Approximate Isocontours and Spatial Summaries for Sensor Networks

Sorabh Gandhi*
Dept. of Computer Science
UC Santa Barbara
Santa Barbara, CA 93106
sorabh@cs.ucsb.edu

John Hershberger
Mentor Graphics
8005 S.W. Boeckman Road
Wilsonville, OR 97070
johnh@wv.mentorg.com

Subhash Suri*
Dept. of Computer Science
UC Santa Barbara
Santa Barbara, CA 93106
suri@cs.ucsb.edu

Abstract

We consider the problem of approximating a family of isocontours in a sensor field with a topologically-equivalent family of simple polygons. Our algorithm is simple and distributed, it gracefully adapts to any user-specified representation size k, and it delivers a worst-case guarantee for the quality of approximation. In particular, we prove that the topology-respecting Hausdorff error in our k-vertex approximation is within a small constant factor of the optimal error possible with $\Theta(k/\log m)$ vertices, where m is the number of contours. Evaluation of the algorithm on real data suggests that the size increase factor in practice is a constant near 2.6, and shows *no* error increase. Our simulation results using a variety of synthetic and real data show that the algorithm smoothly handles complex isocontours, even for representation sizes as small as 32 or 48. Because isocontours are widely used to represent and communicate bi-variate signals, our technique is broadly applicable to in-network aggregation and summarization of spatial data in sensor networks.

Categories and Subject Descriptors: G.1.2 [Numerical Analysis] Approximation of Surfaces and contours. **General Terms:** Algorithms, Theory. **Keywords:** Sensor Networks, Data Aggregation, Approximations.

1. INTRODUCTION

Sensor networks are an attractive architecture for inexpensive and scalable instrumentation of our physical environment, allowing continuous and real-time monitoring of large geographical areas. Current research prototypes have already begun to deliver on the vision of such pervasive sensing: unmanned deployments of sensors providing a continuous or periodic snapshot of a remote environment, including habitat monitoring [20, 27] wildlife dynamics [15], cattle health [21], aquatic observations [5], and surveillance [1, 29].

Many environmental phenomena are "spatial" by nature, in that they are best viewed as a signal landscape or surface. For instance, distributions of temperature, humidity, sound, light, toxin and pollutant levels naturally fit this viewpoint, as do many of the intrinsic attributes of the network's health, such as energy levels of nodes, traffic congestion, wireless link quality, etc. An application-level challenge in sensor networks is to convey a faithful representation of this signal to the user, while utilizing minimum possible network resources. Indeed, the functional lifetime of the network is determined by the energy reserve of its nodes, and wireless communication is the largest consumer of power. Thus, algorithms that can significantly reduce the data flow in the network are highly desirable.

Motivated by these considerations, there has been a significant interest within the sensor network research community in data aggregation methods, and a variety of techniques have been proposed for estimating statistical attributes of sensor data, including min, max, average [19, 31] as well as robust statistics such as median or quantiles [10, 24]. However, most of these techniques have focused on *numerical* statistics, not the *spatial* shape or form of the signal landscape. Neither average nor median (nor even quantiles) tells the user the *geometric shape* of the phenomenon being observed by the sensor field—is it smooth or does it have distinct peaks; are the peaks sharp or flat; are they clustered in a small region or scattered widely, etc. A geometric summary, on the other hand, can help the user *reconstruct* an approximation of the remote signal field and gain a much more detailed insight into its shape and form.

In this paper, we consider a scheme that allows the user to control the fidelity of his reconstruction through a single parameter k, denoting the summary size. In particular, suppose the signal landscape in the sensor field is represented as a family of isocontours. (Isocontours are both a simple and popular method for discretizing a two-dimensional surface.) Our goal is to (distributedly) compute an approximation of this contour family using at most k vertices, while minimizing the error in the approximation. The parameter k limits the size of the *data packets* exchanged by the sensor nodes, hence controlling the total transmission cost. In our scheme, each node will send at most a constant number of size-k packets to construct the contour map.

*The research of Sorabh Gandhi and Subhash Suri was supported in part by National Science Foundation grants CNS-0626954 and CCF-0514738.

IPSN'07, April 25-27, 2007, Cambridge, Massachusetts, USA.

1.1 Problem Description and Challenges

Formally, we consider the following problem: Given a family of polygonal isocontours $C_1, C_2, \ldots, C_m$ and a user-specified parameter k, we want to approximate the isocontours with polygons $P_1, P_2, \ldots, P_m$ using at most k vertices. Our goal is to *minimize* the maximum (Hausdorff) error between our approximation and the original contours, while preserving the topological structure of the contour family. That is, our approximate contours maintain the same nesting order as the original, without intersections among different approximations. We assume that the original contour data is distributed across sensors (each sensor has only some local portion of the contour family), and a communication protocol (e.g., a query-aggregation tree) is used to build the approximate contour map in a distributed fashion. We want a communication-efficient algorithm that is significantly better than each node sending its data to a base station through multi-hop routing.

The key challenges associated with the problem are:

- The user specifies only the global parameter value k—the (distributed) algorithm, which does not have all the data at one node, must decide how to spend this memory on different contours in the family to achieve small overall approximation error.
- Given a certain size, we know how to approximate one contour to get the best error, but the approximation of one contour can intersect (cross) approximations of other contours, violating the topological structure. Thus, we lose the independence of the contour approximations, which in turn complicates the error analysis. The challenge is to preserve the input contours' topology while computing a bounded-error approximation.
- When measuring the quality of an approximation, the maximum error is a natural metric—large local errors can hide important features of the data. However, since different contours can have widely different sizes, the user may prefer the error to be measured relative to the contour size, in which case a suitable normalizing factor for the relative error must be introduced. That is, we need to find a suitable function of the contour area, perimeter, or both to scale each contour's approximation error.

1.2 Our Contributions

Our main contribution is a distributed algorithm with provable error guarantees for approximating a family of contours. In particular, suppose an optimal algorithm can approximate the contour family within maximum error ε using k vertices. We propose a two-phase distributed algorithm in which the first (the main) phase returns an $O(k)$ size polygon family with approximation error $O(\varepsilon)$. A second (post-processing) phase is needed if the polygons in the approximating family intersect each other. Our proposed scheme performs the *untangling* with bounded increase in approximation error, but may introduce additional vertices in the worst case. We conjecture that the increase in the size of the approximation is at worst linear, but at present we can prove only a bound of $O(k \log m)$, where m is the number of contours. Our extensive simulation results support our conjecture: we observe an increase of less than 10% in the worst case. To place this result in context, we point out that even the problem of deciding whether a contour can be approximated by a simple (non-self-intersecting) polygon of k vertices for a given error is known to be NP-complete [11].

Given the intractability of the problem, we use a lower bound based on dynamic programming to argue the near-optimality of our scheme. The dynamic programming solution finds a k-segment approximation with minimum error, but using k segments that are not necessarily linked into a continuous polygon[1]. A continuous k-segment approximation, like the one we produce, clearly can have no better error than the less-constrained dynamic programming approximation.

