Shape Segmentation and Applications in Sensor Networks

Xianjin Zhu Rik Sarkar Jie Gao

Department of Computer Science, Stony Brook University. {xjzhu, rik, jgao}@cs.sunysb.edu

***Abstract*—Many sensor network protocols in the literature implicitly assume that sensor nodes are deployed uniformly inside a simple geometric region. When the real deployment deviates from that, we often observe degraded performance. It is desirable to have a generic approach to handle a sensor field with complex shape. In this paper, we propose a segmentation algorithm that partitions an irregular sensor field into nicely shaped pieces such that algorithms and protocols that assume a nice sensor field can be applied inside each piece. Across the segments, problem dependent structures specify how the segments and data collected in these segments are integrated. This unified topology-adaptive spatial partitioning would benefit many settings that currently assume a nicely shaped sensor field.**

Our segmentation algorithm does not require sensor locations and only uses network connectivity information. Each node is given a 'flow direction' that directs away from the network boundary. A node with no flow direction becomes a sink, and attracts other nodes in the same segment. We evaluate the performance improvements by integrating shape segmentation with applications such as distributed indices and random sampling.

I. INTRODUCTION

Sensor networks have a unique geometric character as sensor nodes are embedded in, and designed to monitor, the physical environment. Thus the physical locations of sensor nodes have a fundamental influence on the system design in all aspects from low-level networking and organization to high-level information processing and applications. Clearly sensor placement affects connectivity and sensing coverage, which subsequently affects basic network organization and networking operations. Recently a number of research efforts have identified the importance of not only sensor locations, but also the global geometry and topological features of a sensor field. The 'topology' here means algebraic topology and refers to holes or high-order features. In the literature, uniformly random sensor deployment is arguably the most commonly adopted assumption on sensor distribution — but is rarely the case in practice. Many algorithms and protocols proposed for a dense and uniform sensor field inside a simple geometric region, may have degraded performance when they are applied to an irregular sensor field with holes, etc.

Let us use routing as an example. Geographical routing, in which a packet is greedily forwarded to the neighbor that is geographically closest to the destination [1], [2], [3], has attracted a lot of interests. It is simple, elegant, and has little routing overhead. In a dense and uniform sensor field with no holes, geographical forwarding produces almost shortest paths and is robust to link or node failures and location inaccuracies. However, when the sensor field is too sparse, has holes or a complex shape, greedy forwarding fails at local minima. This is due to a mismatch of routing/naming rules with the real network connectivity. Two nodes that are geographically close may actually be far away in the connectivity graph. Thus, when these topological features (e.g., holes) become prominent, the naming and its coupled routing protocol should represent the real network connectivity and adjust to these topological features accordingly [4], [5].

Beyond geographical routing, the global topology of a sensor field has fundamental influence on how information gathered in the network should be processed, stored and queried. In a sensor field with narrow bottlenecks, more aggressive in-network processing is expected to minimize the traffic flowing through bottleneck nodes. In a distributed storage scheme, the global geometry should be taken into account to achieve better load balance on storage nodes. Many existing information processing algorithms do not account for the global geometry of a sensor field yet. A typical example is the quadtree type of geometric decomposition hierarchy, which has been extensively used to exploit spatial correlation in sensor data (e.g., DIFS, DIM [6], [7]) for efficient multi-resolution storage. In a sensor field with holes, a standard geometric quadtree (the bounding rectangle is partitioned into 4 equal-size quadrants recursively) may become unbalanced with lots of big empty leaf blocks. An imperfect partition hierarchy subsequently affects the performance, especially load balance, of all algorithms and data structures built on top of it. In another example, random sampling of a sensor node can be conducted by choosing the node closest to a random location. To achieve a uniform distribution, the sampling probability needs to be adjusted by the area of the corresponding Voronoi cell [8], [9]. In an irregular sensor field, the Voronoi cells have vastly varying areas. Thus the sampling efficiency suffers as a lot of trials end up being rejected.

One approach to deal with irregularly shaped sensor field is to develop virtual coordinates with respect to the true network connectivity, as in the case of routing [4], [5]. One may follow this line and re-develop algorithms for all the other problems on virtual coordinate systems. Both the development of virtual coordinates and topology-adaptive algorithms on top of that are highly non-trivial. In this paper, we propose to develop a unified approach to handle complex geometry. We propose a segmentation algorithm that partitions an irregular sensor field into nicely shaped pieces such that algorithms that

assume a uniform and dense sensor distribution can be applied inside each piece. Across the segments, problem dependent structures specify how the segments and data collected in these segments are integrated. This unified topology-adaptive spatial partitioning would benefit many settings that currently assume sensor field with nice simple shape. There is not much prior work on segmenting a sensor field. The mostly related one by Kröller *et al.* [10] proposed a boundary detection algorithm with which one can organize sensor nodes by 'junctions' and 'streets'. Our goal is to further explore segmentation algorithms suitable for a discrete sensor field as well as applications that can benefit from it.

In this paper, we consider a static sensor network with an irregular shape. We take the viewpoint to regard the sensor network as a discrete sampling of the underlying geometric environment and develop a 'shape segmentation' algorithm. This is motivated by the fact that sensor networks are to provide dense monitoring of the embedded space. The analysis of geometric shapes has been extensively studied in graphics and computational geometry with many shape segmentation algorithms proposed in the literature [11], [12], [13], [14], [15]. These algorithms typically work in a centralized setting with ample computational resources. Shape segmentation problem for a discrete sensor field faces a number of new challenges, and requires non-trivial algorithm design to achieve sufficient robustness to input inaccuracies.

- Sensor nodes start with no idea of the global picture. We consider the approach of collecting all information at a centralized node not a scalable solution. Segmentation algorithm needs to be automatic and distributed in nature.
- Sensor nodes may not know their geographical locations — automatic and scalable localization (without GPS) is still a challenging problem. Even when they do, the locations may come with large inaccuracies.
- When sensor locations are not available, the distance between two nodes is often approximated by their minimum hop count value, which is always an integer. This rough approximation introduces inevitable noise to any geometric algorithms that use the hop count to replace the Euclidean distance.

