Boundary Recognition in Sensor Networks by Topological Methods

Yue Wang, Jie Gao
Dept. of Computer Science
Stony Brook University
Stony Brook, NY

Joseph S.B. Mitchell*
Dept. of Applied Math. and Statistics
Stony Brook University
Stony Brook, NY

ABSTRACT

Wireless sensor networks are tightly associated with the underlying environment in which the sensors are deployed. The global topology of the network is of great importance to both sensor network applications and the implementation of networking functionalities. In this paper we study the problem of topology discovery, in particular, identifying boundaries in a sensor network. Suppose a large number of sensor nodes are scattered in a geometric region, with nearby nodes communicating with each other directly. Our goal is to find the boundary nodes by using only connectivity information. We do not assume any knowledge of the node locations or inter-distances, nor do we enforce that the communication graph follows the unit disk graph model. We propose a simple, distributed algorithm that correctly detects nodes on the boundaries and connects them into meaningful boundary cycles. We obtain as a byproduct the medial axis of the sensor field, which has applications in creating virtual coordinates for routing. We show by extensive simulation that the algorithm gives good results even for networks with low density. We also prove rigorously the correctness of the algorithm for continuous geometric domains.

Categories and Subject Descriptors: C.2.1 [Network Architecture and Design]: Wireless communication; C.2.2 [Computer Systems Organization]: Computer-Communication Networks– *Network Protocols*; E.1 [Data]: Data Structures – *Graphs and networks*

General Terms: Algorithms, Theory.

Keywords: Boundary Detection, Shortest Path Tree, Sensor Networks.

1. INTRODUCTION

Wireless sensor networks are tightly coupled with the geometric environment in which they are deployed. On one hand, sensor network applications such as environment monitoring and data collection require sufficient coverage over the region of interest. On the other hand, the global topology of a wireless sensor network plays an important role in the design of basic networking functionalities, such as point-to-point routing and data gathering mechanisms. In this paper we study the problem of discovering the global geometry of the sensor field, in particular, detecting nodes on the boundaries (both inner and outer boundaries).

The viewpoint we take is to regard the sensor network as a discrete sampling of the underlying geometric environment. This is motivated by the fact that sensor networks are to provide dense monitoring of the underlying space. Thus, the shape of the sensor field, i.e., the boundaries, indicates important features of the underlying environment. These boundaries usually have physical correspondences, such as a building floor plan, a map of a transportation network, terrain variations, and obstacles (buildings, lakes, etc). Holes may also map to events that are being monitored by the sensor network. If we consider the sensors with readings above a threshold to be "invalid", then the hole boundaries are basically iso-contours of the landscape of the attribute of interest. Examples include the identification of regions with overheated sensors or abnormal chemical contamination. Holes are also important indicators of the general health of a sensor network, such as insufficient coverage and connectivity. The detection of holes reveals groups of destroyed sensors due to physical destruction or power depletion, where additional sensor deployment is needed.

Besides the practical scenario mentioned above, understanding the global geometry and topology of the sensor field is of great importance in the design of basic networking operations. For example, in the sensor deployment problem, if we want to spread some mobile sensors in an unknown region formed by static sensor nodes, knowing the boundary of the region allows us to guarantee that newly added sensors are deployed only in the expected region. A number of networking protocols also exploit geometric intuitions for simplicity and scalability, such as geographical greedy forwarding [2, 17]. Such algorithms based on local greedy advances would fail at local minima if the sensor networks have non-trivial topology. Backup methods, such as face routing on a planar subgraph, can help packets get out of local minima but create high traffic on hole boundaries and eventually hurt the network lifetime [2, 17]. This artifact is not surprising, since any algorithm with a strong use of geometry, such as geographical forwarding, should adhere to the genuine shape of the sensor field. Recently, there

*Partially supported by the National Science Foundation (CCF-0431030, CCF-0528209).

are a number of routing schemes that address explicitly the importance of topological properties and propose routing with virtual coordinates that are adaptive to the intrinsic geometric features [3, 7]. The construction of these virtual coordinate systems requires the identification of topological features first.

Our focus is to develop a distributed algorithm that detects hole boundaries. Thus, a centralized approach of collecting all of the information to a central server is not feasible for large sensor networks.

Prior and related work. Existing work on boundary detection can be classified into three categories by their major techniques: geometric methods, statistical methods and topological methods.

Geometric methods for boundary detection use geographical location information. The first paper on this topic, by Fang *et al.* [6], assumes that the nodes know their geographical locations and that the communication graph follows the unit-disk graph assumption, where two nodes are connected by an edge if and only if their distance is at most 1. The definition of holes in [6] is intimately associated with geographical forwarding such that a packet can only get stuck at a node on hole boundaries. Fang *et al.* also proposed a simple algorithm that greedily sweeps along hole boundaries and eventually discovers boundary cycles.

Statistical methods for boundary detection usually make assumptions about the probability distribution of the sensor deployment. Fekete *et al.* [9] proposed a boundary detection algorithm for sensors (uniformly) randomly deployed inside a geometric region. The main idea is that nodes on the boundaries have much lower average degrees than nodes in the "interior" of the network. Statistical arguments yield an appropriate degree threshold to differentiate boundary nodes. Another statistical approach is to compute the "restricted stress centrality" of a vertex v, which measures the number of shortest paths going through v with a bounded length [8]. Nodes in the interior tend to have a higher centrality than nodes on the boundary. With a sufficiently high density, the centrality of the nodes exhibits bi-modal behavior and thus can be used to detect boundaries. The major weakness of these two algorithms is the unrealistic requirement on sensor distribution and density: the average degree needs to be 100 or higher. In practice the sensors are not as dense and they are not necessarily deployed uniformly randomly.

There are also topological methods to detect insufficient sensor coverage and holes. Ghrist *et al.* [14] proposed an algorithm that detects holes via homology with no knowledge of sensor locations; however, the algorithm is centralized, with assumptions that both the sensing range and communication range are disks with radii carefully tuned. Kröller *et al.* [19] presented a new algorithm by searching for combinatorial structures called flowers and augmented cycles. They make less restrictive assumptions on the problem setup, modeling the communication graph by a quasi-unit disk graph, with nodes p and q definitely connected by an edge if $d(p, q) \leq \sqrt{2}/2$ and not connected if $d(p, q) > 1$. The success of this algorithm critically depends on the identification of at least one flower structure, which may not always be the case especially in a sparse network.

Towards a practical solution, Funke [10] developed a simple heuristic with only connectivity information. The basic idea is to construct iso-contours based on hop count from a root node and identify where the contours are broken. Under the unit-disk graph assumption and sufficient sensor density, the algorithm outputs nodes marked as boundary with certain guarantees. Specifically, for each point on the geometry boundary the algorithm marks a corresponding sensor node within distance 4.8, and each node marked as boundary is within distance 2.8 from the actual geometry boundary [11]. The simplicity of the algorithm is appealing; however, the algorithm only identifies nodes that are near the boundaries but does not show how they are connected in a meaningful way. The density requirement of the algorithm is also rather high; in order to obtain good results, the average degree generally needs to be at least 15.

Our contributions. We develop a practical distributed algorithm for boundary detection in sensor networks, using only the communication graph, and not making unrealistic assumptions. We do not assume any location information, angular information or distance information. More importantly, we do not require that the communication graph follows the unit disk graph model or the quasi-unit disk graph model. Indeed, actual communication ranges are not circular disks and are often quite irregularly shaped [13]. Algorithms that rely on the unit disk graph model fail in practice (e.g., the extraction of a planar subgraph by the relative neighborhood graph or Gabriel graph [18]). While the unit (or quasi-unit) disk graph assumption is often useful for theoretical analysis, it is preferable to consider algorithms that do not rely on this assumption or that degrade gracefully as the ground truth deviates only modestly from the model. Therefore, we do not put a hard restriction on the communication model in our algorithm. Rather, we use a loose notion of locality in wireless communications: Nearby nodes can communicate directly, and faraway nodes generally do not.