Simulation results using a variety of synthetic and real data show that our algorithm smoothly handles complex isocontours, and even on modest size contour maps (with 2000 to 6000 points), it delivers compression factors of 20 to 100, without losing any important features. The simulation also suggests that both the error and the size bounds implied by our worst-case analysis are extremely pessimistic: in experiments, the algorithm yields error no worse than that of the best k-vertex approximation using roughly $2.6 \times k$ vertices. When compared to the rectangle-simplification scheme proposed by Hellerstein et al. [12], our algorithm gives noticeably better results both in visual appearance and measured error.

2. RELATED WORK

Polygon simplification is a fundamental problem in geographic information systems (GIS), computational geometry and computer vision. The two main variants of the problem are the minimum segment approximation (given a fixed error) and the minimum error approximation (given a fixed number of segments). A popular approximation scheme for simplifying a polyline is the Douglas-Peucker algorithm [6, 13], which greedily prunes the input to a subset of the original vertices (that is, the approximation is not allowed to use non-input points, often called Steiner points). The Douglas-Peucker algorithm lacks good worst-case guarantees, but remains popular due to its simplicity. It was shown in [2] that a greedy merge scheme yields a better polygon approximation than Douglas-Peucker, providing both a worst-case guarantee and a distributed implementation better-suited to sensor networks.

On the other hand, if Steiner points are permitted in the approximation (as is the case in our setting), then even approximating a single contour (homotopy type) optimally is NP-hard [11]. A good survey of various polygon and subdivision approximations can be found in [16]. For isocontours, Estkowski et al. [7] show that the minimum segment approximation with no Steiner vertices is NP-hard to approximate within a factor $n^{\frac{1}{5}-\delta}$ of optimal.

In the sensor network community, the contour detection and simplification problem has received a lot of attention. Many boundary (physical or data space) detection algorithms [3, 8, 17, 18, 23, 25, 28] have been proposed, in

[1]Our dynamic program partitions contours into disjoint fragments and approximates each separately. This potentially gives a less-than-optimal Hausdorff approximation, because the optimal Hausdorff approximation is not constrained to match subcontours with unique approximating segments. However, since topology preservation is one of our goals, this pairing of subcontours with their approximating segments seems desirable.

which each sensor detects and stores a part of the boundary. None of the distributed heuristics proposed for compressing the boundary representation of the contours [12, 26, 32] are able to provide worst-case approximation guarantees, with the exception of [2], which is valid only for a single polygon. There are other schemes [22, 30] that detect and compress these contours (temporally and/or spatially), but no formal analysis is available either for their worst-case approximation performance or their topology preservation.

3. ISOCONTOURS AND SPATIAL SUMMARIES IN SENSOR NETWORKS

We consider a problem setting in which a network of sensors measures a spatially distributed phenomenon in the plane. The precise nature of the phenomenon and the details of the sensing are not relevant to our work, as long as we can view the sensed data as a bi-variate function $S(x, y)$, where $S(x, y)$ is the value of the function at location (x, y). This abstract viewpoint is sufficient for a broad range of sensing applications, including environmental monitoring of temperature, humidity, sound, light, toxin levels, etc., as well as the monitoring of network-centric properties such as network congestion, wireless link quality, etc.

We assume further that the surface S is available to us as a set of discrete isocontours—there are several algorithms available for identifying contour lines in a field (see Section 2). *The focus of our work is to compute a compact representation of these data.* We make no assumptions about the shape of the contour map or the density of the nodes, except that the set of isocontours is stored distributed among the sensor nodes: each node may contain a bounded-size portion of the (local) contour map.

We assume that the sensor nodes use an underlying communication protocol, perhaps using a data-aggregation tree to combine these fragments of the contours into a bounded size approximation of the entire map. We present our algorithm at a high enough level that specific details of this protocol and the network architecture are not critical. Nevertheless, the algorithm is well-suited to both Berkeley's SNA [4] and USC's Tenet [9] architectures. The only primitive we require is for a node to receive the contour summaries from other nodes (say, its children) and then *merge* those summaries along with its own portion of the contour map into a new summary, while maintaining local optimality and topological consistency.

3.1 Problem Statement

The input to our algorithm is a set of m polygonal contours, distributed in a sensor field. We let $|C_i|$ denote the number of vertices in contour C_i, and $N = \sum_i |C_i|$ denote the total size of the input contour family. (Thus, the input size for our problem is determined by the complexity of the contours, and not the number of sensors. Typically, we would expect that the number of sensors involved in contour approximation is smaller than N.) We make no assumption about the shape or size of any contour, except that each C_i is a simple (non-self-intersecting) closed curve, and no two contours intersect—any two contours either have disjoint interiors or one contains the other. Therefore, there is a unique nesting order among the contours, which we call the *topological structure* of the map. Our approximation *must respect* this structure: the output is a set of piecewise linear contours (polygons) $P_1, P_2, \ldots, P_m$.

We use the Hausdorff error metric to measure the quality of approximations. The Hausdorff error between two polylines is the maximum distance of a point on either polyline from the other polyline. Let $d(p, Q)$ be the minimum Euclidean distance of a point p from a polyline Q. Then, for polylines $P = (p_1, p_2, \ldots, p_{n'})$ and $Q = (q_1, q_2, \ldots, q_{n''})$, where p_i and q_i are the vertices defining the segments of P and Q respectively, the Hausdorff error can be formulated as

$$H(P, Q) \equiv \max(\max_{0 \le i < n'} d(p_i, Q), \max_{0 \le j < n''} d(q_j, P)).$$

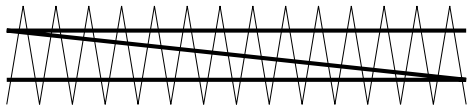


Figure 1: An approximation for the accordion-fold contour that minimizes the Hausdorff distance does not preserve the left-to-right linear shape of the contour at all.

An obvious goal for an approximation P of a contour C is to minimize $H(P, C)$. An *unconstrained* minimization of Hausdorff distance, however, does not preserve the shape well. Figure 1 shows an example where the input contour is a shaped like a folded accordion. If we were to seek a k segment approximation to this accordion, minimizing the Hausdorff error, then we get a k-segment chain crossing the accordion left-to-right k times. This yields a maximum error of about d/k, where d is the vertical height of the accordion. Clearly, however, this is a poor approximation of the input shape; a natural approximation should traverse the accordion *once* from left-to-right (yielding maximum error about $d/2$). In this work, therefore, we choose an approximation strategy more suited to shape preservation. In particular, we partition each contour into a sequence of contour fragments, splitting at the vertices, and associate each segment of an approximation with a single contour fragment. The cyclic order of the approximation segments matches that of the contour fragments. For each approximate segment we measure its Hausdorff distance to its associated fragment and vice versa. The maximum over all segments and fragments is the error of the approximation. For the isocontour approximation, the error is the maximum of all the errors for individual contours and their approximations; we denote this error by ε.