We propose to adapt a shape segmentation scheme by using flow complex [11] to sensor networks. The algorithm uses only the connectivity information and does not assume that sensors know their locations. We first discover all the hole boundaries and the outer boundary, say by the algorithm developed by Wang *et al.* [16]. We let the boundary nodes flood inward and every node records the minimum hop count from the boundary. Each node is then given a 'flow direction', the direction to move away fastest from boundaries. A node may be singular with no flow direction and is named as a sink[1]. The sensor field is partitioned to segments in a way that nodes in the same segment flow to the same sink. This naturally partitions the sensor field along narrow necks. In the geometric version, all the sinks stay on the medial axis of the field, which is the set of points with at least two closest points on the boundary and constitutes a 'skeleton' of the shape. In a discrete network sinks may appear far away from the medial axis due to local noises and connectivity disturbances. In addition, in degenerate cases such as a corridor with parallel boundaries, many nodes on the medial axis may be identified as sinks. We apply a local merging process such that nearby sinks along the medial axis with similar hop counts from the boundary, together with their corresponding segments, are merged. In the end, each segment is given a unique identifier by the sink(s). All the nodes in the same segment are informed of the identifier distributed along the reversed flow pointers. The algorithm is communication efficient. It involves a couple of limited flooding from the boundary nodes to the interior of the network. All the other operations only involve local computations. With given boundaries, the segmentation algorithm incurs a total transmission cost of $O(n)$.

We tested the segmentation algorithm under various topologies and node density. We observed intuitive segmentation along narrow necks in a sensor field with reasonable node degree (around $7 \sim 8$). To show the benefit of segmentation, we evaluated the performance improvements by integrating segmentation algorithm with two existing algorithms that currently assume a uniform sensor field: a distributed index for multi-dimensional data (DIM) [7] and random sampling [8], [9]. For both algorithms, we show that in an irregular sensor field, the segmentation algorithm provides improved performance and load balance.

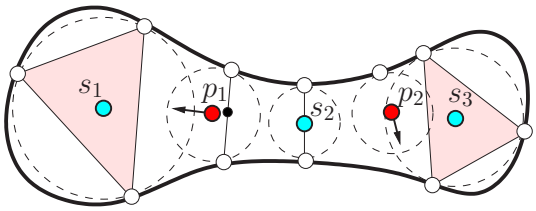


Fig. 2. Two regular points (p_1 and p_2) with their flow vectors. Sinks (s_1, s_2 and s_3) stay inside the convex hull of their closest points on the boundary.

II. Notations and Definitions

We first introduce some notations and definitions defined in the continuous domain [11]. In the next section we show how to adapt them in a discrete network. For a connected continuous region $\mathcal{R}$, denote by $\mathcal{B}$ the boundaries, represented by a set of closed curves, each bounding either an inner hole or the outer boundary. For a point $x \in \mathcal{R}$, the distance from x to the boundaries is define by $h(x) = \min\{||x - p||^2 : p \in \mathcal{B}\}$. The *medial axis* is the set of points in $\mathcal{R}$ with at least two closest points on the boundary. The distance function h is continuous, and smooth everywhere except points on the medial axis. We call a point x a critical point, or a *sink*, if x is inside the convex hull of its closest points on $\mathcal{B}$, denoted by $\mathcal{H}(x)$. For example, sink s_1 has three closest points on the boundary and stays inside the triangle spanned by them. Obviously all the sinks stay on the medial axis. All non-critical

[1]Notice that the sink we refer to in this paper is not a data sink or aggregation center (base station), although the sinks are good indicators of where to place base stations or aggregation centers

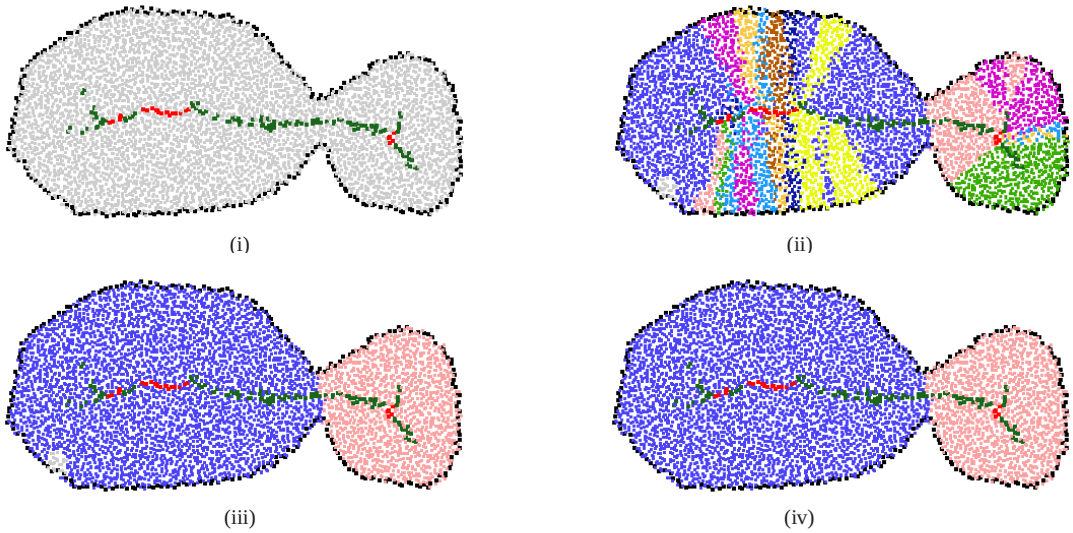


Fig. 1. The fish network. 5000 nodes, Avg. degree 8. Boundary nodes are shown in black. (i) Medial axis nodes shown in dark green. Sink nodes shown in red. (ii) The stable manifolds of the sink nodes, shown in different colors. (iii) Nearby sinks with similar hop counts to the boundary, along with their stable manifolds, are merged. Orphan nodes shown in grey. (iv) The final result after processing orphan nodes.

points are called regular. The *driver*, $d(x)$, is defined as the closest point in $\mathcal{H}(x)$ (e.g., in Figure 2, the driver of p_1 is the smaller black dot). For a point not on the medial axis, its driver $d(x)$ is the unique closest point on $\mathcal{B}$, e.g., p_2 in Figure 2. For a sink, the driver is itself. Now the *flow* is defined as a unit vector $v(x) = \frac{x-d(x)}{||x-d(x)||}$ (i.e., the direction that points away from its driver), if $x \neq d(x)$ and 0 otherwise. It has been proved in [11] that all the points will flow to sinks. The *stable manifold* of a sink x, denoted as $\mathcal{S}(x)$ is simply the set of points that flow to it by following the flow directions. In Figure 2 there are a total of three sinks and two large stable manifolds (the stable manifold for s_2 is just a line segment, called a degenerate segment). Sinks s_1 and s_3 correspond to local maxima of the distance function $h(x)$, sink s_2 is a saddle point. The sink can be considered, to some extent, a 'center' of the segment. Rigorously, we have the following theorem. Define a ball centered at a point to be *locally maximal* if the ball is entirely inside the shape and by moving the center of the ball infinitesimally one cannot enlarge the size of the ball.