Our boundary detection algorithm is motivated by an observation that holes in a sensor field create irregularities in hop count distances. Simply, in a shortest path tree rooted at one node, each hole is "hugged" by the paths in a shortest path tree. We identify the "cut", the set of nodes where shortest paths of distinct homotopy types[1] terminate and touch each other, trapping the holes between them. The nodes in a cut can be easily identified, since they have the property that their common ancestor in the shortest path tree is fairly far away, at the other side of the hole. The detection of nodes in a cut can be performed independently and locally at each pair of adjacent nodes.

When there are multiple holes in the network (indicated by multiple branches of the cut), we can explicitly remove all of the nodes on cut branches except one, thereby connecting multiple holes into one. Our algorithm then focuses on finding the inner and outer boundaries of the network, which, with the cut nodes put back, will give the correct boundary cycles. In a network with only one hole (and one cut branch), one can easily find a hole-enclosing cycle. Indeed, for a pair of nodes that are neighbors across a cut (a "cut pair"), the concatenation of the paths from each node in a cut pair to their common ancestor gives such a cycle. This "coarse" boundary cycle is then refined to bound tightly both the inner boundary and outer boundary. In ad-

[1]Two paths have distinct homotopy types if one cannot be "continuously deformed" into the other.

dition to discovering boundary nodes, we also obtain their relative position information, so that we can connect them into a meaningful boundary. Our methods also readily provide other topological and geometric information, such as the number of holes (genus), the nearest hole to any given sensor, and the sensor field's medial axis (the collection of nodes with at least two closest boundary nodes), which is useful for virtual coordinate systems for load balanced routing [3].

We show by simulation that our algorithm correctly identifies meaningful boundaries for networks with reasonable node density (average degree 6 and above) and distribution (e.g., uniform). The algorithm also works well for non-uniform distributions. The algorithm is efficient. The entire procedure involves only three network flooding procedures and greedy shrinkage of paths or cycles. Further, as a theoretical guarantee, we prove that for a continuous geometric space bounded by polygonal obstacles – the case in which node density approaches infinity – the algorithm correctly finds all of the boundaries.

2. TOPOLOGICAL BOUNDARY RECOGNITION

Suppose a large number of sensor nodes are scattered in a geometric region with nearby nodes communicating with each other directly. Our goal is to discover the nodes on the boundary of the sensor field, using only local connectivity information. We propose a distributed algorithm that identifies boundary cycles for the sensor field.

The basic idea is to exploit special structure of the shortest path tree to detect the existence of holes. Intuitively, inner holes of the sensor field "disrupt" the natural flow of the shortest path tree: Shortest paths diverge prior to a hole and then meet after the hole. This phenomenon is also understood in terms of the continuous limit, when the shortest path tree becomes the "shortest path map", which we discuss in Section 3. We first outline the algorithm and then explain each step in detail.

1. Flood the network from an arbitrary node, r. Each node records the minimum hop count to r. This implicitly builds a shortest path tree rooted at r. We generally prefer to select r as a node on the outer boundary of the sensor field.

2. Determine the nodes that form the *cut*, where the shortest paths of distinct homotopy type meet after passing around holes. Informally, the nodes of a branch of the cut have their least common ancestor (LCA) relatively far away and their paths to the LCA well separated. See Figure 1(ii-iv). If there are multiple branches of the cut, corresponding to multiple holes, delete nodes on branches of the cut in order to merge holes, until there is only one composite hole left in the sensor field.

3. Determine a shortest cycle, R, enclosing the composite hole; R serves as a coarse inner boundary. See Figure 1(v).

4. Flood the network from the cycle R. Each node in the network records its minimum hop count to R. See Figure 1(vi).

5. Detect "extremal nodes" whose hop counts to R are locally maximal. See Figure 1(vii).

6. Refine the coarse inner boundary R to provide tight inner and outer boundaries. These boundaries are in fact cycles of shortest paths connecting adjacent extremal nodes. See Figure 1(viii).

7. Undelete the nodes of the removed cut branches and restore the real inner boundary locally.

8. At this stage we have a set of cycles corresponding to the boundaries of the inner holes and the outer boundary. As a byproduct, we can compute the medial axis of the sensor field. See Figure 1(ix).

2.1 Build a shortest path tree

The first step of the algorithm is to flood the network from an arbitrary root node r. For example, we can select r to be the node with smallest ID. This can be performed in a distributed fashion as follows. First, each sensor node p sets a timer with a random remaining time. Ties are broken by the unique IDs of the sensor nodes. When the timer of p reaches 0, the node p will begin to flood the network and build a shortest path tree $T(p)$. The message sent out by p contains the ID of p and the initial timer p chooses. The tree with a timer of minimum value starts first and suppresses the construction of other trees. When the frontier of $T(p)$ encounters a node q, there are two cases: (i) If q does not belong to any other tree, then q is included in the tree $T(p)$ and q will broadcast the packet; or, (ii) If q is already included in another tree $T(p')$, then the start time of $T(p)$ and $T(p')$ are compared. Only when the start time of $T(p)$ is earlier than that of $T(p')$ will the message from p be broadcast. Eventually the tree with earliest starting time will dominate and suppress the construction of other trees, and the packet from the node with minimum initial timer will cover the whole network. Each node in the network records the minimum hop count from this root. We denote by T the shortest path tree constructed.

When the network size is unknown, the flooding step provides a good approximation to the network diameter. By triangle inequality, the diameter of the network is at most $2d$, where d is the distance between the root r and the deepest node in T. d can be used to generate a reasonable threshold for the minimum size of the holes we aim to discover.

2.2 Find cuts in the shortest path tree

Hints about the presence of holes are hidden in the structure of the shortest path tree T. The "flow" of T forks near a hole, continues along opposite sides of the hole and then meets again past the hole. We detect where the shortest paths meet and denote those nodes as "cut" nodes (e.g., the red nodes in Figure 1(iv)). Nodes of the cut form *cut branches* and *cut vertices*, where three or more cut branches come together; cut branches are the discrete analogue of bisectors in the (continuous) "shortest path map" and cut vertices are the discrete "SPM-vertices"; see Section 3. Figure 3 shows the tree T for a network with 3 holes. In this case, we detect 3 groups of cut nodes (3 cut branches). In general, as will be proved later in the continuous case, if the sensor field has k interior holes and m cut vertices, there are exactly $k + m$ cut branches in the network.