4. THE APPROXIMATION ALGORITHM

Our approximation algorithm consists of two phases: the *compression phase*, followed by the *untangling phase*. The compression phase reduces the total size of all contours to $O(k)$, but the resulting contour cycles may intersect, thus violating the desired topological structure. The untangling phase removes the intersections while keeping the approximation error within a constant of the optimal.

The compression phase itself has two parts, which we call *stick-fitting* and *cycle-formation*. The first part, stick-fitting, performs a crude approximation in which each con-

tour is partitioned into contiguous fragments, and each fragment is approximated by a single line segment (called its *stick*). The stick is the midline of a minimum-width rectangle that completely contains the contour fragment being approximated. The union of these rectangles covers all the contours in the map, and the maximum error in the approximation is half the *width* of the widest rectangle. The stick-fitting phase aims to cover the contour map using at most k rectangles with the minimum width possible. However, the sticks that form the central axes of these rectangles do not necessarily meet end-to-end, and so the second phase (cycle formation) is invoked to add new segments joining consecutive sticks without increasing the approximation error. The untangling phase resolves intersections between the cycles thus formed. In the following sections, we discuss each of these components and present their analysis.

4.1 The Compression Phase

The stick-fitting phase is the only hierarchical, distributed portion of our algorithm. It produces an approximation containing k possibly-disconnected segments; the subsequent phases, performed at a single node or base station, form cycles and untangle them. Stick-fitting is performed at a node when the total size (number of edges) of the approximated contour map (received at that node) exceeds k. For instance, during the upward data flow in an aggregation tree, the total size of all the children's summaries will typically exceed k and the parent node will perform stick-fitting. Stick-fitting makes no assumptions about the input sequence of contours: each contour could have multiple disjoint pieces, and some contours may be completely missing. The only requirement is that each input summary either contains at most k segments of the original contours, or it contains k fragments constructed by stick-fitting.

4.1.1 Stick-Fitting

The stick-fitting phase decomposes the contours into k fragments, and approximates each fragment with a stick (line segment) so that all points of a fragment are within distance ε of its associated stick. This is a relaxed version of our problem since it does not require that the sticks for each contour form a connected polyline. Clearly, though, the optimal error achieved using this relaxation is a lower bound on the error we can obtain in our approximation.

Every node that has a portion of the contour boundaries reduces the number of fragments in its approximation to k before sending it to its parent in the aggregation tree. If the node has k' fragments that it needs to reduce to k, it repeats the following step $k' - k$ times: merge the pair of adjacent contour fragments whose merged fragment has minimum width. This extends the scheme of [2] to multiple contours. The analysis of that paper applies to prove the following lemma[2]:

LEMMA 1. *If the stick-fitting algorithm approximates a single contour using at least $2k_i + 1$ sticks, then the error of the approximation is no larger than the error of an optimal decomposition into k_i fragments.*

[2]A slight extension of the analysis is necessary to handle the case when a set of fewer than k fragments (necessarily all original contour segments) is merged with a set of k fragments.

4.1.2 Stick-Fitting Analysis for the Contour Family

In the stick-fitting problem, we are not concerned with the topological constraints of the different contours. Thus, it is possible to devise an optimal polynomial time algorithm using dynamic programming. The dynamic programming algorithm is centralized and computationally inefficient, but we use it only for error analysis. We will show that the maximum error of our hierarchical scheme of merging adjacent fragments is within a constant factor of this dynamic program's solution.

To construct an optimal stick-fitting approximation for a set of contours, we first use dynamic programming [2] to compute the optimal approximations *for each contour C_i of all sizes from 1 to k*. Taking single-segment approximations for each C_i gives an m-segment approximation for the whole set. To increase the approximation size from m to k, we repeatedly select the contour whose approximation has maximum error and increase its approximation size by one (the dynamic programming precomputation tells us the optimal approximation error for the increased size). This process produces a k-stick approximation with minimum error. The following lemma bounds the error of the stick-fitting algorithm.

LEMMA 2. *If the stick-fitting algorithm approximates all m isocontours using at least $2k + m$ sticks, then the error of the approximation is no more than the error of an optimal decomposition using k sticks.*

PROOF. Let us suppose that the dynamic programming algorithm allots k_i sticks to represent contour C_i, with $\sum_{1 \le i \le m} k_i = k$. The error associated with contour C_i is $\varepsilon_D(k_i)$; the overall dynamic programming error is $\varepsilon(D, k) = \max_i \varepsilon_D(k_i)$. Similarly, suppose that stick-fitting allots k'_i sticks to contour C_i, with $\sum_{1 \le i \le m} k'_i \ge 2k + m$. The stick-fitting error for C_i is $\varepsilon_S(k'_i)$, with overall stick-fitting error $\varepsilon(S, 2k + m) = \max_i \varepsilon_S(k'_i)$.

We need to show that $\varepsilon(S, 2k + m) \le \varepsilon(D, k)$. We assume

$$\varepsilon(S, 2k + m) > \varepsilon(D, k) \tag{1}$$

and prove the result by contradiction. For Equation 1 to hold, Lemma 1 implies that there must be at least one contour, say C_u, for which the number of segments k'_u is strictly less than $2k_u + 1$. Without loss of generality, let us assume that this corresponds to the maximum error in the approximation, that is,

$$\varepsilon(S, 2k + m) = \varepsilon_S(k'_u) \tag{2}$$

Consider the point in the merging process when C_u had exactly $k'_u + 1$ fragments and the stick-fitting algorithm picked two of the fragments from C_u to merge and left it with k'_u fragments. By the pigeonhole principle, there must have been at least one contour at that point, say v, that had strictly more than $2k_v + 1$ fragments. Let the number of fragments in v at that time be k''_v, where $k''_v > 2k_v + 1$. Since these fragments are formed using the merge operation,

$$\varepsilon_S(k''_v - 1) \le \varepsilon_D(k_v) \le \varepsilon(D, k) \tag{3}$$

For the stick-fitting algorithm to pick C_u for merging ahead of C_v, we must have

$$\varepsilon_S(k''_v - 1) \ge \varepsilon_S(k'_u) \tag{4}$$

This equation comes from the fact that merging two fragments of contour C_v leads to more error than merging two

fragments of C_u (when they have k''_v and k'_u+1 segments respectively), hence two fragments of C_u are merged, leaving it with k'_u fragments.

From equations 3 and 4 and the fact that $\varepsilon(S, 2k+m) = \varepsilon_S(k'_u)$ (Equation 2), we get

$$\varepsilon(D,k) \geq \varepsilon(S, 2k+m) \tag{5}$$

This contradicts the assumption made in Equation 1 and therefore proves our result. □

4.1.3 Cycle-Forming

After the stick-fitting phase is done, we are left with a set of k sticks that approximate the given set of contours with maximum error ε. We now show how to form closed cycles using these sticks while increasing the error by at most a factor of two.