Theorem 2.1. *All locally maximal balls are centered at sinks.*

Proof: Consider a locally maximal ball B centered at node x. If x is not a sink, it must have a flow direction. Then B will become larger if the center of the ball is shifted a small distance in the direction of the flow, because the distance function increases along the follow direction. This contradicts to the fact that B is locally maximal. So x must be a sink. ■

This theorem gives some property of the non-degenerate segments induced. Intuitively, we want to obtain a few number of large and 'fat' pieces. One way to measure the fatness of a segment $\mathcal{S}(x)$ is to consider the largest ball, completely inside $\mathcal{R}$, centered at a point inside $\mathcal{S}(x)$ (this ball can stick out of the segment) against the minimum enclosed ball of $\mathcal{S}(x)$. Theorem 2.1 says that we obtain k non-degenerate segments, with k equivalent to the number of locally maximal balls — the largest ball centered inside $\mathcal{S}(x)$ is exactly the locally maximal ball centered at the sink x. If we merge two segments, the fatness of the resulting segment will be hurt since the size of the largest ball inside the segment does not increase but the size of the minimum enclosed ball may increase. One may improve the fatness of segments by reducing the sizes of the minimum enclosed balls, at the expense of introducing more segments.

III. Segmentation Algorithm

The flow and stable manifolds described in section II naturally partition a continuous domain into segments along narrow necks — each stable manifold is a segment. In this section we show how to implement the flow and the segmentation in a discrete sensor field, when we do not have node location information, nor the distance function. Unlike the continuous case, here we approximate the Euclidean distance function to the boundary by the minimum hop count to boundary nodes. As for the notion of closest *points* on the boundary, an interior node x has one or more closest *intervals* — each interval is a consecutive sequence of nodes on the boundary with minimum hop count from x. We want to find, for each sensor x, a flow pointer that points to one of its neighbors, in order to move 'fastest' away from the boundary. The challenge is to assign these flow pointers and identify the sinks in a robust way such that there is no loop and an intuitive segmentation can be derived. We describe an outline of the algorithm followed by the details of each step.

1) **Boundary detection.** Find the boundaries of the sensor field using the algorithm described in [16]. This algorithm identifies boundary nodes and connects them into cycles that bound the outer boundary and interior hole boundaries of the sensor field.
2) **Construct the distance field.** The boundary nodes simultaneously flood inward the network and each node records the minimum hop count to the boundary, as well as the interval(s) of closest nodes on the boundary.

Nodes on the medial axis can identify themselves as they have two or more closest intervals.

3) **Compute the flow.** Each node x computes a flow pointer that points to its *parent* — the neighbor with a higher hop count from the boundary and the most symmetric closest intervals on the boundary among all such neighbors. Nodes on the medial axis with no neighbor of higher hop count become *sinks*. See Figure 1(i) and (ii).
4) **Merge nearby sinks.** Nearby sinks on the medial axis with similar hop count from the boundary are merged into a *sink cluster* and agree on a single *segment ID*.
5) **Segmentation.** The nodes that ultimately flow to the same sink cluster are grouped into the same segment. We let each sink disseminate the segment ID along the reversed *parent* pointers. See Figure 1 (iii) for the merged segments.
6) **Final clean-up.** Due to irregularities in node distribution, some nodes have locally maximum hop count to boundary but do not stay on medial axis — thus didn't get recognized as sinks. These, and nodes that flow into them are left *orphan*. In the final clean-up phase, we merge the orphan nodes to their neighboring segments.

A. Boundary detection

We use the boundary detection algorithm proposed by Wang *et al.* [16] to identify boundaries with the connectivity information. The boundaries of inner holes and outer boundary are assigned unique identifiers, and nodes along each boundary cycle are assigned ordered sequence numbers. Every boundary node thus knows the identifier and the length of the boundary to which it belongs, and its own sequence number on that boundary. We refer to the set of nodes on boundary j as B_j.

B. Construct the distance field

With the boundary nodes identified, we construct a distance field such that each node is given a minimum 'distance' to the sensor field boundary. Since we do not assume location information, our only measure of distance in the network is the number of hops to the boundary nodes. The problem is that a node typically has more than one nearest boundary nodes with the same hop counts away (thus may be identified to be on the medial axis). Hence we keep not the *closest node* but the *interval* of closest nodes. An interval I on the j^{th} boundary cycle is simply a sequence of nodes along the boundary cycle. It can be represented uniquely by a 4-tuple $(j, start_I, end_I, |B_j|)$, where $start_I$ and end_I are the two end points, $|B_j|$ is the length of the j^{th} boundary.

We have the boundary nodes synchronize among themselves [17], [18] and start to flood the network at roughly the same time. The boundary nodes initiate a flood with messages of the form (I, h) where I is an interval nearest to the transmitting node, and h is the distance to nodes in I. A node p keeps track of the set S_p of *intervals* of boundary nodes nearest to it. On receiving a message (I, h), a node p compares h to its current distance h_p to the boundary:

- If $h > h_p$, discard the current message.
- If $h < h_p$, discard all existing intervals, set $h_p := h$, $S_p := \{I\}$ and send $(I, h+1)$ to all neighbors.
- If $h = h_p$, merge I with adjacent and overlapping intervals on the same boundary if there is any. Otherwise, simply add I into S_p. Send $(I, h+1)$ to all neighbors.