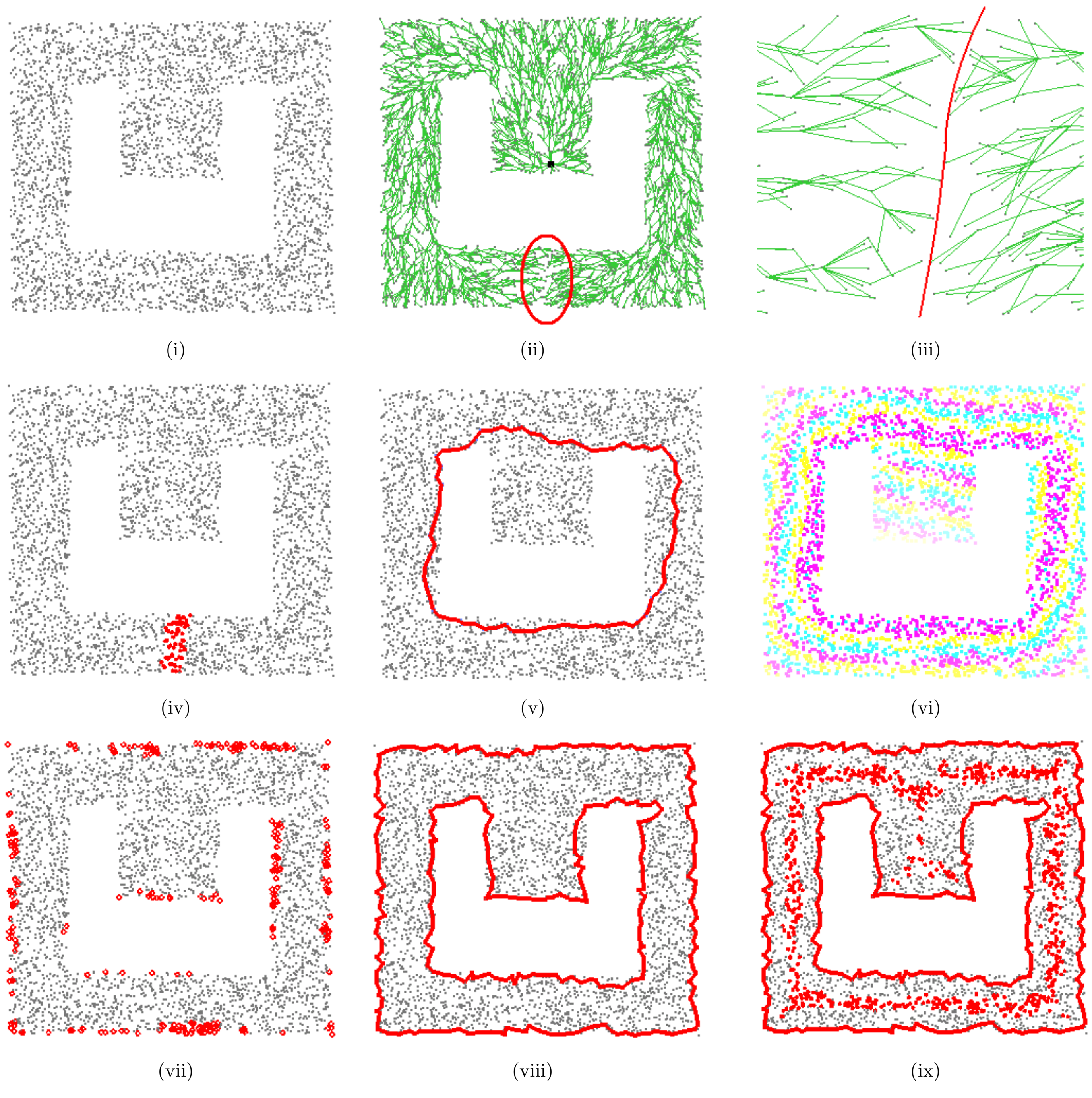


Figure 1: Boundary detection algorithm for an example with one concave hole. The average degree is about 20. (i) The sensor nodes; (ii) The shortest path tree T (green), rooted at the black node; (iii) The zoomed-in portion (within the loop of (ii)) of the tree where the shortest paths meet; (iv) The marked cut nodes (red); (v) The course inner boundary formed by shortest paths from a pair of cut nodes to their least common ancestor; (vi) The nodes colored by hop count (giving iso-contours) in a flooding from the coarse inner boundary; (vii) The nodes (red) with locally maximum hop count (extremal points) from the coarse boundary; (viii) The refinement of the coarse inner boundary, giving tight inner and outer boundaries; (ix) The medial axis nodes of the sensor field, obtained as a by-product of boundary detection.

Intuitively, the nodes in a cut are the neighboring pairs, (p, q), in the communication graph whose least common ancestor, LCA(p, q), in the shortest path tree T is "far" from p and q, with paths from LCA(p, q) to p and to q "well separated." More formally,

Definition 1. *A* cut pair (p, q) *is a pair of neighboring nodes in the network satisfying the following conditions: (i) The (hop) distance between p or q and y =LCA(p, q) is above a threshold δ_1; and, (ii) The maximum (hop) distance between a node on the path in T from p to y and the path in T from q to y is above a threshold δ_2 (See Figure 2). The* cut *is the union of nodes that belong to a cut pair. Each connected component of the cut is a subgraph which, when thinned, is a* cut tree*; the nodes of degree greater than 2 in this tree are* cut vertices*, and the paths of degree-2 vertices, and their neighboring (non-cut-vertex) cut nodes, form the* cut branches.

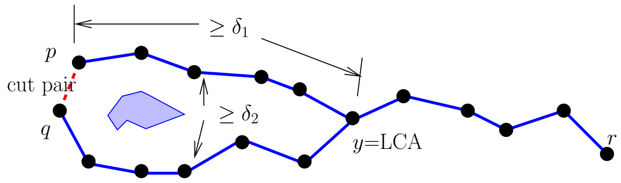


Figure 2: Definition of a cut pair (p, q).

The two parameters in the definition of a cut pair, δ_1 and δ_2, specify the minimum size of the holes we want to detect. Typically, we choose them as some constant fraction of the diameter of the sensor field (e.g., experiments reported in this paper use $\delta_1 = \delta_2 = 0.18d$). The condition on whether a node belongs to a cut can be checked locally. Alstrup *et al.* gave a distributed algorithm to compute the LCA [1]. The idea is to label the nodes of a rooted tree with $O(\log n)$ bits such that by using only the labels of two nodes one can calculate the label of their LCA. With this labeling, each pair of neighbors in the network check for their common ancestors. If the LCA is more than distance δ_1 away, we check if two shortest paths from these two neighbor nodes to their LCA satisfy the second condition in the definition above. This can be implemented by a local flooding from each node in the cut pair up to distance δ_2. Nodes that satisfy both conditions mark themselves as being in the cut.

The nodes in the cut will then connect themselves into connected components. Furthermore, each connected component agrees on the node closest to the root (ties are broken by the smaller ID). Specifically, each node u in the cut keeps its current knowledge of the ID of the node with smallest distance to the root. If a cut node u receives a message from its neighboring cut node about a closer node, u updates its knowledge and sends the change to its neighboring cut nodes. Eventually, the cut nodes connect themselves into connected components with each cut identified by the ID of the closest node. If there is no hole in the network, there will be no cut, then no connected components correspondingly. By simulation we find that this algorithm correctly finds all the cuts when the sensor field has a reasonable density (e.g., when the average degree is about 7 or more).

When there are multiple holes and multiple cuts in the network, we artificially merge the holes by removing nodes on cut branches, until there is only one composite hole left. As will be clear later, the real boundary cycles can be easily

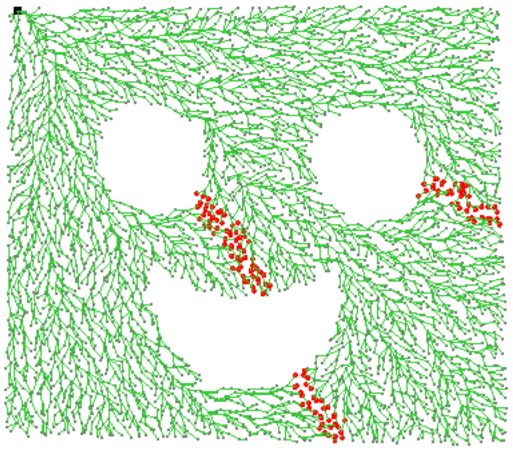


Figure 3: The cut pairs in a multi-hole example. 4050 nodes and the average degree is 10.

restored by undeleting the removed cut nodes. We remove all of the nodes on cut branches except the one branch furthest away from the root. The interior holes either connect to themselves or connect to the outer boundary. Thus, the multi-hole case is turned into a single hole scenario; thus, in the following steps we focus on the single hole case.

2.3 Detect a coarse inner boundary

With the cut nodes detected, we would like to find a coarse inner boundary R that encloses the (composite) interior hole. R is then refined to provide tight inner and outer boundaries.

Definition 2. *A* coarse inner boundary R *is a shortest cycle enclosing the interior hole in the sensor field.*

Recall that we have turned any multi-hole sensor field into a single composite hole sensor field by removing all cut branches except one. All the nodes in the unique cut branch have the ID of the node closest to the root. Thus, the closest node p, together with its partner q in the cut pair, will find the shortest paths between them that do not go through any cut node. This can be implemented in two ways. An obvious way is to use any shortest path algorithm to find this path. In order to prevent the path from going through any cut nodes, we remove all the edges between cut pairs. Alternatively, we can use the two shortest paths from p and q to LCA(p, q). Together with the edge pq, we obtain a cycle that encloses the hole. This cycle is not necessarily the shortest cycle. But we can greedily shrink it to be as tight as possible, by the following k-hop shrinking process. For any two nodes that are within k hops on the cycle, we check whether there exists a shorter path between these two nodes. If so, we use the shorter path to replace the original segment on the cycle and shorten the total length. For example, a 2-hop shrinking checks, for every three adjacent nodes on the cycle, say x, y, z, whether (x, z) is an edge. If so, we shrink the cycle by excluding y. In a sensor field with reasonable density, the greedy process stops at the shortest cycle so that no more improvement is made. For a sensor field with only one convex shape hole, the coarse inner boundary actually is the real inner boundary, as in Figure 4(i). If the graph has a concave hole (Figure 1(v)), the coarse inner boundary R will be a shortest circuit containing this concave hole and, thus, approximates its convex hull. A multi-hole example is shown in Figure 4(ii). Notice that this coarse inner boundary provides a consistent ordering of the nodes on this cycle.