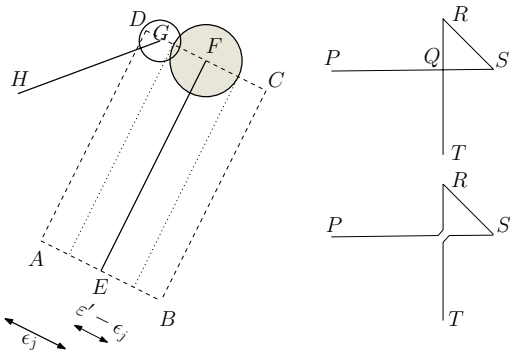


Figure 2: Cycle-forming (left) and simple closed curves (right)

Let the set of errors associated with the approximation segments (sticks) of some contour C_i be $\epsilon_1, \epsilon_2, \ldots, \epsilon_{k_i}$, where k_i is the number of sticks. Let the maximum of these individual fragment error values be $\varepsilon' \leq \varepsilon$.

While forming a connected chain we exploit the fact that not every stick has ε' error associated with it. As an example, consider a stick $\overline{EF}$ of the approximation as shown in Figure 2 and let the error associated with it be ϵ_j. Let the bounding rectangle corresponding to $\overline{EF}$ be $ABCD$ as shown in the figure; hence the length of $\overline{AB}$ and $\overline{CD}$ is $2\epsilon_j$. It is not difficult to see that any point E' that is within distance $\varepsilon' - \epsilon_j$ of E can be used to replace E without increasing the overall error of the approximation. The loci of all such points form disks around end vertices. If the disks corresponding to two consecutive sticks intersect, then any intersection point can be used to replace the two end points without increasing the error to more than ε'

In the left part of Figure 2, $\overline{GH}$ is the next stick. The disks corresponding to F and G intersect, and hence any point in the intersection region can be used to replace both F and G. However, if these circles do not intersect, we add two segments from F and G to the common vertex of the fragments corresponding to $\overline{EF}$ and $\overline{GH}$. These segments lie inside the bounding rectangles of the fragments, and hence each lies within distance 2ε of its corresponding fragment.

The polygons produced by stick-fitting and cycle-forming are not necessarily simple: that is, they may self-intersect. To resolve these self-intersections while preserving the error guarantee, we break crossing segments at their intersections and reconnect them into a simple, Eulerian tour. Figure 2 (right) shows a simple case. In general we remove self-intersections as follows: embed the polygon in the plane; break every segment of the polygon at all its crossings; create a simple polygon by traversing the outer face of the embedded polygon; erase all the (sub)segments of this outer polygon; recursively compute a simple polygon for each component of edges that remains (erasing the outer polygon leaves all vertex degrees even, so the recursion is possible); and finally hook each inner polygon to the outer one by performing an edge reconnection as in the figure. This process enforces simplicity and preserves the error guarantee, but it increases the number of segments in the approximation. However, as our experiments show (Section 5), this procedure, while theoretically required, is rarely if ever needed in practice.

4.2 Untangling Phase

After the compression phase each contour is approximated by a simple closed polygon. These approximating polygons, however, might intersect each other, violating the topological structure of the original contours. Therefore, we now show how to resolve these intersections, with only a constant factor error increase in the worst case. In particular, we argue that when two approximate contours intersect, we can use part of one to approximate a portion of the other, resolving the intersection while maintaining a guaranteed worst-case error.

4.2.1 Untangling Phase Analysis

Recall that every contour is partitioned into fragments by the stick-fitting algorithm. Every fragment is associated with a minimum-width rectangle that encloses the whole fragment. We call such a rectangle an *MBR*, short for Minimum Bounding Rectangle. The following lemma characterizes the relationships between the MBRs and the original contour.

LEMMA 3. *Any input contour C_i lies completely inside the set of MBRs associated with its approximation, and every MBR intersects at least two other MBRs.*

PROOF. The MBR corresponding to a fragment contains all the fragment's vertices. Hence every segment of C_i lies inside some MBR. Every pair of consecutive fragments shares a common vertex. The common vertex must be inside or on the boundary of the two corresponding MBRs, so those two MBRs must intersect. There are two such common vertices associated with every MBR, establishing the second part of the lemma. □

Consider two intersecting contour approximations P and Q, and let the contours associated with them be $C(P)$ and $C(Q)$ respectively. We assume for simplicity that the approximations intersect with each other at two points; multiple intersections can be handled by repeatedly applying the procedure described below. If there are multiple intersections, one can show by induction that there must be a *lens*—two intersections that are consecutive along both P and Q. Untangling the lens reduces the number of crossings by two, so we can remove all the intersections recursively.

Suppose that the input contours are nested, with $C(P)$ enclosing $C(Q)$. (Similar arguments work if $C(Q)$ encloses $C(P)$, or neither encloses the other.) Let the two intersection points be p_a and p_b. Let the portion of P between

p_a and p_b be P' and let the corresponding part of Q be Q'. The Jordan curve theorem [14] states that *any simple closed curve in the plane separates the plane into two disjoint regions, the inside and the outside.* Using this we can define Q' as the part of Q outside P and P' as the part of P inside Q. We use this notion of inside and outside throughout the analysis.

Let us define $H'(U,V)$ to be the one-sided Hausdorff distance between the two polylines U and V. That is,

$$H'(U,V) \equiv \max_{p \in U} d(p,V),$$

where U and V include the segments of the two polylines, not just their vertices, and $d(p,V)$ is the minimum distance from the point p to the polyline V.

We now prove a key lemma concerning the distance between improperly nested approximations and the contours they represent.

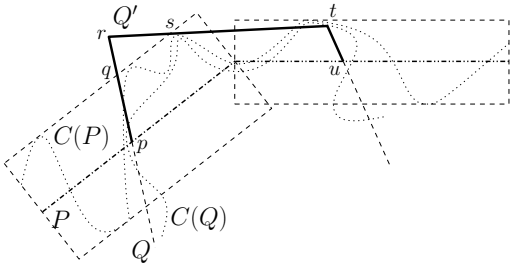


Figure 3: Illustration for Lemma 4 showing Q', $C(Q)$, and parts of P and $C(P)$.

LEMMA 4. *Let P and Q be two intersecting approximations, with P' and Q' as defined above. Then*

$$H'(Q', C(P)) \leq 4\varepsilon.$$

PROOF. Because both the contours and their approximations lie completely inside the MBRs associated with the approximations, by Lemma 3, we have the following inequality:

$$H'(P, C(P)) \leq 2\varepsilon. \tag{6}$$

The same argument also implies that $H'(C(P), P) \leq 2\varepsilon$.