After this computation of sets of nearest intervals for all nodes in the network, nodes on the medial axis can be identified as follows:

Definition 3.1. ***Nodes on medial axis****: A node p is a medial axis node if $|S_p| > 1$.*

Note that our definition of medial axis differs from that used in existing articles (e.g. [5]). What is the most appropriate analog of the continuous medial axis in a discrete network is yet to be debated. We find this definition quite robust and most suitable for our purposes. Figure 1 (i) shows the medial axis of the fish network found with this protocol. The protocol described above is easy to implement and works well in simulations. As pointed out in [5], if all boundary nodes initiate the flood at about the same time, then this method keeps the total communication cost very low. The distance field can also be constructed with two rounds of boundary flooding: in the first round each node records the hop count to the boundary; in the second round each node broadcasts their closest intervals. To simplify the theoretical analysis of message complexity, we use the two-round version in Section III-G.

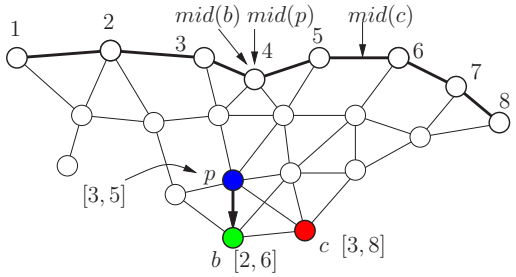


Fig. 3. Node p selects node b as its parent $v(p)$, as b is the more symmetric neighbor. The closest intervals of node p, b, c are shown.

C. Compute the flow pointer

Once each node p learns its minimum distance h_p from the boundaries and the intervals of closest nodes at distance h_p from it, it can construct the flow pointer and find sinks locally. Observe that for any pair of neighboring nodes p and q, if $h_p < h_q$, then the closest intervals of p must be included in the closest intervals of q. Rigorously, $\forall\, I \in S_p\,, \exists\, I' \in S_q$ such that $I \subseteq I'$.

Each node creates a *flow pointer* to its *parent*, the neighbor who is strictly further away from the boundary than itself, and whose closest intervals are most *symmetric* with respect to its own closest intervals. In Figure 3, node p selects node b as its parent $v(p)$ because the mid point of b's interval is closer to the mid point of its own interval. The intuition behind this is to select the neighbor that represents the best movement away from all parts of the boundary. We make this notion rigorous by defining the angular distance of two neighboring nodes.

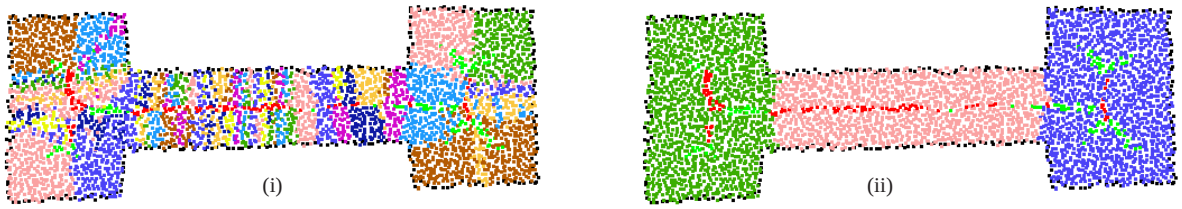


Fig. 4. The corridor network. (i) Opposite boundaries run parallel to each-other, producing several sinks in succession, resulting in the fragmented segmentation. (ii) Segmentation after merging nearby sinks with similar distances to the boundary.

Definition 3.2. ***Mid point****: The mid point $mid(I)$ for an interval I defined on the boundary j, is the $\frac{|I|+1}{2}$-th element in the continuous sequence modulo $|B_j|$ of I, if $|I|$ is odd, else it is the mean of the $(\frac{|I|}{2})$-th and the $(\frac{|I|}{2}+1)$-th elements.*

Definition 3.3. ***Angular distance****: The angular distance $\delta(p,q)$ between neighboring nodes p and q where $h_p < h_q$ is defined as $\delta(p,q) = \sum_{I \in S_p} \min_{I' \in S_q} |mid(I) - mid(I')|$, I and I' must be on the same boundary.*

Using the function δ, each node p selects a neighbor q such that $h_p < h_q$, and the sum of distances from the mid points of its intervals to the corresponding intervals of q is less than that for any other neighbor of p.

Definition 3.4. ***Flow pointer****: Let H_p be p's neighbors with higher hop count from the boundary, i.e., $h_p < h_q$, for $q \in H_p$. Then the parent of p, $v(p)$ is defined as the neighbor in H_p with minimum angular distance, $v(p) = \arg\min_{q \in H_p} \delta(p,q)$.*

A typical node, for example p in Figure 3 would have only one boundary interval nearest to it. Nodes on the medial axis have more than one such interval, in which case, the parent is chosen based on the sum of mid point distances instead of a single distance, as described above.

Definition 3.5. ***Sink****: A node c is a* sink*, if c is a medial axis node, and has locally maximum hop count from the boundary.*

The sinks of the fish network, by this definition, are shown in Figure 1(i). Sinks are those medial axis nodes without a parent. With the flow, the sequence of directed edges starting at any non-sink node p ends at a unique root c. Since a node p selects its parent only if its parent has a higher hop count from the boundary, there cannot be a cycle in the directed graph implied by the flow. Thus, the nodes in the network are organized into directed forests, with the nodes in the same tree flow to the same root by following their flow pointers. In a continuous domain, the *stable manifolds* of the sinks form the segments. In a discrete network, the analog of the *stable manifold* of a sink c would be the directed tree rooted at c, such that any directed path in this tree ends at c.

D. Merge nearby sinks

One can take the stable manifolds of the sinks, i.e., the trees rooted at the sinks, as the segmentation. But this may result in a heavily fragmented network (see Figure 1(ii)). This happens when there are a cluster of sinks on the medial axis, and there is a tree rooted at each. This situation becomes severe when some parts of boundaries run parallel to each other. See Figure 4 (i). In this case we have a sequence of nodes on the medial axis where no node is farther from the boundary than its neighbors, and all these nodes become sink nodes.

