fragments after every merge operation. The simplest way to achieve this is to maintain the convex hull of the points in each fragment [18]. When two neighboring fragments are merged, the convex hull of the resulting fragment is the convex hull of the union of the convex hulls of the old fragments. Thus when a merge operation occurs, the merging fragments need to exchange information about their individual convex hulls.

In the worst case the convex hull of n points can have $\Theta(n)$ vertices. This would give a message complexity of $\Theta(n^2 \log(n/k))$ for AGM, which is more than we would like. Fortunately, we can approximate each convex hull H using only a constant number of points [2, 5], such that the width of the approximation satisfies

$$(1-\delta)width(H) \leq width(approx(H)) \leq width(H)$$

for any desired $0 < \delta < 1$. This degrades the approximation quality of the result by a small relative error, but allows the algorithm to run much faster. (The proof of correctness appears in the full paper.)

Using this convex hull approximation, we can implement the GREEDY-MERGE algorithm on each group of $2k$ fragments by sending all the fragment data (of total size $O(k)$) to a coordinator node within the group, and let it run GREEDY-MERGE locally. If the group encompasses N segments of S, this takes $O(kN)$ message complexity and $O(N)$ time. Summing over all groups in all rounds of the algorithm, we get total message complexity $O(kn \log(n/k))$ and total time $O(n)$. Putting it all together, we have the following theorem:

Theorem 2. *Given an n-vertex polyline stored in a neighbor-connected sequence of sensors, the algorithm* ADAPTIVE-GROUP-MERGE *computes an approximation by k segments whose approximation error is at most $(1+\delta)$ times the error of the optimum approximation by $k/2$ segments, for any $0 < \delta < 1$. The algorithm has total message complexity $O(kn \log(n/k))$ and total running time $O(n)$, with the constant factor dependent on δ.*

The ADAPTIVE-GROUP-MERGE algorithm's approximation of S consists of k disconnected segments: adjacent segments do not necessarily meet. Thus the

output is neither a polygon, nor directly comparable with other schemes like wavelets or Douglas–Peucker, because k disjoint segments require $2k$ vertices to describe the output. Of course, we could simply join each pair of adjacent segments, but that naïve scheme always produces a $(2k-1)$-segment approximation. In practice this may be improved: whenever joining two consecutive segments at the intersection of their supporting lines would not degrade the local approximation quality beyond the worst-case bound for the whole partition, we can omit a connecting segment. Our simulation results (cf. Section 5) show that indeed the size of the resulting polyline approximation remains close to k.

Note that the best polyline approximation by k segments has error at least as large as the error of the best approximation by k disconnected segments. This allows us to convert the Floating-Segment approximation guarantees of this section to bounds in the Fixed-Segment or Min-Link models, with the loss of another worst-case factor of two.

5 Experiments and Results

In this section we experimentally demonstrate the quality of approximations obtained by Distributed-Wavelet and Adaptive-Group-Merge. We use dynamic programming as the optimal reference approximation.

The implementation of Adaptive-Group-Merge computes an approximation by k disconnected segments, then heuristically reduces the number of vertices in the final approximation by linking consecutive segments at the intersection of their supporting lines whenever that does not increase the error of the overall approximation. In our experiments we found that this heuristic reduced the size of the final approximation from $2k$ vertices to around $1.2k$.

Our implementation does not use the Dudley approximation [2, 5] as described in Section 4, because we did not want to introduce another parameter into the experimental setup. We computed the width based on the full convex hulls of the fragments. Using the Dudley approximation might degrade our approximation quality slightly. Interestingly enough, the full hulls were very small in practice—none contained more than eight vertices. This suggests that a practical implementation should be coded with a threshold, so that it uses the Dudley approximation only when the true convex hull has too many points.

We believe that the key measure of approximation quality in practice is the error associated with a given output size. Thus when we compare AGM against other algorithms whose output size is fixed, we choose an input parameter k for AGM that produces the same output size. In a practical setting, the user would specify an input parameter k, with the knowledge that the output will contain slightly more than k vertices. Because the algorithms produce polylines as output, we use dynamic programming with the Fixed-Segment error as our benchmark of quality.

In the following subsections we first compare the approximation performance of the three algorithms, then give a few brief vignettes focused on the approximation behavior of individual algorithms.