We note that the shortest paths from a cut pair to their LCA do not go through any cut nodes, as proved later (Section 3). Thus, the removal of cut nodes in a multi-hole scenario is not a problem for the discovery of the coarse inner boundary by the greedy shrinking scheme.

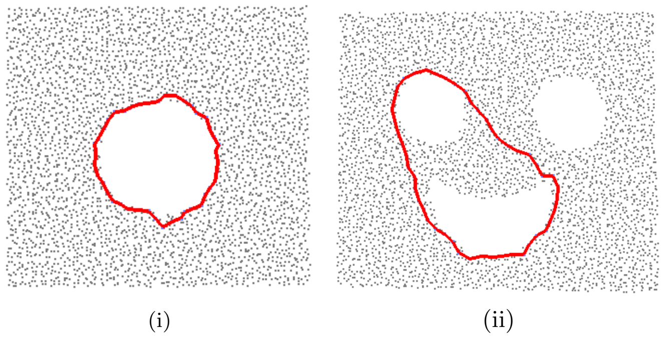


Figure 4: The coarse inner boundary for (i) a convex hole scenario; (ii) a multi-hole scenario with all but one cut branch removed, one interior hole connected to the outer boundary.

2.4 Find extremal nodes

The coarse inner boundary R is not tight in the case of a concave hole. Also, we have no idea about the outer boundary. Next, we refine the coarse inner boundary to provide tight cycles for both inner and outer boundaries. This refinement is through finding what are called "extremal nodes" with respect to R.

Definition 3. *An* extremal node *is a node whose minimum hop count to nodes in R is locally maximal.*

To discover extremal nodes, we have the nodes on R synchronize among themselves and start to flood the network at roughly the same time [5] [12]. Each node in the network records the minimum hop count to nodes in the coarse inner boundary R. This is as if we merge the nodes in R to a dummy root σ, and build a shortest path tree $T(\sigma)$, rooted at σ for the whole network. The extremal nodes are the ones with locally maximum distance to R. Each extremal node can detect itself by checking its direct neighbors. Intuitively, the extremal nodes are on the outer boundary or are the ridges on the real inner boundary of a concave hole (Figure 1(vii)).

With the extremal nodes identified, we also need to know their relative orderings to connect them to a consistent boundary. This ordering can be derived from the ordering of the nodes on R. Specifically, for each node on R, we assign a label in $[0, L]$ that indicates its position on R. A node u that is k hops away from R may have multiple neighbors $(k-1)$ hops away from R. Thus, the label of u, $\ell(u)$, is taken as the average of the labels of all of its neighbors that are $(k-1)$ hops away from R. Each node in the network records the distance to R as well as its label. The relative ordering of extremal nodes is decided by their labels.

When there are many extremal nodes and some of them have similar labels, we would like to eliminate some of them and take a few representative nodes for the next step. The goal for this clean-up is to take sampled extremal nodes so that we can easily sort them. We first find the connected components for all extremal nodes, denoted as extremal connected components (ECC). These can be done with the same distributed strategy as used to find connected components for the cuts. Next, we select at most two nodes as the representatives for each ECC (an ECC may contain only one node). These two nodes have the greatest distance between them in their affiliated ECC. These nodes will be connected according to their labels to refine inner and outer boundaries.

2.5 Find the outer boundary and refine the coarse inner boundary

With the extremal nodes and a linear ordering of them, we would like to connect them into a cycle consistent with this ordering. Notice that here we only consider the case with one inner hole, since we have removed the cuts to connect multiple holes into a composite hole. If this composite hole is convex, then all extremal nodes belong to the outer boundary. If the composite hole is concave, as in the multi-hole case, there will be two types of extremal nodes, those on the outer boundary and those on the interior of R. We use the following rule to differentiate extremal nodes on different sides of R.

The removal of R, together with all of the 1-hop neighbors of R, partitions the network into several connected components C_i, $i = 1, \ldots, w$. Now we would like to connect the extremal points into extremal paths in each connected component, which will then be further connected into boundary cycles. We first focus on a particular connected component and describe how an extremal path is constructed. Specifically, all of the nodes that are 2 hops away from R will connect themselves through local flooding. This results in a path (or a cycle) P_j with a corresponding maximum index and minimum index along R. Now we would like to refine this path P_j so as to force it to go through the extremal nodes in this connected component as well as the two nodes on P_j with maximum and minimum indexing. Notice that each extremal node has shortest paths from some nodes on R. Now we connect two extremal nodes with adjacent ordering by the shortest path between them. To find the shortest path between two extremal points, again we can either use any shortest path algorithm or a greedy approach. For the latter, we notice that there is a natural path between any two extremal nodes u, v, composed by the shortest path from u to its closest point on P_j, the shortest path from v to its closest point on P_j and a segment of P_j between the correspondences of u, v. The shortest paths from extremal points to P_j can be found by following the parent pointer towards R (in the tree $T(\sigma)$). Then, we can greedily refine this path and find the shortest path between u and v.

By the above procedure, the extremal nodes are connected into a path P_j, in each connected component. If one of the paths is actually a cycle, then it is already a boundary cycle. If there are paths that are not connected into cycles, then we tour the coarse boundary R and connect them into boundary cycles. In particular, we start from anywhere on

R and tour along R; when there is a neighboring node (in 2 hops) who is on an extremal path P_j, then we branch on P_j. This will eventually come back and close a cycle. Then, we tour R again, from a node that is not on the first cycle, and branch on extremal paths as long as we can. This will generate another cycle. So far, we classify extremal nodes and connect them into two cycles, corresponding to the inner and outer boundary. Figure 5 shows the results for two examples after this stage.

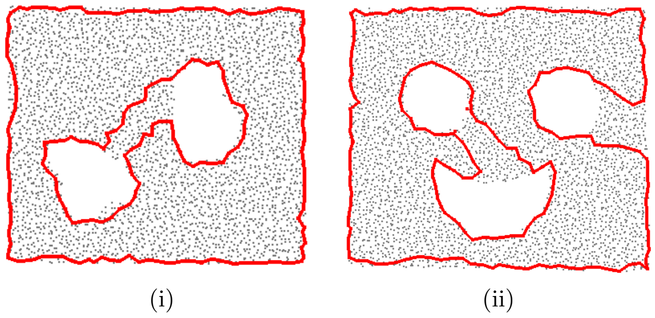


Figure 5: The outer boundary and the refined inner boundary. (i) 2-hole example; (ii) multi-hole example.

As will be shown in Section 3, in the continuous case all of the boundary points are identified as extremal points. However, in a discrete network, the shortest path is computed on a combinatorial graph. Thus, not all of the boundary nodes are identified as extremal nodes. The boundary refinement can be performed in an iterative fashion such that we flood from the current boundary cycle and identify more extremal points until the boundary cycle is sufficiently tight.

2.6 Restore the inner boundary

The final step of our method is to recover the inner holes in the sensor field and find their boundaries. We undelete the cut nodes we removed earlier and restore the correct inner boundary. For each cut, we find a cut pair (p, q) such that the inclusion of edge pq in the refined inner boundary R partitions the inner boundary into two boundary cycles. If p, q are by themselves not on the refined inner boundary R, then we do a local search from p, q to discover nodes on R. The two new boundary cycles will share nodes p, q. Then, we shrink the cycles locally to make them tight. Here, the shrinking procedure has a restriction that the shrunk path still has to pass through the extremal nodes. Thus, we partition the refined inner boundary into a number of cycles, each representing the boundary of an inner hole. Figure 6 shows our final results for two examples.

2.7 The medial axis of the sensor field

The boundaries of the sensor field capture the global geometric shape information, and thus can be used to generate other structures related to the global geometry. For example, we can define the quasi-Voronoi diagram for the boundaries such that sensor nodes are classified by their closest boundary. In another example, we can compute the medial axis of the sensor field. The medial axis is defined as the set of nodes with at least two closest boundary nodes[2]. The

[2]Since hop counts are discrete, we allow a node on the medial axis with two closest boundary nodes that differ by 1 in hop count.