We would like to merge nearby sinks with similar distances from boundary as well as their corresponding segments. We call the merged sinks a *sink cluster*. A sink cluster K is represented by the tuple $(id, h_{\max}, h_{\min})$, where id is the minimum ID of all the sinks in the cluster, i.e., the ID of the *leader* of the cluster. $h_{\max}$ and $h_{\min}$ are the maximum and minimum distances from any sink to the boundary respectively. We set a user defined threshold t to guarantee that $|h_{\max} - h_{\min}| \leq t$. Each sink node waits for a random interval to start the merging process to avoid contention. Initially each sink is by itself a sink cluster, and also a sink cluster leader. A sink cluster leader searches along the medial axis for nearby sinks (or sink clusters) to be merged. Specifically, a sink cluster leader c sends a *search message* of its current sink cluster $(id, h_{\max}, h_{\min})$ to all neighboring nodes on the medial axis. Each medial axis node p of cluster $(id', h'_{\max}, h'_{\min})$, on receiving this message, executes the following rule (let $H_{\max} = \max(h_{\max}, h'_{\max})$ and $H_{\min} = \min(h_{\min}, h'_{\min})$) :

- if $|H_{\max} - H_{\min}| > t$, discard the message.
- else forward the message to all neighboring nodes on the medial axis. Furthermore, if p is a sink cluster leader, p would like to merge its sink cluster with c's sink cluster.

In the second case, when p wants to be merged, it sends a *merging request* with $(id', h'_{\max}, h'_{\min})$ to c and waits for the response. The leader node c makes merging decision based on its latest $h_{\max}$ and $h_{\min}$ values at the time the request reaches c (as c may have merged with other sink clusters before it receives the request from p) and selects the smallest sink ID in the merged cluster as the new cluster leader and cluster ID. We can suppress messages by only sending out *merging response* after a sink receives a few merging requests, so that a set of merged sink clusters can be updated simultaneously. Note that the handshake is carried out by cluster leaders, and search messages die out when they reach nodes on the medial axis with hop count too large or too small. When a new cluster appears, its cluster leader sends out a new search message looking for sink clusters to be merged. The process terminates automatically when no merging request is received after a timing threshold. In a sparse network when the medial axis is not connected, we search 2 or 3 hop neighbors for nearby medial axis nodes.

The variable t is a threshold defined by the user. It determines the granularity of segmentation. Smaller values of t imply that merging step will stop at a small change of the

distance from the boundary and hence collect fewer sinks together into sink clusters. Thus there will be more segments created by the algorithm. For larger values of t, the algorithm will create fewer and larger segments. Figure 5 shows the differences caused by variation in the value of t. In many such situations, the most preferable segmentation is likely to be dependent on the nature of the application. We leave it to the user's discretion to set the value of t. Note that the user can change the value of t even after the network has been deployed by flooding a message from any one node in the network. This would require a re-computation of only the last three steps of the algorithm, starting at merging of sinks.

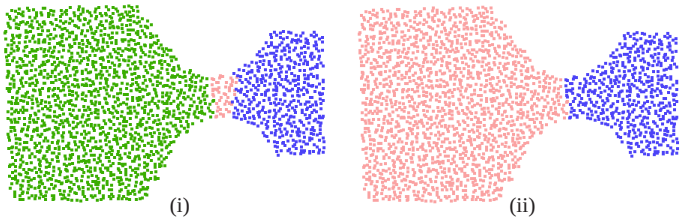


Fig. 5. Network with 2200 nodes, with avg 6 neighbors per node. Segmentation with threshold (i) $t = 2$; (ii) $t = 4$.

A second approach to merge sinks and segments is to use *second derivative of distance field*. The idea is that nodes around a narrow neck may have significant difference on minimum hop counts to the boundary. Every node can calculate the second derivative of the distance to the boundary locally. Merging starting at any cluster leader would stop at a node with distinct different second derivative value.

E. Segmentation

Each sink cluster defines a segment, as all the trees rooted at nodes in the sink cluster. To create the segments of the network, each sink node c propagates the ID of the sink cluster to all the nodes in the tree rooted at c. This can be simply done by reversing the flow pointers. The ID of the sink cluster is also considered as the ID of the segment. Figure 1(iii) and Figure 4(ii) show the result of this construction.

F. Final clean-up

Due to noises and local disturbances, it is possible that some nodes have locally maximum hop count to the boundary, but are not medial axis nodes. This is likely to happen in sparser regions of the network or near the boundaries if the boundaries detected are not tight. In such cases, the node at the local maximum and all nodes in the tree rooted at it are left without any segment assignment. We refer to such nodes as *orphan nodes* (e.g., the grey nodes in Figure 1(iii)). At the final clean-up stage, we assign the orphan nodes to a nearby segment. In a connected network, there always exists an orphan node p such that some neighbor q of p is not orphan. Each such node p selects randomly a non-orphan neighbor q, and merges to that segment. This is executed by all orphan nodes until all nodes are assigned a segment. Figure 1(iv) shows the final result.

Depending on the requirements of applications, the information about the newly formed segments can be disseminated across the network. Each segment has a natural leader — the sink node whose ID the segment takes. This sink node easily collects information about the segment such as node count, neighboring segments, bounding rectangle (if location information is available) etc. This information can be delivered to all nodes in the network, by transmitting along the medial axis and reversed flow pointers. Since the global features of the network have been abstracted into a compact presentation, each node only transmits and stores a small amount of data.

G. Algorithm complexity

We analyze the complexity of the segmentation algorithm with given boundaries. The total communication cost is dominated by the second step, i.e., constructing the distance field, which incurs a couple of limited flooding. Other steps only involve local operations, so the cost is at most $O(n)$, where n is the number of nodes in the network. For the simplicity of analysis, the second step can be implemented in two passes. In the first pass, nodes record the minimum hop count from the boundaries. If the boundary nodes flood inwards almost simultaneously, each node will receive the message from the closest boundary node earlier than any other boundary nodes, thus each node only broadcasts once and all messages received later are suppressed. In the second pass, nodes broadcast their closest intervals. Each node constructs and broadcasts its final interval after receiving messages from all neighbors with lower hop counts. In summary, the distance field can be constructed with $O(n)$ messages. The proposed shape segmentation algorithm is efficient and incurs communication cost of $O(n)$ in total.

H. Simulations

We simulated the algorithm for different shapes of network, and found that an intuitive partitioning into pieces with regular shape is obtained. These networks either represent practical scenarios, like an intersection of two roads (Figure 6(i)), rooms connected by a corridor (Figure 4), or some pathetically difficult cases we come up with. Several examples are shown in Figure 6. In general, the algorithm performs consistently well when the average degree is $7 \sim 8$ or higher. Good results can be obtained for networks of low density by considering a two or three hop neighborhood in the steps of finding flow pointers, merging the sinks and constructing segments. We used a three hop neighborhood in simulations.

