Bounded Stretch Geographic Homotopic Routing in Sensor Networks

Kan Huang*, Chien-Chun Ni†, Rik Sarkar‡, Jie Gao† and Joseph S. B. Mitchell*

*Department of Applied Mathematics and Statistics, Stony Brook University. {khuang, jsbm}@ams.stonybrook.edu
†Department of Computer Science, Stony Brook University. {chni, jgao}@cs.stonybrook.edu
‡Department of Informatics, University of Edinburgh, UK. rsarkar@inf.ed.ac.uk

***Abstract*—Homotopic routing asks for a path going around holes according to a given "threading". Paths of different homotopy types can be used to improve load balancing and routing resilience. We propose the first lightweight homotopic routing scheme that generates constant bounded stretch compared to the shortest path of the same homotopy type. Our main insight is that in a sequence of triangles to traverse, a message always routed to the nearest point on the next triangle in the sequence travels at most a constant times the length of any shortest path going through the same sequence of triangles. Our routing scheme operates on two levels enabled by a coarse triangulation. The top level is used to specify and represent the requested homotopy type, while the bottom level executes the local greedy routing on a triangle sequence. After a preprocessing step that triangulates the given region and creates a minimum-size auxiliary structure, routing operates greedily at two different resolutions. We also present simulation analysis in a variety of settings and show that the paths indeed have small stretch in practice, considerably shorter than the bounds guaranteed by the theory.**

I. Introduction

In this paper we consider the problem of routing in a wireless sensor network deployed in a complex geometric domain Σ with holes. This is a common situation for large-scale sensor networks, as the shape of the sensor deployment region (due to obstacles, terrain variation and other deployment forbidding factors) necessarily comes to play with the network design and management.

Our goal is to find short paths of a specific *homotopy type*, i.e., paths that go around holes in some specific ordering. In the example of Figure 1, there are many different ways to "thread" a route from s to t in the network with three holes. Observe that paths α, β, γ are all different in a global sense, in that, e.g., one can't deform α to β without "lifting" it "over" some hole. In contrast, paths γ and δ are only different in a local sense; one can deform γ to δ continuously through local changes, keeping δ within the domain. This difference is characterized by the *homotopy type* of a path. Two paths in a domain are *homotopy equivalent* if one path can be continuously deformed to the other, while staying within the domain.

For paths in a network, differences in homotopy types are differences at a global scale, and are crucial factors in adapting to large dynamic obstructions – such as fires or floods that gradually destroy sensors in a region. The obvious method of using shortest paths globally, and making local detours to get around faults is not a good strategy in such cases. The phenomenon will continue to destroy local detours, causing loss of messages, forcing repeated detours and eventually blocking all local paths, requiring reconstruction of large parts of routing table. Knowing the topology of the network, and using homotopy types, we can effortlessly switch to a completely different type of path when we notice persistent disturbances in a region.

978-1-4799-3360-0/14/$31.00 © 2014 IEEE

In Figure 1, a regional failure connecting the upper two holes may destroy both γ and δ, while paths α and β remain available. Despite the importance of homotopic routing in terms of improving load balancing and routing resilience, it is only very recently that homotopic routing has been explicitly addressed [26]. In Zeng *et al.* [26], greedy routing in a virtual coordinate space finds paths of different homotopy types. However, the algorithm has no theoretical guarantee on the *path stretch*, i.e., there is no bound on the path length in comparison with the length of a shortest path *of the same homotopy type*.

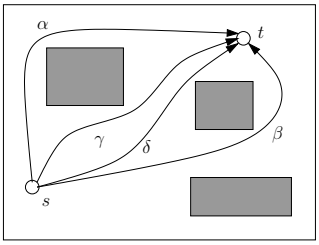


Fig. 1. In a network of 3 holes (shaded), paths α, β, γ have distinct homotopy types; γ and δ are homotopy equivalent.

Our contribution. In this paper we introduce a routing framework with a modest state per node that guarantees *constant worst-case stretch* for any given homotopy type. This is the first work that achieves a provable bound.