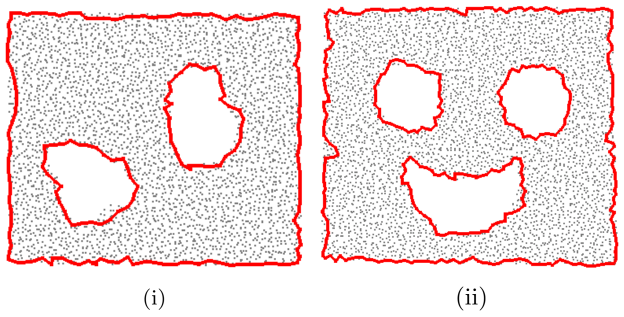


Figure 6: The boundary cycles created by the connection of nodes: (i) 2-hole example; (ii) multi-hole example.

medial axis can be used to generate virtual coordinates for efficient greedy routing [3]. Both the quasi-Voronoi diagram and the medial axis can be discovered by a simple flooding from the boundary cycles. Figure 7 shows the results for our 3-hole face example.

With our boundary information, we start from all boundary nodes and flood the network simultaneously. Each node thus records its closest boundary. All of the nodes with the same closest boundary are naturally classified to be in the same cell of the quasi-Voronoi diagram. Further, the nodes at which the frontiers collide belong to the medial axis. Specifically, the detection of nodes on the medial axis can be done locally. Denote the boundary cycles by R_i, $i = 1, \ldots, k$. During the flooding from the boundaries, each node in the boundary sends out a packet containing the boundary containing it, its position based on the originator boundary, and the number of hops this packet has traveled, as denoted by a tuple $(i, \ell_i(v), h_i(v))$. A node, upon receiving a packet $(i, \ell_i(v), h_i(v))$, does one of the following things. If a node has not received earlier packets with smaller hop counts to boundaries, it records $(i, \ell_i(v), h_i(v))$, where $h_i(v)$ is the minimum hop count distance to the originator of the packet, drops earlier tuples with larger hop counts, and retransmits the packet. In the end, each node only contains the tuples with the minimum hop count distance. There are three possible cases: (1). Only one minimum hop count tuple is left for the node, so the node is not a medial axis node. (2). More than one minimum hop count tuple is left, and these tuples are generated by different boundary cycles, which means the node has more than one closest boundary node, so the node is on the medial axis. (3). More than one minimum hop count tuple is left, and these tuples are generated by the same boundary cycle, so we need to compare the ℓ values in these tuples; if they differ a lot, then the node is on the medial axis. This case corresponds to an inward convex (reflex) segment in the boundary.

3. PROOF OF CORRECTNESS IN THE CONTINUOUS CASE

In this section we prove rigorously that the algorithm will correctly detect all the inner and outer boundaries in a continuous domain with polygonal boundaries. (The results apply also to non-polygonal domains, since they are approximated by polygons.) Let $\mathcal{F}$ be a closed polygonal domain in the plane, with k simple polygonal obstacles inside; $\mathcal{F}$ is a simple polygon P, minus k disjoint polygonal holes. We refer

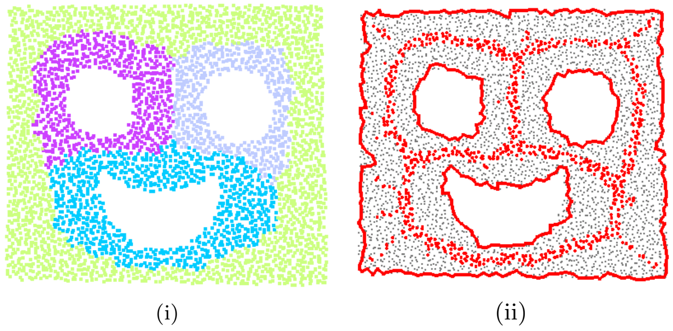


Figure 7: (i) A quasi-Voronoi diagram; (ii) the medial axis diagram.

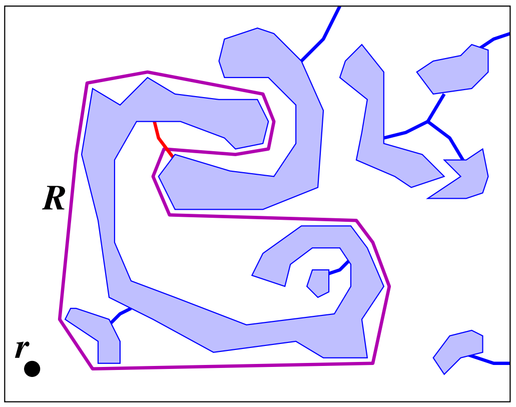


Figure 8: A shortest path map SPM(r) for $\mathcal{F}$ with $k = 8$ obstacles (blue), showing the bisector set $\mathcal{B}$ (red), which has one SPM-vertex, 7 connected components, and 9 arcs. R is the shortest cycle corresponding to the red bisector arc. Other bisector arcs are blue, having become part of the obstacle set for $\mathcal{F}'$.

to $\mathcal{F}$ as the free space and its complement, $\mathcal{O}$, as the obstacle space. $\mathcal{O}$ consists of k open, bounded simple polygonal holes and an unbounded complement of the simple polygon P. Denote by V the set of all vertices of $\mathcal{F}$.

For any two points $p, q \in \mathcal{F}$, we denote by $g(p, q)$ the geodesic shortest path in $\mathcal{F}$ between p, q. The Euclidean length of $g(p, q)$ is denoted by $d(p, q)$. The shortest path $g(p, q)$ is not necessarily unique. In fact, our algorithm aims to detect points with one or more shortest paths to the root. Rigorously, given a root $r \in \mathcal{F}$, the shortest path tree at r is the collection of shortest paths from each point in $\mathcal{F}$ to r. A geodesic shortest path is a polygonal path with turn points at vertices of $\mathcal{F}$ [4]. We say that a vertex $s \in V$ is a *parent* of a point $p \in \mathcal{F}$ if for some geodesic shortest path $g(r, p)$, s is the last vertex along the path $g(r, p) \setminus \{p\}$ at which $g(r, p)$ turns. If the geodesic path is a straight line from r to p, then r is the parent of p. The free space $\mathcal{F}$ can be partitioned to maximal regions, called *cells*, such that all the points in the same cell have the same parent or set of parents. This partition is called the *shortest path map*, SPM(r) (see Figure 8). We define the *bisector*, $\mathcal{C}(v_i, v_j)$, of two vertices v_i, v_j as the set of points in $\mathcal{F}$ with the set of parents $\{v_i, v_j\}$. A point in $\mathcal{F}$ with three or more parents is called an *SPM-vertex*. The union of all bisectors and SPM-vertices, $\mathcal{B}$, is often called the *cut locus*. Our boundary detection algorithm makes use of the properties of the shortest path map. We list below the useful properties that are proved in [20].

Lemma 4 ([20]). *Given a closed polygonal region $\mathcal{F}$ in the plane, with k simple polygonal obstacles inside. The shortest path map at an arbitrary root $r \in \mathcal{F}$ has the following properties.*

1. *Each bisector is the union of a finite set of closed subarcs of a common hyperbola (a straight line is a degenerate case of a hyperbola).*

2. *The collection of bisectors and SPM-vertices, denoted by $\mathcal{B}$, forms a forest.*

3. *There is at least one bisector point on the boundary of each obstacle.*

4. *$\mathcal{F} \setminus \mathcal{B}$ is simply connected.*

Lemma 5. *For a region $\mathcal{F}$ with k polygonal holes inside and m SPM-vertices, there are exactly $k + m$ bisector arcs.*

Proof. This follows from the fact that a tree of v vertices has $v-1$ edges, where, here, $v = k+1+m$ since the bisector arcs connect the k obstacle boundaries, the m SPM-vertices, and the 1 outer boundary. □

In fact, our boundary detection algorithm finds the bisector arcs. For completeness, we re-state the outline of the boundary detection algorithm for the continuous case. The focus is on the correctness of the algorithm so we explain it in a centralized setting. The boundary detection algorithm for $\mathcal{F}$ works as follows.