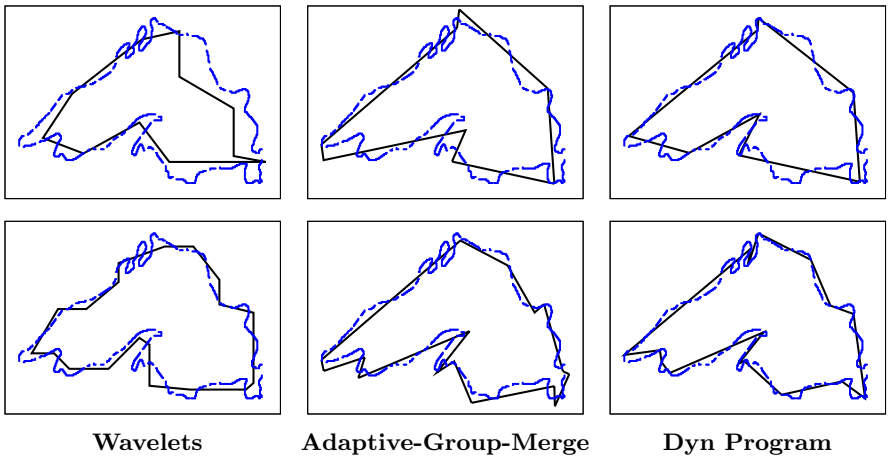

Fig. 3. Approximations for the Lake Superior dataset. The top row shows outputs for $k = 8$, the bottom row for $k = 16$.

5.1 Overall approximation quality

We compare the approximation performance of the algorithms on a GIS data set that digitizes the boundary of Lake Superior into 1024 points. Fig. 3 shows the approximations obtained with $k = 8$ and 16. Because wavelets aim to minimize L_2 error, the wavelet approximations cut off the extreme points and round them out. Adaptive-Group-Merge does better, and the dynamic programming reference algorithm gives the best results, as expected. The trend was similar for other data sets (Lake Huron boundary and Death Valley), and values of k ranging from 8 to 64.

Next we show the approximations obtained by Adaptive-Group-Merge on GIS datasets digitizing the boundaries of India and England into 1383 and 1213 points respectively. Fig. 4 shows the approximations obtained with $k = 32$, 48 and 64 points for both these boundaries. Adaptive-Group-Merge captures these complex boundaries faithfully using a relatively small amount of memory; approximation quality improves as k increases.

Next we evaluate quantitatively the approximations that are obtained by the three schemes. We measure the algorithms using the Hausdorff error and the *relative area error* ε_A we define as follows: if A_{diff} is the area of the symmetric difference between the regions enclosed by the original and approximate curves, and A_S is the area enclosed by the original curve, then $\varepsilon_A = A_{diff}/A_S$. We again work with the Lake Superior dataset to analyze the approximation performance of the three algorithms. In Figs. 5 and 6 we show the Hausdorff and ε_A errors respectively. Adaptive-Group-Merge consistently and significantly outperforms wavelets, and is typically very close to the optimal dynamic programming

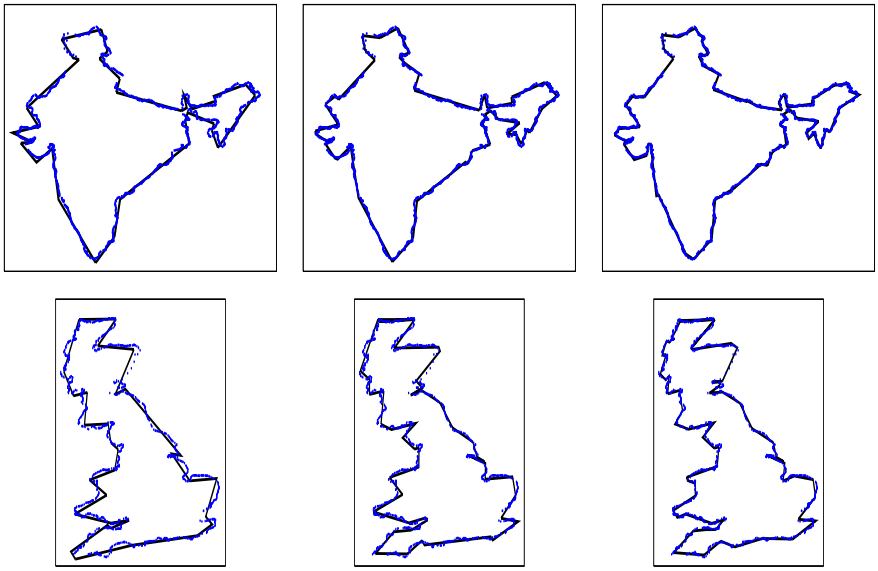

Fig. 4. AGM approximates complex shapes faithfully. The first row shows approximations for the boundary of India, and the second row shows approximations for the boundary of England.

solution. Results for the L_1 and L_2 metrics were intermediate between those for the Hausdorff and ε_A errors.