IV. Applications

In this section, we present two specific applications that benefit from shape segmentation: a distributed index for multi-dimensional data [7] and random sampling [9]. Both of these applications assume the availability of locations. Such geographical information gives a node local picture of its neighborhood, but nodes are still unaware of global features of the network. Our shape segmentation scheme actually runs without any geographical information. Simulation results show that shape segmentation improves performance in terms of communication cost and load balance.

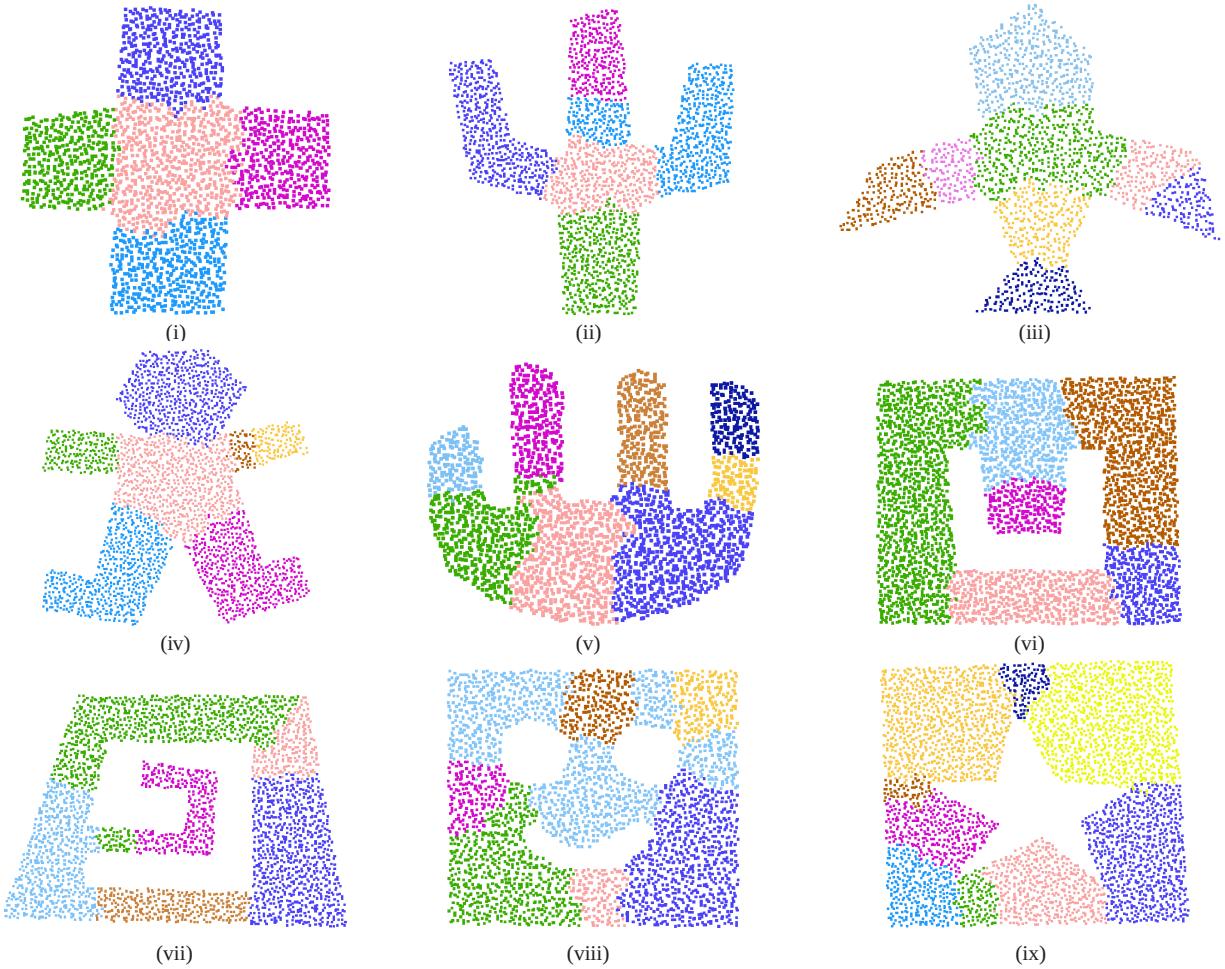


Fig. 6. Segmentation results for miscellaneous shapes and densities. (i) cross: 2200 nodes, avg 12 neighbors per node. (ii) cactus: 2100 nodes, avg 9 neighbors per node. (iii) airplane: 1900 nodes, avg 7.8 neighbors per node. (iv) gingerman: 2700 nodes, avg 8 neighbors per node. (v) hand: 2500 nodes, avg 6.5 neighbors per node. (vi) single-hole: 3700 nodes, avg 13 neighbors per node. (vii) spiral: 2900 nodes, avg 11 neighbors per node. (viii) smiley: 2900 nodes, avg 8 neighbors per node. (ix) star: 3900 nodes, avg 9 neighbors per node.

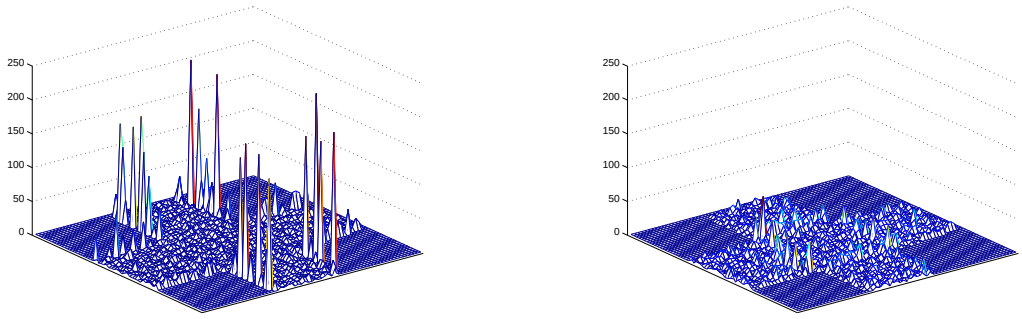
A. Distributed Index

Distributed index for multi-dimensional data (DIM [7]) is a quadtree type hierarchy that supports efficient multi-resolution data storage and range query. The key idea of DIM is to map an event with certain values to a specific area called as *zone*, and store the event with geographic routing to the node owning that zone. The zone is determined by dividing the bounding rectangle of the network alternatively with a vertical or horizontal line until there is a single node inside the zone. When a node generates an event, it estimates the destination zone based on the event value and routes it towards there.