We assume that the sensor nodes are densely deployed in a geometric domain (e.g., campus map, floor plan) that is represented by a polygon (possibly with holes) Σ. While the network may have a huge number of nodes, the domain in which the nodes are deployed is often of a much smaller complexity. Let us denote by n the number of sensor nodes and k the number of vertices of Σ. In practice, $k \ll n$. Instead of building a structure on the network topology, we take the geometric domain in which the network is embedded and use a structure on Σ to encode the path homotopy type. Notice that this approximation gives us a number of benefits. The complexity of Σ is much smaller. Further, a structure operating on Σ is relatively stable, while network links can be volatile. As is common in geographical routing, we assume that nodes know their own geographical location through GPS or other localization schemes.

Our method operates on a two-level structure. On the top level, we use a coarse triangulation of the *geometric domain* to encode the network shape and to compute the path homotopy

type. On the bottom level, we show that a simple, greedy geographic routing scheme within a sequence of triangles, which together with the top-level gives a constant (bounded) worst-case stretch compared to the shortest path of the same homotopy type.

We decompose Σ into triangles using certain diagonals connecting vertices of the polygon Σ. The dual graph of the triangulation is a planar graph $\mathcal{D}$. If Σ has h holes, we cut the domain along h diagonals (*cut edges*) that interconnect the holes, thereby obtaining a simply connected domain whose corresponding dual graph is a tree T. For a particular homotopy type, the shortest path stays inside a sequence of triangles $S = \{\triangle_1, \triangle_2, \cdots, \triangle_m\}$ in the triangulation, whose dual is a simple path in T. Since the simple path connecting two nodes in a tree is unique, any existing method of routing on a tree, such as hyperbolic embedding [17] or compact routing labels [21], can be used to find S using a greedy algorithm on the tree T. Thus, the top-level greedy algorithm simply reveals the triangles of S, one at a time, which contains a shortest path of the required homotopy type. Our bottom-level algorithm will then realize a path within S, whose length is at most a constant times the shortest path inside S, i.e., the shortest path of the requested homotopy type. We remark that triangulation of Σ and the tree T can be computed at the network initialization phase. The corners of each triangle are pre-loaded at the sensor nodes that are inside the triangle, along with the corners of the (at most 3) triangles that are adjacent to it. This way, finding the "global path" in T is done at runtime in the network using local, greedy information.

The main technical contribution of the paper is the algorithm for the bottom level routing scheme, along with analysis showing it to have worst-case stretch bounded by a constant. It operates with only local information of the geometry of the current triangle Δ_i and the next diagonal to cross (the common edge of Δ_i and the next triangle, Δ_{i+1}, in the sequence) and yet finds globally short paths. Our algorithm is very simple: From the point where the path enters the current triangle, Δ_i, the path heads directly (greedily) to the closest point on the common edge of the next triangle, Δ_{i+1}, oblivious to the triangle sequence beyond Δ_{i+1}. It may appear counter-intuitive that this algorithm can have a bounded competitive ratio, since the strategy does not utilize the position of the destination t. The proof of the bounded stretch factor is highly non-trivial and is the major technical contribution of this paper.

To summarize, both the top-level and bottom-level routing algorithms are of a greedy nature. The top level computes a sequence of geometric triangles that the homotopic routing path should visit; the bottom level uses a greedy algorithm *in the sensor network* to find a network routing path realizing the requested homotopy type. The top level uses virtual coordinates for finding the homotopy type; the bottom level uses the nodes' true geographical coordinates to realize one such path. After the initial preprocessing, each node only stores the Euclidean coordinates of the corners of the triangle containing it and the adjacent triangles (for bottom-level routing), the virtual coordinates of its own and adjacent triangles (for top-level routing), and the h cut edges of the dual graph (for specifying the path homotopy types). The storage requirement for each node is of size $O(h)$, where h is the number of holes in Σ, and is independent of the network size or the complexity of the geometric domain Σ. The preprocessing involves computing a triangulation of Σ and the top-level embedding (e.g., by [17]) or computation of routing labels (e.g., by [21]), whose complexity is roughly linear in the complexity, k, of the geometric domain Σ and independent of the number, n, of sensor nodes, which can potentially be much larger.