5.2 Wavelets and the effects of sparse sampling

Because wavelet approximations try to minimize mean squared error instead of maximum error, they can miss some important features, as illustrated in Fig. 7. The figure shows a hand-crafted point set of size 64 and the 8 point approximations obtained by Distributed-Wavelet (Fig. 7(a)) and Adaptive-Group-Merge (Fig. 7(b)). The wavelet approximation tends to weigh all 64 points in the original curve equally, and so large errors for a few extreme points are offset by small errors for the rest of the points. On the other hand the Adaptive-Group-Merge algorithm seeks to minimize maximum error and thus produces a much more acceptable approximation. This shortcoming of wavelet approximations is seen in more realistic data sets as well.

5.3 Approximation quality in theory and practice

Although our primary concern is approximation quality as a function of the number of vertices in a polyline approximation, our provable quality bounds in

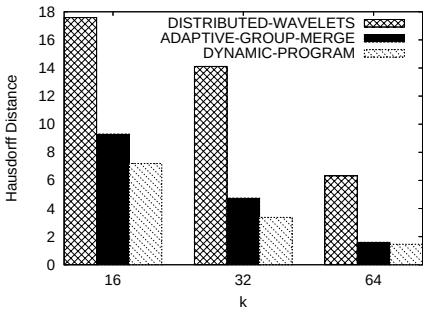

Fig. 5. Hausdorff distances from S to A

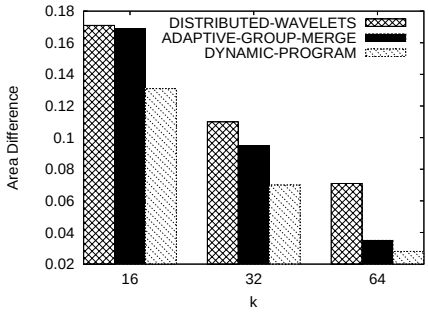

Fig. 6. Fraction of area missed (ε_A)

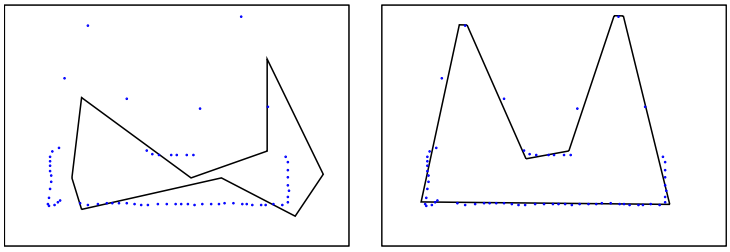

Fig. 7. Bad approximation by wavelets in low density regions

Section 4 use the Floating-Segment error. We compared the Hausdorff errors of Adaptive-Group-Merge and dynamic programming in the Floating-Segment model to see how tight our bounds are. See the table below for the numerical results on the Lake Superior data set. As predicted, Adaptive-Group-Merge results for k segments are somewhat worse than the optimum k-segment approximation, but better than the optimum $(k/2)$-segment approximation.

# segments	Adaptive-Group-Merge	Optimum
8	15.8	13.3
16	7.43	5.14
32	3.27	2.39
64	1.33	1.01

6 Discussion

Reflecting on our simulation results, it seems clear that approximation quality improves markedly as an algorithm pays closer attention to the *geometry* of the shape. The wavelet algorithm uses a "generic" form of compression to reduce the representation size, which tends to treat all vertices the same. This has the virtue of simplicity, but often leads to poor approximation.

We also considered the Douglas–Peucker polygon simplification algorithm [4], which is popular in geographical information systems (GIS), computational geometry, and computer graphics. This is a greedy scheme that starts with a coarse representation (say, the four extreme vertices) then successively refines it by adding a new line segment at each step. At each step, the algorithm adds segments to the vertex that is farthest from the current approximation. In typical GIS applications, the refinement continues until the maximum distance between the approximation and the input polyline drops below a specified threshold. In our setting, the termination occurs when the approximation reaches k vertices.

By design, the Douglas-Peucker scheme has a centralized flavor: at each step, it requires global computation to determine the vertex of the contour that is *farthest* from the current approximation. We have developed *distributed* variants of Douglas-Peucker, but decided to emphasize Adaptive-Group-Merge (AGM) for several reasons. First, much of the simplicity and computational efficiency of Douglas-Peucker is lost in adapting it to a distributed environment. Second, while it generally yields good approximations in practice, Douglas-Peucker does not have a good worst-case theoretical guarantee, while AGM does. And, finally, our experiments showed that distributed AGM produces approximations at least as good as the *centralized* versions of Douglas-Peucker, and hence we expect AGM to be the algorithm of choice in distributed settings.

By design, AGM is well-tailored for distributed environments. The localized nature of AGM allows the algorithm to carry out contour data reduction *independently* at nodes. In particular, if consecutive portions of the contour are available at m different nodes, then each node can reduce the contour size to $O(k)$ through entirely *local processing*, without risking global approximation quality. Thus, AGM may be especially well-suited for heterogeneous multi-tiered architectures like Tenet [7] where clusterhead nodes will aggregate data from nearby motes, and the application software will run only on clusterheads.