DIM provides a scalable index structure for data storage and performs well in a network field with simple geometric topology. However, it suffers a lot from load unbalancing in a complex shaped sensor field. For a network with arbitrary shape, there will be large empty space in the bounding rectangle. Some nodes (especially those boundary nodes) must take care of a larger zone, and hence store more data than others. Overloaded nodes would be depleted faster than other nodes, which may lead to network partitioning and shorten network lifetime.

With shape segmentation, we can avoid above problems by applying DIM on each segment. Specifically, we first divide the entire event range into several sub-ranges. Let N_i denote the number of nodes belonging to the i^{th} segment, and N denote the total number of nodes. Sub-ranges are divided based on the ratio of N_i/N. The first segment takes care of events within range $[0, N_1/N)$, and the second segment takes care of the range $[N_1/N, (N_1+N_2)/N)$, and so on. A new generated event is divided into several sub-events, each of which is sent towards the corresponding segments respectively. Inside each segment, the sub-event is processed in the same way as the basic DIM algorithm.

To compare the performance of DIM with and without shape segmentation, we run simulations on various network scenarios. We generated 10000 events with values uniformly distributed in a fixed range $[0, 1000]$, and stored them into the network. Figure 7 shows the distribution of storage load for the cross network. We can see that the boundary nodes in the basic DIM structure suffer much higher loads than the rest of the network. On the other hand, with shape segmentation, since each segment has tighter bounding rectangle and each node is associated with an almost equal sized zone, data is seen to be well distributed across the network with no particular



(i) (ii)

Fig. 7. (i) Distribution of storage load in basic DIM structure. (ii) Distribution of storage load in shape segmentation integrated DIM structure.

preference for occurrence of peaks. The peaks in Figure 7(i) reach 248, while the highest peak in Figure 7(ii) is only 65.

Shape segmentation also helps reduce communication cost by mapping events into more accurate locations. Table I shows that the average communication cost in terms of hop counts for every event insertion is much less with shape segmentation in all three different network scenarios, viz. cross (Fig. 6(i)), corridor (Fig. 4) and fish network (Fig. 1). In the cross and corridor network, shape segmentation saves $60\% \sim 70\%$ cost. The gain in fish-type network is about 20%, not as significant as the previous two cases. The reason is that each piece of 'fish' does not tightly match with its bounding rectangle.

cost per event insertion	cross	corridor	fish
without shape segmentation	293.69	359.21	254.37
with shape segmentation	84.15	151.05	204.58

TABLE I. Average data insertion cost for DIM with and without shape segmentation.

B. Random Sampling

We discuss the benefits of shape segmentation with another example - random sampling. Uniform random sampling of a sensor node is a fundamental operation that is used as a basic element in many scenarios such as gossip [9] and information diffusion and storage [19].

The basic sampling procedure works as follows [8]. A node who wants to pick a random sensor in the network first chooses a random geographical location inside the bounding rectangle, and uses geographical routing to route towards that location. The message will eventually arrive at the node closest to the picked location. A node p is picked with a probability proportional to the area of its Voronoi cell. To achieve a uniform sampling distribution, the acceptance probability of sampling at each node needs to be adjusted, as the one with a large Voronoi cell is more likely to be picked. Basically, each sampled node will be accepted with probability $r_i = \min(\tau/a_i, 1)$, where τ is a given threshold and a_i is the area of the Voronoi cell associated with node i. If a node rejects a sample, it will pick a new location and repeat the above process. In an irregular sensor field, the Voronoi cells of different nodes have vastly varying areas. Nodes with large Voronoi cells are picked more likely, yet often get rejected afterwards. Thus, the sampling efficiency suffers as a lot of trials end up in vain. Furthermore, since the fate of each sample can only be determined at the destination node, samples may be rejected after traveling a long path, which incurs expensive communication cost and wastes network resources.

Random sampling integrated with shape segmentation can dramatically reduce the number of unnecessary trials, at the same time achieving uniform sampling. The adapted algorithm runs as follows. Each time before sampling, we first randomly select a segment. Each segment is selected with probability $P_i = N_i/N$. After that, we pick a random location within the bounding rectangle of the selected segment. Within each segment we apply the same sampling algorithm and sampling rejection policy as before. Segments are divided into Voronoi cells with much smaller variation, thus no node would reject samples with abnormally high probability.

We run simulations on the same three typical networks. Results are averaged on 10 rounds, and in each round, we randomly pick 100 samples. For the basic random sampling algorithm, we set τ to the ratio of the size of the network field and the total number of nodes. Each segment has its own τ as the ratio of the segment size and number of nodes belonging to that segment. In Table II, we compare the average number of trials taken to get 100 samples. The basic random sampling algorithm tried 168, 149 and 136 times for 'cross', 'corridor' and 'fish' respectively. Shape segmentation reduces the number to 112, 115 and 123. Table III shows the average communication cost per sample. As expected, the cost in shape segmentation case is less than the basic case.

no. of trials	cross	corridor	fish
without shape segmentation	168	149	136
with shape segmentation	112	115	123

TABLE II. Average number of trials for 100 random sampling.

cost per sampling	cross	corridor	fish
without shape segmentation	477.84	511.95	361.80
with shape segmentation	102.49	182.32	238.47

TABLE III. Average cost per sampling.

With the same observation we got in DIM, shape segmentation shows different levels of improvements in different network scenarios. For these two applications, the performance more or less depends on whether the bounding rectangle is tight enough. We notice that this is due to an inherent assumption of the basic sampling algorithm that uses a bounding rectangle on the sensor field. Further improvement can be

made by using a tighter polygon to approximate the shape of the segment in the basic sampling algorithm.