1. Find the bisectors and SPM-vertices; each of them has at least two geodesic shortest paths to the root r.

2. Delete bisector arcs until there is only one bisector arc left. Correspondingly, we connect $k-1$ holes into one composite hole. This results in a new domain $\mathcal{F}'$ with only one interior hole O' and one bisector arc.

3. Find the shortest cycle R in $\mathcal{F}'$ enclosing the inner hole O'.

4. Refine R to find both the inner boundary and outer boundary of $\mathcal{F}'$. In particular, we use the following algorithm for boundary refinement.

   (a) Compute the Voronoi diagram of R in $\mathcal{F}'$. Find the *extremal points*, defined as the points that do not stay on the interior of any shortest paths from points of $\mathcal{F}'$ to its closest point on R, denoted by E. Furthermore, we find for each extremal point the closest point(s) on R.

   (b) Order the extremal points by the sequence of their closest point(s) on R. The extremal points are naturally connected into paths or cycles. Tour the coarse boundary R and replace the segments of R by their corresponding extremal paths. Touring the boundary twice will come up with two cycles that correspond to the inner and outer boundaries.

5. Undelete the removed bisector arcs. Restore the boundaries of interior holes and the outer boundary.

Next we will prove that this algorithm correctly finds all the boundaries of $\mathcal{F}$. The proof consists of a number of lemmas.

Lemma 6. *The domain $\mathcal{F}'$ has one interior polygonal hole.*

Proof. This is due to the fact in Lemma 4 that $\mathcal{F} \setminus \mathcal{B}$ is simply connected. Thus, by removing all but one bisector arc and all SPM-vertices, we obtain a polygonal region $\mathcal{F}'$ with one interior hole. □

Now we focus on $\mathcal{F}'$ and argue that we indeed find the correct outer and inner boundaries of $\mathcal{F}'$ by the iterative refinement. We only prove for the outer boundary R^+; the correctness of the inner boundary R^- can be proved in the same way. By the refinement algorithm, we can find the Voronoi diagram of R by a wavefront propagation algorithm (e.g., [15]). The extremal points are the points that do not stay on the interior of any shortest paths from points of $\mathcal{F}'$ to its closest point on R. Due to the properties of wavefront propagation, the extremal points must be either on the boundary or on the medial axis, where wavefronts collide [16, 20]. However, since cycle R is a shortest path cycle surrounding the hole, we argue that the extremal points on the exterior of R must stay on the boundary of $\mathcal{F}'$.

Lemma 7. *The extremal points are exactly the boundary points of $\mathcal{F}'$.*

Proof. First we argue that all the inward concave vertices of R must be on the outer boundary of $\mathcal{F}'$. By the properties of geodesic shortest paths [21], all the vertices of R must be vertices of $\mathcal{F}'$. If an inward concave vertex is a vertex of the inner hole, then we can move the concave vertex outward and shrink the cycle R, as shown in Figure 9(i) (the vertex of the inner hole w can not be on R). This is a contradiction.

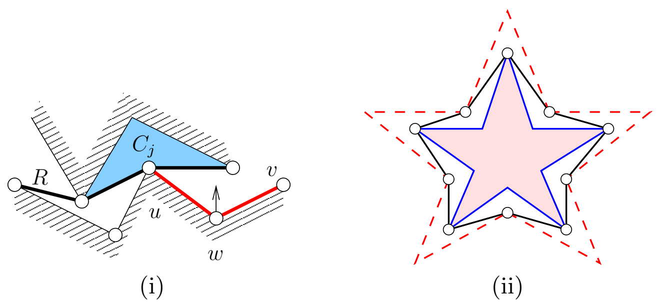


Figure 9: (i) All the inward concave vertices of R must be on the outer boundary of $\mathcal{F}'$. (ii) The dark cycle is the coarse inner boundary R. The cycles in blue and (dashed) red are inner and outer boundaries.

Thus, the removal of cycle R partitions the space $\mathcal{F}'$ into disjoint components $\{C_j\}$. Correspondingly, we denote by R_j the segment of R that bounds the component C_j. By the above argument, each boundary segment R_j is inward concave. The wavefront propagation of R is actually composed of Voronoi wavefront propagation of each segment of R_j to its bounded region C_j. Since each segment is concave and the C_j's are disjoint, the wavefront propagation does not collide. All the extremal points can only happen at wavefront-boundary collision. In fact, all the boundary points collide with the wavefront propagation and thus are considered as extremal points. □

At this point we can consider R to be a coarse approximation to the outer (and inner) boundary. By the refinement algorithm we will improve the approximation and obtain the correct outer boundary. Specifically, the removal of R from $\mathcal{F}'$ partitions the space $\mathcal{F}'$ into disjoint components $\{C_j\}$, $j = 1, \ldots, w$. For each connected component C_j, the extremal points connect themselves into either a path (with open endpoints) or a cycle P_j.

- If $w = 1$, the extremal points form a cycle. R is already the inner boundary. All the extremal points are on the outer boundary.
- If $w \geq 2$ and one of P_j is a cycle, this cycle is the outer boundary. The rest of the extremal points refine R to the inner boundary. Specifically, we replace the segments on R by their extremal paths.
- Otherwise, we will tour R and discover the inner and outer boundaries. Specifically, we travel along R and always prefer to branch on an extremal path. When we close the cycle, we obtain a boundary cycle R_1. Then we visit R again, starting at an extremal point that is not on R_1. Again we prefer to branch on extremal paths. This will give a different cycle. The two cycles are inner and outer boundaries. For an example, see Figure 9 (ii).

With the correct inner and outer boundaries of $\mathcal{F}'$, we are now ready to recover the real boundaries of $\mathcal{F}$. We simply put back the deleted bisectors. Indeed, the boundaries of $\mathcal{F}'$ are the boundaries of $\mathcal{F}$ plus all the bisectors and SPM-vertices. Thus we remove the bisectors from the boundaries of $\mathcal{F}'$ and obtain $k + 1$ cycles that are the real boundaries.

Theorem 8. *Given a closed polygonal domain $\mathcal{F}$ in the plane, with k simple polygonal obstacles inside, the boundary detection algorithm will find the boundary of the region correctly.*

4. SIMULATIONS

We performed extensive simulations in various scenarios, with the goal to evaluate the performance of the algorithm with respect to the network topology, node density and distribution, etc. We particularly note that our method works well even in cases of very low average degree, such as less than 10, or even as low as 6 in some models. Degree 6 has been shown to be optimal for mobile networks [22].

4.1 Effect of node distribution and density

For each figure in this part, we show a pink circle in the upper left corner to illustrate the communication range of the sensor field.

4.1.1 *Random distribution of sensors*

In this group of experiments, we randomly place, according to a uniform distribution, 3500 nodes in a square region with one hole. The average degree of the graph is varied by adjusting the communication radius. Figure 10 shows the results using our method. As expected, as the average degree of the network increases, the performance of the algorithm improves.

The unsatisfactory result in Figure 10(i) is due to insufficient connectivity. The average degree is 7; however, the random distribution tends to have clustered nodes and holes. Thus, in sparse regions some nodes are incorrectly judged to be extremal, and the final outer boundary is then required to pass through these mistaken extremal nodes. When the average degree reaches 10, the communication graph is better connected, and the results improve.

For comparison, SHAWN [19] did not find a starting solution (a "flower") for cases (i) and (ii); it found apparently correct boundaries for cases (iii) and (iv), indistinguishable from our results. The method of Funke and Klein [11] had difficulty with all 4 cases; see Figure 11(i), (ii).

Connectivity is necessary for computing the shortest path tree and determining cuts, etc. In fact, this low-degree graph with insufficient connectivity is the major troubling issue for prior boundary detection methods. Since our method only requires the communication graph, we can apply some simple strategies to increase artificially the average degree. For a disconnected network, we use the largest connected component of the graph to build our shortest path tree. Then we artificially enlarge the communication radius by taking 2-hop/3-hop neighbors as "fake" 1-hop neighbors. In this way, the connectivity of the graph will be ameliorated and the results will be improved correspondingly. Figure 12 shows the improvement by this simple strategy. The result using 3-hop neighbors has fewer incorrectly marked extremal nodes, and the final boundary is in good shape except that the boundary cycle is not very tight. This is understandable since we make the communication range artificially larger, so that more nodes are "on" the boundary now.