Let us denote the set of minimum bounding rectangles of P by $MBR(P)$. The main part of the lemma is to prove that no point on Q' is more than a constant times ε away from the $MBR(P)$. For illustration, Figure 3 shows an intersection between P and Q, with Q' being represented by (p, q, r, s, t, u).

We need to bound the distance from $C(P)$ of all those points of Q' that are outside $MBR(P)$ (those inside are certainly within 2ε of $C(P)$). A portion of Q' can be outside $MBR(P)$ only when it is also outside P (otherwise, by definition it is not part of Q'). In Figure 3, this corresponds to polyline qrs; let us denote this portion by Q'_{out}. We know that $C(P)$ lies inside $MBR(P)$ (from Lemma 3), and $C(Q)$ lies inside $C(P)$. Hence, no portion of $C(Q)$ goes outside both P and $MBR(P)$. No point on Q'_{out} is farther than 2ε from $C(Q)$. Every point on Q'_{out} is closer to $MBR(P)$ than it is to $C(Q)$. So we have

$$H'(Q', MBR(P)) \leq 2\varepsilon. \tag{7}$$

But every point on $MBR(P)$ is within 2ε of $C(P)$ (Lemma 3). Hence

$$H'(MBR(P), C(P)) \leq 2\varepsilon. \tag{8}$$

Applying the triangle inequality to Equations 7 and 8 gives the desired result. □

This lemma shows that the portion of one approximation between two intersection points can be approximated by some portion of the contour corresponding to the other approximation. Let us define $C(Q')$ to be the portion of $C(Q)$ being approximated by Q'. Any fragment of $C(Q)$ whose stick is fully contained in Q' belongs to $C(Q')$; segments of Q that are only partially in Q' may lead to partial fragments in $C(Q')$. That is, if $nn(p,U)$ is the nearest neighbor of a point p in the set U, F is a fragment of $C(Q)$ whose stick is only partially in Q', and Q_F is the set of up to three segments of Q that approximate F, then we add to $C(Q')$ the partial fragment $\{q \in F \mid nn(q, Q_F) \in Q'\}$. Note that $C(Q')$ may be discontinuous in the first and last MBRs associated with Q'. We would like to be able to claim that

$$H(P', C(Q')) \leq c\varepsilon$$

for some constant c, but counterexamples show that this is false. We need something more than P' to approximate $C(Q')$. The fact that $H'(Q', C(P)) \leq 4\varepsilon$ suggests that part of P might be used to approximate $C(Q')$.

We now explain the procedure used to determine $P(Q')$, the part of P that is used to approximate $C(Q')$ (it includes P', if nothing else). Recall that all contours and approximations are provided to us in cyclic order. Extending the nearest neighbor function to sets, we define $nn(Q', P) \equiv \cup_{q \in Q'} nn(q, P)$ as the portion of P closest to Q'. This is a possibly disconnected subset of P. Suppose it consists of j disconnected pieces. We form a connected polyline by adding into $nn(Q', P)$ the $j-1$ fragments of $P \setminus nn(Q', P)$ closest to P' in cyclic order. This connected polyline is $P(Q')$.

In the following two lemmas we show that the Hausdorff distance (two-sided) between $P(Q')$ and $C(Q')$ is at most a constant times ε.

LEMMA 5. *If $C(Q')$ is the portion of $C(Q)$ approximated by Q', and $P(Q')$ is the polyline defined above, then*

$$H'(C(Q'), P(Q')) \leq c\varepsilon$$

for some constant c.

PROOF. From Lemma 4, we know that $H'(Q', C(P)) \leq 4\varepsilon$. Since $H'(C(P), P) \leq \varepsilon$, the triangle inequality implies $H'(Q', P) \leq 5\varepsilon$. But $P(Q')$ is defined by proximity to Q', so $H'(Q', P(Q')) \leq 5\varepsilon$. By definition $H'(C(Q'), Q') \leq \varepsilon$, so one more application of the triangle inequality gives $H'(C(Q'), P(Q')) \leq 6\varepsilon$. □

LEMMA 6. *If $C(Q')$ is the portion of $C(Q)$ approximated by Q', and $P(Q')$ is the polyline defined above, then*

$$H'(P(Q'), C(Q')) \leq c\varepsilon$$

for some constant c.

PROOF. Consider the four polylines shown in Figure 4. The figure shows parts of P (it is showing $P(Q')$), Q (showing Q'), $C(P)$ (shows the part of $C(P)$ approximated by

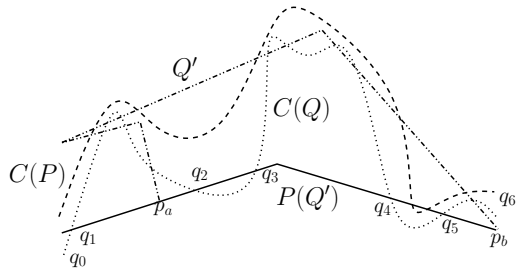


Figure 4: $P(Q')$ shown with Q' , $C(Q')$ and part of $C(P)$

$P(Q'))$ and $C(Q)$ (showing $C(Q')$). The figure also shows the intersection points p_a and p_b; these are the beginning and end points of Q' respectively. Let $I(P)$ denote the interior of the polygon P. $C(Q')$ can be divided into portions inside and outside P. In the figure $C(Q') \cap I(P)$ consists of the portions of $C(Q')$ between q_0-q_1, q_2-q_3 and q_4-q_5, and $C(Q') \setminus I(P)$ consists of the portions of $C(Q)$ between q_1-q_2, q_3-q_4 and q_5-q_6.

Consider the parts of $P(Q')$ that join the endpoints of individual fragments of $C(Q') \cap I(P)$; clearly these parts of $P(Q')$ are closer to $C(Q') \cap I(P)$ than those parts of Q' that are actually approximating it. Hence this portion of $P(Q')$ is within $c_1\varepsilon$ of $C(Q')$, for some constant c_1. Now consider $C(Q') \setminus I(P)$ and the corresponding parts of $P(Q')$ that approximate it. Every point on such a part of $P(Q')$ is closer to $C(Q') \setminus I(P)$ than to any point on $C(P)$ (which $P(Q')$ is approximating). Hence these parts of $P(Q')$ are also within a distance of $c_2\varepsilon$ of $C(Q') \setminus I(P)$, where c_2 is another constant. Hence every part of $P(Q')$ is within a distance of $c\varepsilon$ of $C(Q')$. □

The scheme outlined above uses a part of the outer polygon P to approximate the inner polygon Q while resolving the intersections. The same guarantees can be obtained while resolving the intersections the other way, i.e., using a part of Q to approximate the improperly nested part of P.

The analysis of how much the untangling phase increases the size of the approximation is complicated. If P is treated as fixed, and portions of P are used to replace improperly nested parts of Q, then the size of Q may increase by $O(|P|)$. Untangling a set of m contours in worst-case order may increase the total approximation size to $O(mk)$. However, by careful choice of the untangling order (first fix the smallest contour in the middle third of the nesting hierarchy, then recurse), we can reduce the worst-case cost of untangling to $O(k \log m)$. We conjecture that a more careful untangling increases the size by at most constant factor, and are currently working to prove that.