As a by-product, our bottom-level greedy routing algorithm within a sequence of triangles can be extended to greedy routing within a sequence of consecutively adjacent simple polygons, with the same worst-case stretch, since we can always triangulate each simple polygon. Thus, our algorithm applies to a number of geometric routing schemes that first decompose a network into convex or nearly convex polygons [10, 25, 27], providing constant stretch for local routing.

Related Work. Our scheme is in the family of geometric routing schemes [11] that use the nodes' coordinates to guide a message to the destination. In particular, the simplest geographical greedy routing [15] delivers the message to the neighbor whose distance to the destination is the smallest. One well known limitation of this approach is that a message can get stuck at a node that does not have any neighbor closer to the destination; this often happens when the domain Σ is not convex. To resolve this issue, a number of schemes (such as GLIDER [10]) first decompose the network into pieces such that simple greedy routing can be carried out inside each piece and the adjacency of the pieces are extracted and propagated to the network on top of which global routing is performed. In some papers (e.g., [25, 27]), the sensor network domain is partitioned into convex or nearly convex pieces, and, again, a similar routing scheme is designed by first finding the sequence of pieces to visit and then using a local, greedy algorithm to deliver the message to the next piece in the sequence. However, none of these prior local routing schemes guarantees bounded stretch (compared to the shortest path through the same sequence of cells). Further, none of the global routing schemes explicitly considers path homotopies.

In the geometric setting, computing the shortest homotopic path inside a polygon Σ, when we have global knowledge of Σ, can be done in almost linear time. A simple polygon with k edges can be triangulated in $O(k)$ time [1, 8]. A polygon with k vertices and h holes can be triangulated in $O(k+h\log^{1+\epsilon} h)$ time [2], or in $O(k\log^* k + h\log k)$ expected time [24]. Given a triangulated polygon, computing the shortest path with a given homotopy type can be done in time linear in the number of times the path crosses a diagonal of the triangulation [13]. All of the above schemes assume full knowledge of the entire triangulation Σ and compute only a geometric path; they do not route within a sensor network deployed inside Σ.

There has been a number of related papers about online navigation for robot motion planning, which uses models sim-

ilar to our bottom-level routing algorithm. In one of the most investigated models [5], we are given a planar straight line graph H with n vertices, whose edges are weighted by their Euclidean lengths, the source s and destination t are vertices of H, and a packet can only move on edges of H. A packet only knows s, t, $N(v)$ (the set of neighbors of v), and the location of the packet. Various studies have been done under this model [4–7, 18]. It has been established that deterministic oblivious (i.e., "memoryless") algorithms can be found for triangulations, but no algorithm has constant competitive ratio. For triangulations that have the "diamond property", a constant competitive algorithm exists if the algorithm can use $O(1)$-memory within the packet being routed [6]. In particular, if the triangulation has exactly two ears (as does a sequence of triangles), there is a simple algorithm with competitive ratio 9. This setting is different from ours. In our setting, the triangulation is coarser than the network resolution. The routing path does not need to follow the edges of the triangulation; it can pass through interiors of triangles, potentially allowing the path to be significantly shorter.

The problem of robot motion planning and online navigation has also been considered in a geometric domain with obstacles under various models of the robots' vision; refer to the survey [20]. A *tactile* robot learns the boundary of an obstacle only when it encounters it and then moves along it [19]. A *vision-based* robot only learns the obstacles when it can see them. For tactile robots in an environment with square obstacles, constant competitive ratios can be achieved; when the obstacles can have unbounded aspect ratio, no constant competitive algorithm exists [9, 22]. If the domain is a special simple polygon called a "street", in which s, t split the polygon boundary into two chains such that any point on one chain is visible to some point on the other chain, then online algorithms exist in which a robot with a vision sensor can search for the destination t with a constant competitive ratio [16]. Additional results can be obtained for star-shaped polygons, etc.; refer to [3]. Note that a sequence of triangles, as in our setting, is a street. Note, though, that in our model the "vision" of the robot (message) is restricted to the current containing triangle.