4.1.2 *Grid with random perturbation*

In this simulation, we put about 3500 nodes on a grid and then perturbed each point by a random shift. In particular, for each original grid node we create two random numbers modulo the length and the width of each block of the grid, the use these two small numbers to perturb the positions of the nodes. This distribution may be a good approximation of manual deployments of sensors; it also gives an alternative means of modeling "uniform" distributions, while avoiding clusters and "holes" that can arise from the usual continuous uniform distribution or Poisson process. Refer to Figure 13. Our method gives very good results for graphs with average degree 6 or more.

SHAWN [19] found good boundaries for (ii)-(iii), but did not find a starting solution for (i). The method of Funke and Klein [11] did well for case (iii) but less well for the lower degree cases; see Figure 11(iii).

4.1.3 *Low density, sparse graphs*

In this group of experiments (Figure 14), we scatter nodes in a square region with one hole. In order to guarantee good connectivity, the nodes are distributed on a randomly perturbed grid. The leftmost image of Figure 14 has about 3500 nodes; then, from left to right, each example has about 1000 fewer nodes, becoming more and more sparse. Our experiments show that if we fix the communication radius, and decrease the density of nodes, our method performs well, even for low density or sparse graphs (Figure 14(iv)), as long as the average degree is at about 7 or more.

SHAWN [19] did not find a starting solution for case (iv); it found good boundaries for cases (i)-(iii). The method of Funke and Klein [11] also performs well in cases (i)-(iii), but it performs less well in the lowest-degree case (iv); see Figure 11(iv).

4.2 Further examples

In Figure 15, we illustrate more results for a variety of more intricate geometries, including a spiral shape, a floorplan, etc.

5. DISCUSSION

While our new boundary detection algorithm provably finds boundaries in the continuous case (Section 3), in discrete sensors networks several implementation issues arise.

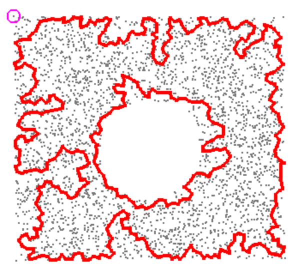
First, even for a given homotopy type, there need not be a unique shortest path between two nodes. Thus, the boundary cycle discovered by our algorithm, as shown in the simulations, may not tightly surround the real boundaries. Currently, we have two approaches to improve it. One is to make use of the fact that the nodes with lower degree are more likely to be on the boundary; thus, we implemented a preferential scheme for low-degree nodes when computing shortest paths. Another approach is to use an iterative method to find more extremal nodes, and then refine the boundary; this can also help to address the issue that several extremal points may have the same positions because we use hop counts to approximate true distances.

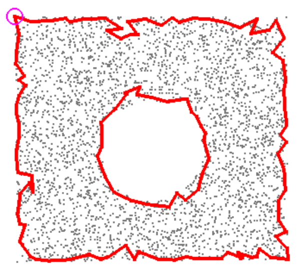
Second, deciding the correct orderings of the extremal nodes requires some care. In the continuous case, extremal nodes project to their nearest node in the rough inner boundary, resulting in a consistent ordering of extremal nodes, as shown in Section 3. In the discrete case, since we use hop count to approximate the true distance, it is possible that different extremal points are mapped to the same position on the inner boundary, obscuring their ordering. Again, by using an iterative procedure, we delete all the extremal nodes with duplicate positions except one and then iteratively find more extremal points and refine the boundary gradually.

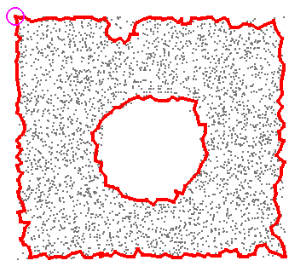
In practice, the sensor nodes often know some partial location information or relative angular information. Such positional information can help to improve the performance of our boundary detection algorithm. For example, if the nodes have knowledge of a universal north direction, it is easier to distinguish the extremal nodes in the interior and exterior of rough boundary. Also, if we have estimated distance or other rough localization information, other than pure hop count, the procedure to find shortest paths will become more reliable.

Finally, our method discussed until now assumes a sensor field with holes. We remark that the case with no holes can be solved as well. If a network has no hole, we discover this fact, since the network has no cut. In addition, the rough inner boundary degenerates to the root node. Then, we can discover the extremal nodes and connect them to find the outer boundary.

Acknowledgements. We thank Stefan Funke and Alexander Kröller for running comparison experiments with their algorithms on our data. We thank the reviewers for numerous helpful suggestions that improved the presentation.


(i)


(ii)


(iii)

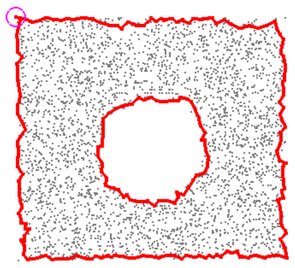
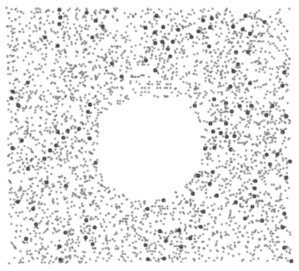
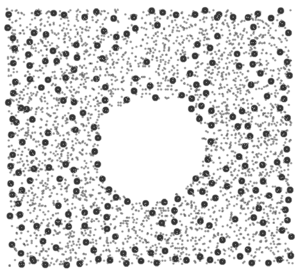
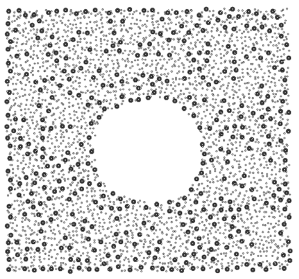

(iv)

Figure 10: Uniformly distributed sensor field. (i) the average degree is 7; (ii) the average degree is 10; (iii) the average degree is 13; (iv) the average degree is 16.


(i)


(ii)


(iii)

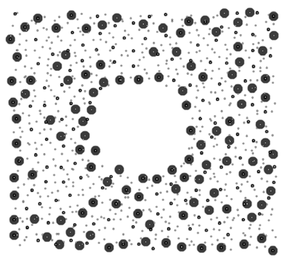
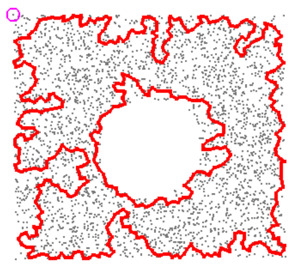
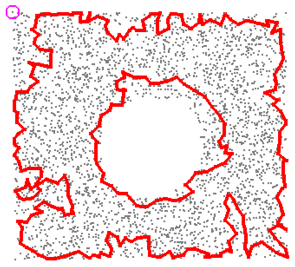

(iv)

Figure 11: Results of [11]: (i) uniform distribution, average degree 7; (ii) uniform, average degree 16; (iii) perturbed grid, average degree 8; (iv) sparse example (842 nodes), average degree 7.


(i)


(ii)

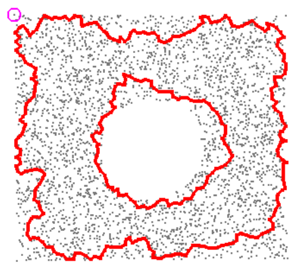
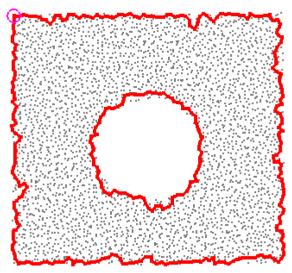
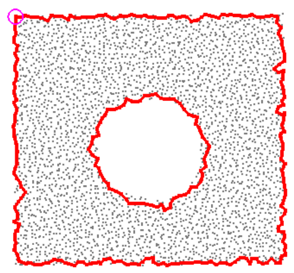

(iii)

Figure 12: Using 2-hop/3-hop neighbors as fake 1-hop neighbors to improve the performance in the low average degree case. (i) based on the original neighbors; (ii) using 2-hop neighbors as fake 1-hop neighbors; (iii) using 3-hop neighbors as fake 1-hop neighbors.