Thus, for the compression phase of our algorithm, we have a worst-case guarantee of $O(k)$ space and $O(\varepsilon)$ error. For the untangling phase, however, we currently have only the $O(k \log m)$ size bound and $O(\varepsilon)$ error. Our simulation results (next section) show the worst-case analysis is highly pessimistic, and the untangling algorithm performs much better in practice, never increasing the size by more than 10%.

Due to lack of space, we omit further details of this analysis and summarize our main theoretical result.

THEOREM 1. *Given a family of m contours and a size parameter k, our algorithm produces a set of approximate (possibly intersecting) contours of total size $O(k)$, whose maximum error is within a constant factor of the k-segment optimal dynamic program approximation. If the approximate polygons intersect each other, an untangling algorithm is used to remove any intersections and restore topological consistency. We prove that the worst-case increase in the size of the approximation due to untangling is $O(k \log m)$, but conjecture that it is only $O(k)$.*

5. EXPERIMENTAL EVALUATION

In this section, we report on simulation results for our new algorithm, which we call TOPOLOGY-SENSITIVE-COMPRESS. We use various digitized contour boundary datasets in our experiments, as well as synthesized residual energy datasets. The datasets vary in size from 2000 points to 6000 points. All simulations are done in C++.

The stick-fitting algorithm is implemented exactly as described in Section 4.1.1, and the cycle-formation scheme (Section 4.1.3) is used to form simple closed chains. The (possibly) intersecting chains are then resolved using the untangling algorithm; the approximations for inner contours are used to replace improperly nested portions of outer approximations. All k values shown in the experiments are the final sizes produced, unless specified otherwise.

5.1 Approximation Quality

We first discuss the quality of approximations produced by our algorithm for various datasets, and its dependence on the size parameter k. We also discuss the empirical effectiveness of the untangling phase.

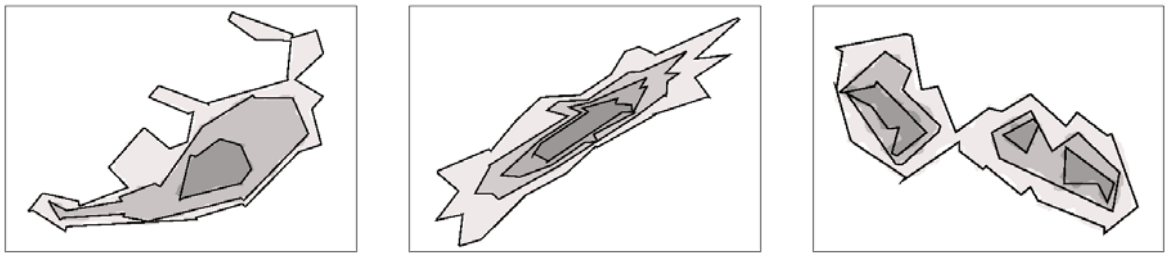
5.1.1 Adaptation to Features

We begin with an experiment to illustrate the general effectiveness of the algorithm in approximating a wide and complex range of contour shapes. Due to limited space, we show only three representative pictures in Figure 5. We chose these examples because of their rich contour structures as well as the appropriateness of their data to potential sensornet applications. The leftmost figure (labeled *windspeed*) shows a contour map of windspeed values over the southeastern United States (the input uses 3500 contour segments). The middle figure (labeled *noise*) shows the noise contours around the Cleveland Hopkins International Airport using 5000 segments. The rightmost (labeled *eScan*) is a synthetic data set generated to mimic the energy depletion in sensor networks, using the eScan model proposed by [32]. In particular, the eScan dataset was synthesized using three hotspots in a network of 1500 sensor nodes. This dataset has 6000 segments defining its boundary (after smoothing the contours using anti-aliasing).

In all three figures, the size parameter for the approximation map is $k = 64$, resulting in a compression factor of between 50 and 100. Still, as can be easily seen, the algorithm nicely adapts to the complex features and distributions of the contours, with very little loss in geometric information. These results (both in compression factors and visual quality) are typical of the experiments we ran.

5.1.2 Size-Quality Tradeoff

This section considers the size-quality tradeoff, measuring the improvement in approximation quality as the approxi-



Windspeed dataset **Noise dataset** **eScan dataset**

Figure 5: Approximations produced by Topology-Sensitive-Compress with $k = 64$.

mation size increases. See Figure 6. The figure shows the plots for the same three datasets: *noise* (5000 segments), an Australian rainfall dataset (3000 segments), and an Australian temperature dataset (4200 segments). These datasets were chosen because they exhibit increasing geometric complexity in terms of the contour boundaries (noise being the simplest and the temperature being the most complex). The datasets and their approximations appear in Figure 5 (noise), Figure 10 (rainfall), and Figure 9 (temperature). The error shown in Figure 6 uses the absolute error, normalized by the total perimeter of all contours in the dataset.

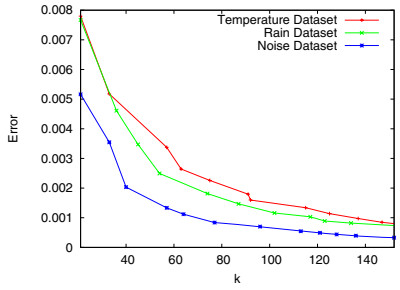
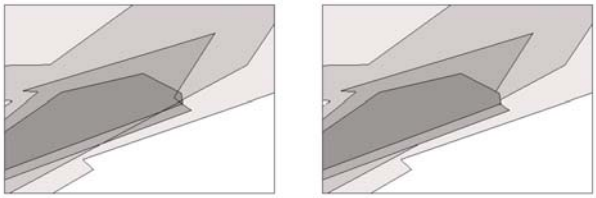


Figure 6: Size-quality tradeoff

The figure shows that for all datasets, the error initially decreases rapidly as memory increases, and then levels off. For an approximation size of $k = 90$ or more, the error is already below 0.2% of the total perimeter. The error decrease is slower after $k = 90$. Despite a broad range of shape complexity in these contours, our algorithm performs equally well on all datasets—as expected, it only takes a little bit more memory for the more complex datasets to reach the low error value.

5.1.3 *Discussion of the Untangling Phase*

In this section we demonstrate a few important aspects of the untangling phase of the algorithm. Figure 7 shows an example of contour simplification for the noise data set, with approximation size $k = 32$. The approximate contours cross in the figure on the left; the result of untangling is shown on the right. In this case, untangling does not increase the maximum approximation error. In fact, this was the case in all of our experiments—untangling corrected the topological violation but did not increase the error. This can be attributed to the fact that the Hausdorff is a maximum error metric, and so for the error to increase, the maximum



Unresolved approximations Resolved approximations

Figure 7: Untangling phase

error segment must be intersected and replaced in a manner that increases the error, which appears to be rare.