II. Bounded Stretch Homotopic Routing

We assume a dense collection of sensor nodes deployed inside a given polygonal domain Σ with $h \geq 0$ holes. The number of vertices of Σ is k. The number of sensors inside Σ is n. Typically, $h \ll k \ll n$. Our objective is to prepare the sensor nodes with minimal information such that one can easily answer the query of *homotopic routing* using local greedy algorithms. We start by explaining how to define path homotopy in the query.

A. Path Homotopy

First we triangulate the polygon Σ, i.e., adding diagonals (edges connecting visible vertices of Σ) to decompose Σ into triangles. Note that no Steiner points are added. An example is shown in Figure 2.

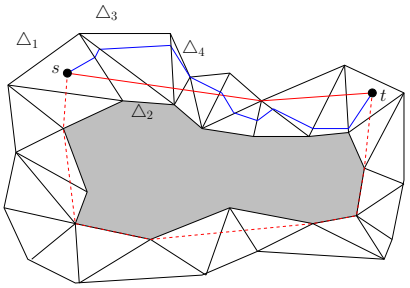


Fig. 2. A triangulated region and routing paths inside. The shaded part is a hole with no wireless nodes while the rest of the domain is densely covered by sensors (not shown in the figure). The red path shows the shortest path from point s to point t, while the blue path is created by following our greedy routing strategy. The dashed path is a shortest path with a different homotopy type.

Any path from a source s to a destination t must go through a sequence of triangles. We say a path follows a sequence S, if it visits the triangles in the order S. The homotopy type of a path is captured by S. Figure 2 shows (in solid red and dashed red) the shortest geometric paths from s to t of two different homotopy types.

To define a path of a certain homotopy type, we work with the dual graph ($\mathcal{D}$) of the triangulation, in which each triangle is represented by a vertex, and vertices corresponding to adjacent triangles are connected by edges. See Figure 3 for an example. A path in a domain with holes can go around holes in different ways; e.g., consider the red and green paths connecting z and d in Figure 3. To characterize these differences formally, we introduce *cut edges* to the triangulation, which are diagonals (or edges in the dual graph) that connect holes to the outer boundary (possibly via other holes). The cut edges are chosen one per hole arbitrarily . Removing the cut edges makes Σ a simple polygon and makes the dual graph $\mathcal{D}$ a tree, denoted by T. In Figure 3 the cut edges (shown dashed) are ab and cd in the dual; they cross corresponding triangulation edges (also shown dashed).

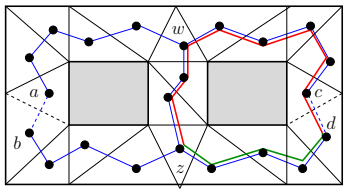


Fig. 3. A triangulated polygon. Dual graph shown with blue edges. Cut edges are shown in dashed lines. Red and green curves show paths of different homotopy types. The solid edges in the dual constitute the tree T.

This demarcation of cut edges suffices to distinguish paths of different homotopy types. Any two paths that cross the same sequence of cut edges are of the same homotopy type. Note that we are interested only in some *basic* homotopy types. In general, a path may go around a hole many times, but such paths are not of practical interest in network applications. Therefore we are only interested in the homotopy types in which the shortest path only visits each triangle once. This means that the path winds around a hole at most once, i.e., any cut appears at most once in the type description. Thus, the length of a description for a practical path is $O(h)$, where h is

the number of holes of Σ. The homotopy type definition is only with respect to the geometric domain Σ and is independent of the sensor network within Σ.

Our goal is, for a given homotopy type (sequence of cut edges), to find a path of that type *in the sensor network*, from the source to the destination, by using a local greedy decision rule. Our routing protocol follows a two-level structure, and adopts greedy decisions to make progress at each level:

1) **Top level:** Determines the sequence S of triangles that the shortest path with the specified homotopy type goes through.
2) **Bottom level:** Routes the message in the sensor network, following the sequence of triangles S.