(i)


(ii)

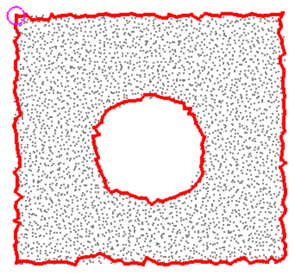

(iii)

Figure 13: Results for randomly perturbed grids. (i) the average degree is 6; (ii) the average degree is 8; (iii) the average degree is 12.

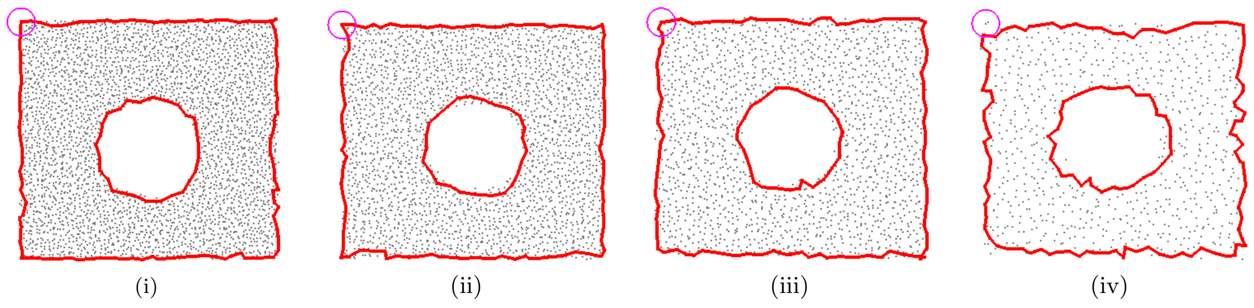


Figure 14: Results when the density of the graph decreases. (i) 3443 nodes and the average degree is 35; (ii) 2628 nodes and the average degree is 25; (iii) 1742 nodes and the average degree is 16; (iv) 842 nodes and the average degree is 7.

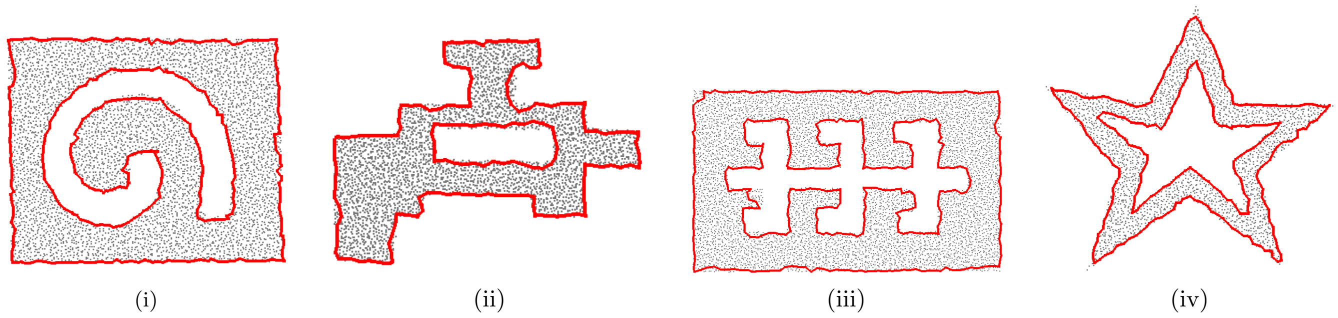


Figure 15: Results for more interesting examples. (i) A spiral shape with 5040 nodes and the average degree is 21; (ii) A building floor shape with 3420 nodes and the average degree is 20; (iii) A cubicle shape in an office with 6833 nodes and the average degree is 17; (iv) A double star shape with 2350 nodes and the average degree is 17.

6. REFERENCES

[1] S. Alstrup, C. Gavoille, H. Kaplan, and T. Rauhe. Nearest common ancestors: a survey and a new distributed algorithm. In *Proc. 14th ACM Sympos. Parallel Algorithms and Architectures*, pp. 258–264, 2002.

[2] P. Bose, P. Morin, I. Stojmenovic, and J. Urrutia. Routing with guaranteed delivery in ad hoc wireless networks. *Wireless Networks*, 7(6):609–616, 2001.

[3] J. Bruck, J. Gao, and A. Jiang. MAP: Medial axis based geometric routing in sensor networks. In *Proc. ACM/IEEE Internat. Conf. on Mobile Computing and Networking (MobiCom)*, pp. 88–102, 2005.

[4] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf. *Computational Geometry: Algorithms and Applications*. Springer-Verlag, Berlin, 1997.

[5] J. Elson. *Time Synchronization in Wireless Sensor Networks*. PhD thesis, University of California, Los Angeles, May 2003.

[6] Q. Fang, J. Gao, and L. Guibas. Locating and bypassing routing holes in sensor networks. In *Proc. Mobile Networks and Applications*, vol. 11, pp. 187–200, 2006.

[7] Q. Fang, J. Gao, L. Guibas, V. de Silva, and L. Zhang. GLIDER: Gradient landmark-based distributed routing for sensor networks. In *Proc. 24th Conference of the IEEE Communication Society (INFOCOM)*, pp. 339–350, 2005.

[8] S. P. Fekete, M. Kaufmann, A. Kröller, and N. Lehmann. A new approach for boundary recognition in geometric sensor networks. In *Proc. 17th Canadian Conference on Computational Geometry*, pp. 82–85, 2005.

[9] S. P. Fekete, A. Kröller, D. Pfisterer, S. Fischer, and C. Buschmann. Neighborhood-based topology recognition in sensor networks. In *Proc. ALGOSENSORS*, Springer LNCS vol. 3121, pp. 123–136, 2004.

[10] S. Funke. Topological hole detection in wireless sensor networks and its applications. In *Proc. Joint Workshop on Foundations of Mobile Computing*, pp. 44–53, 2005.

[11] S. Funke and C. Klein. Hole detection or: "How much geometry hides in connectivity?". In *Proc. 22nd ACM Sympos. Computational Geometry*, pp. 377–385, 2006.

[12] S. Ganeriwal, R. Kumar, and M. B. Srivastava. Timing-sync protocol for sensor networks. In *Proc. 1st Internat. Conf. on Embedded Networked Sensor Systems*, pp. 138–149, 2003.

[13] D. Ganesan, B. Krishnamachari, A. Woo, D. Culler, D. Estrin, and S. Wicker. Complex behavior at scale: An experimental study of low-power wireless sensor networks. UCLA/CSD-TR 02-0013, UCLA, 2002.

[14] R. Ghrist and A. Muhammad. Coverage and hole-detection in sensor networks via homology. In *Proc. 4th Internat. Sympos. Information Processing in Sensor Networks*, pp. 254–260, 2005.

[15] M. Held. Voronoi diagrams and offset curves of curvilinear polygons. *Computer Aided Design*, 30(4):287–300, 1998.

[16] J. Hershberger and S. Suri. An optimal algorithm for Euclidean shortest paths in the plane. *SIAM Journal on Computing*, 28(6):2215–2256, 1999.

[17] B. Karp and H. Kung. GPSR: Greedy perimeter stateless routing for wireless networks. In *Proc. ACM/IEEE Internat. Conf. on Mobile Computing and Networking*, pp. 243–254, 2000.

[18] Y.-J. Kim, R. Govindan, B. Karp, and S. Shenker. Geographic routing made practical. In *Proc. 2nd USENIX/ACM Sympos. Networked System Design and Implementation*, pp. 217–230, 2005.

[19] A. Kröller, S. P. Fekete, D. Pfisterer, and S. Fischer. Deterministic boundary recognition and topology extraction for large sensor networks. In *Proc. 17th ACM-SIAM Sympos. Discrete Algorithms*, pp. 1000–1009, 2006.

[20] J. S. B. Mitchell. A new algorithm for shortest paths among obstacles in the plane. *Ann. Math. Artif. Intell.*, 3:83–106, 1991.

[21] J. S. B. Mitchell. Geometric shortest paths and network optimization. In J.-R. Sack and J. Urrutia, eds., *Handbook of Computational Geometry*, pp. 633–701. Elsevier, 2000.

[22] E. M. Royer, P. M. Melliar-Smith, and L. E. Moser. An analysis of the optimum node density for ad hoc mobile networks. In *Proc. IEEE Internat. Conf. on Communications*, pp. 857–861, 2001.