C. Discussion

We mainly presented the integration of shape segmentation with applications that assume simple geometric regions and rely on geographical locations. But the applications of shape segmentation can go beyond that. For example, the recent work on information dissemination and collection [20] can be directly integrated with shape segmentation. Here, we further discuss a few more fundamental applications. 1) *Load-balanced routing*: Segmentation gives a high level map of the underlying network field. With proper traffic information in each segment, we can choose routes with lighter load to avoid congestion and improve load balance. For example, in a sensor network monitoring two big rooms connected by a couple of corridors with different capacity, from one room to the other, we can distribute traffic based on the capacity of each corridor. 2) *Data aggregation*: Data aggregation is used extensively to explore the data correlation and reduce messages transmitted. However, it is not a trivial problem to place aggregation sinks without the knowledge of the global topology. With shape segmentation, critical points are naturally the candidates of aggregation sinks. The flow pointers for nodes inside each segment naturally form the aggregation tree. Furthermore, by aggregating data within each segment first, we can dramatically reduce network traffic through bottlenecks. 3) *Designing virtual coordinate system*: Shape segmentation even benefits the construction of virtual coordinate systems. Take a landmark-based routing scheme [4] for an example in which the placement of landmarks has a critical impact on its performance. Since the segments capture useful geometric features, like holes, concavity, etc, a few landmarks inside each segment would suffice for routing in and between segments.

V. Conclusion

In this paper we introduced a simple distributed algorithm that partitions an irregular sensor field into nicely shaped segments, by using the connectivity information. We show that segmentation is a generic approach to handle complex geometric features and improve the performance of algorithms that assume a nice regular sensor field. We expect that more applications will benefit from this general approach with improved performance in an irregular sensor field.

In shape segmentation, a generally unsolved issue is that there is no well accepted definition on good segmentation so far. The choice of appropriate segmentation may also depend on the applications. For example, a spiral-like sensor field is equivalently nice as a long corridor for routing protocols, but it needs to be segmented further for applications that require a quad-tree type hierarchy. Therefore, it is always an open choice for the upper level applications to pick a definition and choose proper segmentation granularity. One interesting problem is to classify applications into several categories so that more precise segmentation definitions can be found for each category. We regard this as our future work.

Acknowledgment

We thank Yue Wang for sharing with us the boundary detection code in [16] and Alexander Kröller for a few useful discussions.

References

[1] P. Bose, P. Morin, I. Stojmenovic, and J. Urrutia, "Routing with guaranteed delivery in ad hoc wireless networks," *Wireless Networks*, vol. 7, no. 6, 2001.

[2] B. Karp and H. Kung, "GPSR: Greedy perimeter stateless routing for wireless networks," in *MobiCom '00: Proceedings of the ACM/IEEE International Conference on Mobile Computing and Networking*, 2000.

[3] F. Kuhn, R. Wattenhofer, Y. Zhang, and A. Zollinger, "Geometric ad-hoc routing: Of theory and practice," in *PODC '03: Proceedings of 22^{nd} ACM Int. Symposium on the Principles of Distributed Computing*, 2003.

[4] Q. Fang, J. Gao, L. Guibas, V. de Silva, and L. Zhang, "GLIDER: Gradient landmark-based distributed routing for sensor networks," in *INFOCOM '05: Proceedings of the 24th Conference of the IEEE Communication Society*, 2005.

[5] J. Bruck, J. Gao, and A. Jiang, "MAP: Medial axis based geometric routing in sensor networks," in *MobiCom '05: Proceedings of the ACM/IEEE International Conference on Mobile Computing and Networking*, 2005.

[6] B. Greenstein, D. Estrin, R. Govindan, S. Ratnasamy, and S. Shenker, "DIFS: A distributed index for features in sensor networks," in *Proceedings of First IEEE International Workshop on Sensor Network Protocols and Applications*, 2003.

[7] X. Li, Y. J. Kim, R. Govindan, and W. Hong, "Multi-dimensional range queries in sensor networks," in *Proceedings of the first international conference on Embedded networked sensor systems*, 2003.

[8] B. Bash and J. C. J. Byers, "Approximately uniform random sampling in sensor networks," in *DMSN '04: Proceedings of 1th Workshop on Data Management in Sensor Networks*, 2004.

[9] A. G. Dimakis, A. D. Sarwate, and M. J. Wainwright, "Geographic gossip: efficient aggregation for sensor networks," in *IPSN '06: Proceedings of the fifth international conference on Information processing in sensor networks*, 2006.

[10] A. Kröller, S. P. Fekete, D. Pfisterer, and S. Fischer, "Deterministic boundary recognition and topology extraction for large sensor networks," in *Proceedings of 17th ACM-SIAM Sympos. Discrete Algorithms*, 2006.

[11] T. K. Dey, J. Giesen, and S. Goswami, "Shape segmentation and matching with flow discretization," in *WADS '03: Proceedings of workshop on Algorithms and Data Structures*, 2003.

[12] T. Sebastian, P. Klein, and B. Kimia, "Recognition of shapes by editing shock graphs," in *ICCV '01: Proceedings of International Conference on Computer Vision*, 2001.

[13] K. Siddiqi, A. Shokoufandeh, S. J. Dickinson, and S. W. Zucker, "Shock graphs and shape matching," in *ICCV '98: Proceedings of International Conference on Computer Vision*, 1998.

[14] F. F. Leymarie and B. B. Kimia, "The shock scaffold for representing 3d shape," in *Proceedings of 4th International Workshop Visual Form*, 2001.

[15] M. Hilaga, Y. Shinagawa, T. Kohmura, and T. L. Kunii, "Topology matching for fully automatic similarity estimation of 3d shapes," in *SIGGRAPH '01: Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, 2001.

[16] Y. Wang, J. Gao, and J. S. B. Mitchell, "Boundary recognition in sensor networks by topological methods," in *MobiCom '06: Proceedings of the ACM/IEEE International Conference on Mobile Computing and Networking*, 2006.

[17] J. Elson, "Time synchronization in wireless sensor networks," Ph.D. dissertation, University of California, Los Angeles, 2003.

[18] S. Ganeriwal, R. Kumar, and M. B. Srivastava, "Timing-sync protocol for sensor networks," in *SenSys '03: Proceedings of the 1st international conference on Embedded networked sensor systems*, 2003.

[19] A. G. Dimakis, V. Prabhakaran, and K. Ramchandran, "Ubiquitous access to distributed data in large-scale sensor networks through decentralized erasure codes," in *IPSN '05: Proceedings of the fourth international symposium on information processing in sensor networks*, 2005.

[20] P. Skraba, Q. Fang, A. Nguyen, and L. Guibas, "Sweeps over wireless sensor networks," in *IPSN '06: Proceedings of the fifth international conference on Information processing in sensor networks*, 2006.