When multiple contours share common boundaries, we can use an implicit method of representation to avoid duplication of data: in particular, we use pointers from the modified chain to its replacing subchain, and so the untangling does not increase the number of vertices in the approximation.

Even without using an implicit representation, the memory impact of untangling is small. Figure 8 shows the impact of cycle formation and untangling on the number of segments in the approximations for the *noise* and rainfall datasets. Starting from a stick-fitting approximation with k segments (and nominally $2k$ vertices), cycle formation increases the number of segments by up to 30% (but reduces the number of vertices to $1.3 \times k$), and untangling adds up to another 10%. As k increases, the approximate contours are closer to the input contours, and less untangling is necessary. For these examples, no untangling is needed for $k \geq 70$.

5.2 Comparison with the Rectangle-Refinement Heuristic

The problem of contour-map approximation was discussed by Hellerstein et al. [12] as a natural setting for in-network aggregation of spatial data. Their proposed algorithm for simplifying a single contour uses a simple refinement heuristic: start with a rectangular bounding box of the contour, and then repeatedly *subtract* the largest area rectangle from the bounding box that does not include any point of the contour. Each such subtraction increases the approximation size by at most four new vertices. This step is repeated until the approximation has k vertices.

We refer to the rectangle refinement heuristic of Hellerstein et al. [12] as RR, and show below a comparison between that heuristic and our scheme. We also compared our

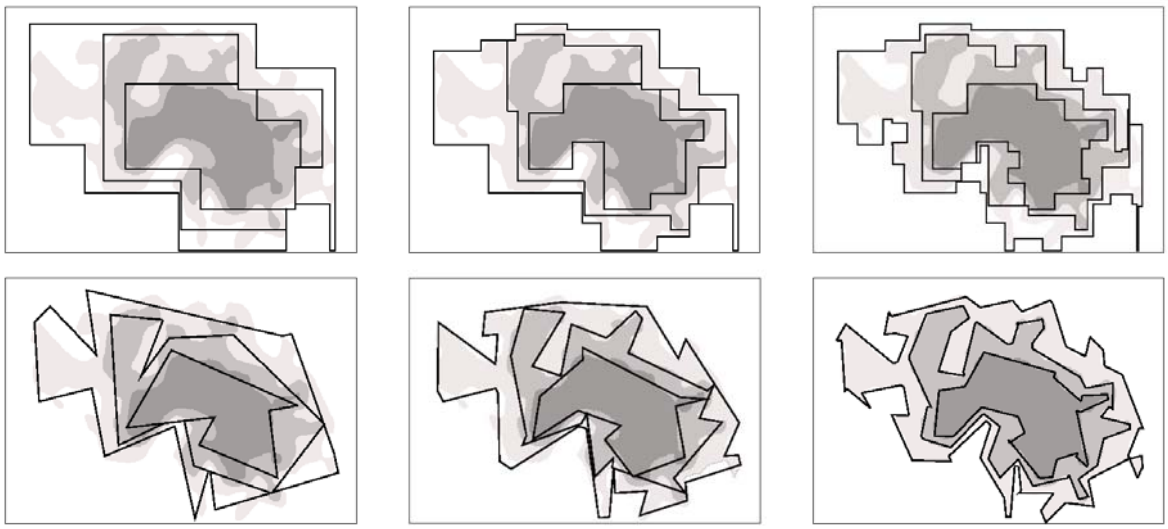


Figure 9: Comparative evaluation: First row shows approximations for RR and second row for Topology-Sensitive-Compress

algorithm against approximation schemes based on wavelets and the Douglas-Peucker algorithm [2]. Our new algorithm outperforms them all, but due to space constraints we include the details only for RR.

The RR heuristic computes connected approximations for individual contours without respecting the topological structure of the different contours. We resolve the intersections among different contour approximations in exactly the same way as in TOPOLOGY-SENSITIVE-COMPRESS.

Figure 9 shows the approximations produced by RR and TOPOLOGY-SENSITIVE-COMPRESS for the temperature dataset with $k = 32$, 64, and 128 segments. Note that in this case, k is the number of segments after the cycle-formation stage, in order to compare the memory allocation schemes of the two algorithms (also, the RR algorithm does not propose any scheme to resolve the intersections).

For $k = 32$ both the approximations are quite coarse. For $k = 64$, TOPOLOGY-SENSITIVE-COMPRESS approximates the original contours quite closely, while RR is hampered by the rectilinear style of approximation. At $k = 128$, TOPOLOGY-SENSITIVE-COMPRESS represents the dataset well, while the approximation by RR still has large local deviations.

5.3 Quantitative Performance

In this section we compare the error in the approximations produced by our algorithm, RR, and the dynamic program [3]. that uses disconnected segments. Clearly, the dynamic program gives smaller error than any algorithm that produces connected approximations with the same number of segments.

Table 1 compares the three schemes quantitatively. For all memory sizes our algorithm clearly outperforms RR. The approximations generated by TOPOLOGY-SENSITIVE-COMPRESS using k segments have smaller error than the dynamic program approximations using $k/2$ segments, suggesting that our worst-case analysis is highly pessimistic. This performance was observed for all of our datasets.

[3]Due to lack of space, we do not describe the dynamic program algorithm, but it is quite straightforward and will be presented in the full version of the paper.

# segments	RR	TSC	DP
32	66.00	16.24	8.18
64	51.78	7.96	4.55
128	49.76	3.14	1.60

Table 1: Error comparison between RR, Topology-Sensitive-Compress, and dynamic programming.

5.4 Error Metrics

The stick-fitting algorithm uses absolute error when deciding which fragments to merge. This means that it always merges the two fragments whose merge leads to smallest increase in the error. This implicitly favors larger contours and causes more merges to occur among the smaller contours. If there is a large variance among the contour sizes, the absolute-error-based merging will allocate a bigger share of the memory k to larger-size contours.

An alternative would be to use relative error, scaling the absolute error by the size (say, some function of the perimeter) of the contour. All of our theoretical results carry over to the relative error as well, with the caveat that untangling must proceed from the smallest-error to largest-error contours to guarantee that no contour's error grows too much. Figure 10 illustrates the difference in empirical results one gets from these two error measures on the Australian rainfall dataset. In this case also, k is the number of segments after the cycle formation phase, so as to compare the distribution of memory by the two error schemes. It shows a contour map in which the innermost contour is significantly smaller than the outermost one. When the approximation size is small ($k = 32$), the figure on the top-left shows that the absolute error does a very poor job of approximating

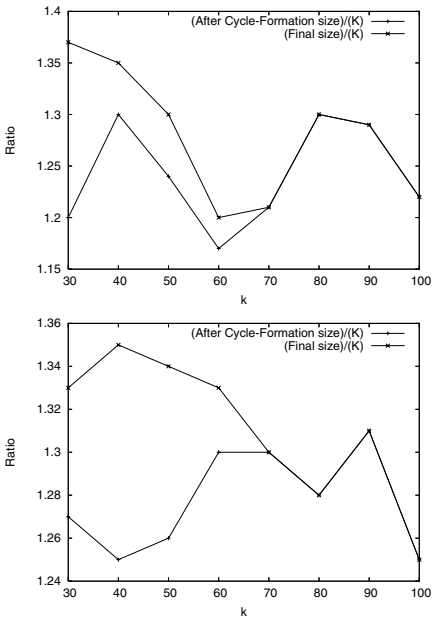


Figure 8: The effect of cycle formation and untangling on representation size in the *noise* (top) and rainfall (bottom) datasets.