AGM guides its approximation by discarding those vertices whose removal leads to least increase in the error, and thus pays close attention to local features of the input—a long sequence of nearly collinear points may get replaced by just the endpoints, while peaks are preserved. We find it encouraging that (1) such a *locally adaptive* scheme yields a worst-case approximation guarantee, which others including wavelets and Douglas–Peucker do not; (2) even though AGM is limited by being tailored to the distributed architecture of sensor networks, it outperforms both wavelets and Douglas–Peucker in the quality of its approximation; and (3) in most cases, AGM performs almost as well as the optimal dynamic programming scheme (which is both centralized and slow).

References

1. N. Bulusu, J. Heidemann, and D. Estrin. GPS-less low cost outdoor optimization for very small devices. *IEEE Personal Communications Magazine*, 7(5):28–34, 2000.

2. T. Chan. Faster core-set constructions and data stream algorithms in fixed dimensions. In *Proc. 20th Annu. ACM Sympos. Comput. Geom.*, pages 152–159, 2004.
3. K. K. Chintalapudi and R. Govindan. Localized edge detection in sensor fields. In *IEEE Intl. Workshop on Sensor Network Protocols and Applications*, pages 59–70, 2003.
4. D. Douglas and T. Peucker. Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. *Canadian Cartographer*, 10(2):112–122, December 1973.
5. R. M. Dudley. Metric entropy of some classes of sets with differentiable boundaries. *J. Approx. Theory*, 10:227–236, 1974.
6. A. C. Gilbert, Y. Kotidis, S. Muthukrishnan, and M. J. Strauss. One-pass wavelet decompositions of data streams. *IEEE Trans. on Knowledge and Data Engineering*, 15(3):541–554, 2003.
7. R. Govindan, E. Kohler, D. Estrin, F. Bian, K. Chintalapudi, O. Gnawali, S. Rangwala, R. Gummadi, and T. Stathopoulos. Tenet: An architecture for tiered embedded networks. Technical report, University of California, Los Angeles, November 10 2005.
8. J. M. Hellerstein, W. Hong, S. Madden, and K. Stanek. Beyond average: Toward sophisticated sensing with queries. In *Information Processing in Sensor Networks: 2nd Intl. Workshop*, pages 63–79. Springer-Verlag, 2003. LNCS 2634.
9. J. Hershberger and J. Snoeyink. Speeding up the Douglas-Peucker line simplification algorithm. In *Proc. 5th Intl. Symp. on Spatial Data Handling*, pages 134–143, 1992.
10. A. Kolesnikov. *Efficient Algorithms for Vectorization and Polygonal Approximation*. PhD thesis, Department of Computer Science, University of Joensuu, 2003.
11. D. Kotz, C. Newport, and C. Elliott. The mistaken axioms of wireless-network research. Technical Report TR2003-467, Dept. of Computer Science, Dartmouth College, July 2003.
12. S. Madden, M.J. Franklin, J. Hellerstein, and W. Hong. Tag: a tiny aggregation service for ad-hoc sensor networks. In *Proc. of OSDI '02*, 2002.
13. S. Mallat. *A Wavelet Tour of Signal Processing*. Academic Press, 1998.
14. R. Moses, D. Krishnamurthy, and R. Patterson. A self-localization method for wireless sensor networks. *EURASIP J. Applied Signal Processing*, 2003(4):348–358, 2003.
15. R. Nowak and U. Mitra. Boundary estimation in sensor networks: Theory and methods. In *Information Processing in Sensor Networks: 2nd Intl. Workshop*, pages 80–95. Springer-Verlag, 2003. LNCS 2634.
16. N. Shrivastava, C. Buragohain, D. Agrawal, and S. Suri. Medians and beyond: New aggregation techniques for sensor networks. In *Proc. of SenSys'04*, 2004.
17. M. Singh, A. Bakshi, and V. K. Prasanna. Constructing topographic maps in networked sensor systems. In *Proc. of Workshop on Algorithms for Wireless and Mobile Networks (ASWAN)*, 2004.
18. G. T. Toussaint. Solving geometric problems with the rotating calipers. In *Proc. IEEE MELECON '83*, pages A10.02/1–4, 1983.
19. M. Yarvis, N. Kushalnagar, H. Singh, A. Rangarajan, Y. Liu, and S. Singh. Exploiting heterogeneity in sensor networks. In *Proc. of IEEE INFOCOM 2005*, 2005.
20. J. Zhao and R. Govindan. Understanding packet delivery performance in dense wireless sensor networks. In *Proc. of the 1st Intl. Conf. on Embedded Networked Sensor Systems, SenSys'03*, pages 1–13, 2003.