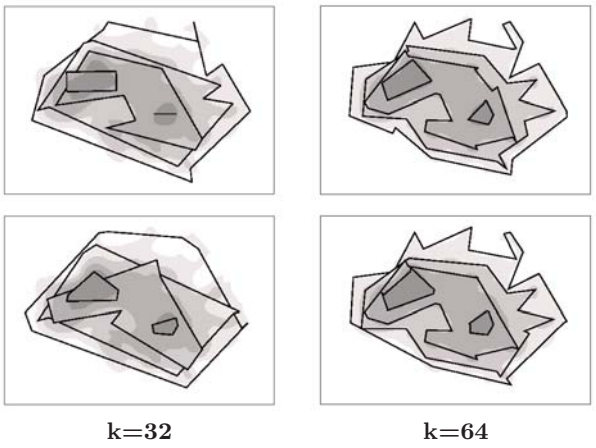


Figure 10: Approximations obtained using absolute error (top row) and relative error (bottom row); first column $k = 32$, second column $k = 64$.

the innermost contour, reducing it to a single segment. By comparison, the (perimeter-weighted) relative error does a much better job. As k grows to 64, both the absolute and relative errors give reasonable approximations.

In our experiments, we found that in general the absolute error produces more pleasing and predictable approximations, with the relative error mostly outperforming when k is small and there is a large variance in contour sizes. Our general recommendation is to use absolute error by default.

6. REFERENCES

[1] A. Arora, P. Dutta, S. Bapat, V. Kulathumani, et al. A line in the sand: A wireless sensor network for target detection, classification, and tracking. *Computer Networks*, 2004.

[2] C. Buragohain, S. Gandhi, J. Hershberger, and S. Suri. Contour approximation in sensor networks. In *DCOSS*, 2006.

[3] K. Chintalapudi and R. Govindan. Localized edge detection in sensor fields. In *SNPA*, 2003.

[4] D. Culler, P. Dutta, C. Tien, R. Fonseca, et al. Towards a sensor network architecture: Lowering the waistline. In *HotOS*, 2005.

[5] A. Dhariwal, B. Zhang, B. Stauffer, C. Oberg, et al. NAMOS: Networked aquatic microbial observing system. In *ICRA*, 2006.

[6] 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*, 1973.

[7] R. Estkowski and J. Mitchell. Simplifying a polygonal subdivision while keeping it simple. In *SOCG*, 2001.

[8] S. Funke. Topological hole detection in wireless sensor networks and its applications. In *DIALM-POMC*, 2005.

[9] O. Gnawali, B. Greenstein, K. Jang, A. Joki, et al. Tenet architecture for tiered sensor networks. In *SENSYS*, 2006.

[10] M. Greenwald and S. Khanna. Power-conserving computation of order-statistics over sensor networks. In *PODS*, 2004.

[11] L. Guibas, J. Hershberger, J. Mitchell, and J. Snoeyink. Approximating polygons and subdivisions with minimum-link paths. In *ISAAC*, 1992.

[12] J. Hellerstein, W. Hong, S. Madden, and K. Stanek. Beyond average: Toward sophisticated sensing with queries. In *IPSN*, 2003.

[13] J. Hershberger and J. Snoeyink. Speeding up the Douglas-Peucker line simplification algorithm. In *SDH*, 1992.

[14] C. Jordan. *Cours d'Analyze de l'cole Polytechnique*. 1887.

[15] P. Juang, H. Oki, Y. Wang, M. Martonosi, et al. Energy efficient computing for wildlife tracking: Design tradeoffs and early experiences with Zebranet. In *ASPLOS*, 2002.

[16] A. Kolesnikov. *Efficient Algorithms for Vectorization and Polygonal Approximation*. PhD thesis, Department of Computer Science, University of Joensuu, 2003.

[17] A. Kroller, S. Fekete, D. Pfisterer, and S. Fischer. Deterministic boundary recognition and topology extraction for large sensor networks. In *SODA*, 2006.

[18] P. Liao, M. Chang, and C. Kuo. Contour line extraction with wireless sensor networks. In *ICC*, 2005.

[19] S. Madden, R. Szewczyk, M. Franklin, and D. Culler. Supporting aggregate queries over ad-hoc wireless sensor networks. In *WMCSA*, 2002.

[20] A. Mainwaring, J. Polastre, R. Szewczyk, D. Culler, et al. Wireless sensor networks for habitat monitoring. In *WSNA*, 2002.

[21] K. Mayer, K. Ellis, and K. Taylor. Cattle health monitoring using wireless sensor networks. In *ICCCN*, 2004.

[22] X. Meng, L. Li, T. Nandagopal, and S. Lu. Event contour: An efficient and robust mechanism for tasks in sensor networks. Computer Networks.

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

[24] N. Shrivastava, C. Buragohain, D. Agrawal, and S. Suri. Medians and beyond: New aggregation techniques for sensor networks. In *SENSYS*, 2004.

[25] M. Singh, A. Bakshi, and V. Prasanna. Constructing topographic maps in networked sensor systems. In *ASWAN*, 2004.

[26] I. Solis and K. Obraczka. Efficient continuous mapping in sensor networks using isolines. In *MOBIQUITOUS*, 2005.

[27] G. Tolle, J. Polastre, R. Szewczyk, D. Culler, et al. A macroscope in the redwoods. In *SENSYS*, 2005.

[28] Y. Wang, J. Gao, and J. Mitchell. Boundary recognition in sensor networks by topological methods. In *MOBICOM*, 2006.

[29] G. Werner-Allen, J. Johnson, M. Ruiz, J. Lees, et al. Monitoring volcanic eruptions with a wireless sensor network. In *EWSN*, 2005.

[30] W. Xue, Q. Luo, L. Chen, and Y. Liu. Contour map matching for event detection in sensor networks. In *SIGMOD*, 2006.

[31] Y. Yao and J. Gehrke. The Cougar approach to in-network query processing in sensor networks. In *SIGMOD*, 2002.

[32] Y. Zhao, R. Govindan, and D. Estrin. Residual energy scan for monitoring sensor networks. In *WCNC*, 2002.