The top-level procedure is concerned with finding the sequence of triangles that the shortest path of the given homotopy type should follow. Our hope of achieving short stretch paths rests on the bottom-level procedure, which must be designed with care and constitutes the major contribution of this paper. We will first describe our algorithm and prove its performance in the geometric case of a continuous Euclidean metric, and then consider its adaptation to a discrete network setting in Section II-D.

B. Greedy Routing at the Bottom Level

In this subsection we design and analyze the performance of the bottom-level protocol for greedy routing in a sequence of triangles $S = \triangle_1, \triangle_2, \cdots$.

Definition 2.1 (Bottom-Level Greedy Routing). *For a message at point p, inside $\triangle_i$ and destined for t,*

1) *It is routed along the shortest path to (any point in) the next triangle $\triangle_{i+1}$ in the sequence S.*
2) *If $\triangle_i$ is the last triangle in S, then it is routed along the shortest possible path from p to t.*

Such a path is shown in blue in Figure 2. While the algorithm is conceptually simple, the proof that it achieves short paths is quite involved. For lack of space, we will present only the important ideas and theorems here; the complete proofs of the theorems can be found in a full version online [14].

In the Euclidean plane, the *shortest path* between two points is the (straight) segment, while that from a point to triangle is simply the segment from the given point to the nearest point of the triangle. The greedy path from s to t therefore consists of such a sequence of segments through the triangles of S (shown in blue in Figure 2). We prove the following theorem.

Theorem 2.2. *Given a non-repeating sequence S of triangles, the greedy routing algorithm finds a path of length at most ρ times the length of the shortest path following the same sequence S, where ρ is a constant independent of the input.*

A non-repeating sequence means that no triangle appears more than once in S. Thus the shortest path never visits the same triangle again, and therefore does not self intersect. The theorem implies that while the algorithm operates greedily with very local information, it still produces good paths, not much longer than the shortest one in its homotopic category.

For points u and v on a path P, we use $P_{u,v}$ to denote the part of P in between them, and $|P_{u,v}|$ to denote its length. For general points x and y, $|xy|$ denotes the Euclidean distance between x and y. For vertices v_i and v_j on P, we denote the path between them by $P_{i,j}$, if there is no ambiguity.

Let us represent the greedy path by $P_{s,t}$ and the shortest path by $Q_{s,t}$. These two paths may intersect each other at points other than at s and t, but we need to only consider regions between intersections. To see why this is true suppose that the paths intersect at an intermediate point w. In this case, if $|P_{s,w}| \leq \rho|Q_{s,w}|$ and $|P_{w,t}| \leq \rho|Q_{w,t}|$, then by simply adding we have $|P_{s,t}| \leq \rho|Q_{s,t}|$. Thus we only need to prove the ρ stretch for each segment of P, Q between consecutive intersections.

If the line segment connecting s, t is inside the sequence of triangles, $Q_{s,t}$ is straight and its length equals $|st|$. Otherwise, $Q_{s,t}$ makes intermediate turns. We can divide the path $Q_{s,t}$ into subpaths each having all turns in the same direction. More formally, a *spiral* is a directed simple path with every turn in the same direction, clockwise or counterclockwise.

Lemma 2.3. *If v and v' are successive intersections of P and Q, then $Q_{v,v'}$: the shortest path between these two intersections, is a spiral.*

This lemma implies that to estimate the stretch between intersections, which is our goal, we can assume that the shortest path segment is a spiral.

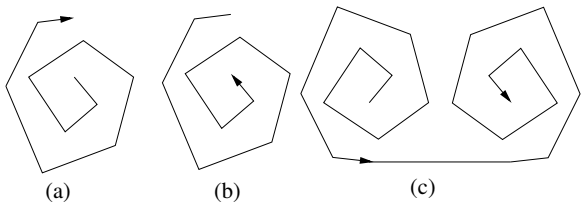


Fig. 4. Spiral paths. (a) Growing spiral; (b) Shrinking spiral; (c) Combined spiral, can be seen as a concatenation of growing and shrinking spirals

A spiral can be of different types; see Figure 4. We focus on the first case of a growing spiral shortest path segments; other cases are analogous.

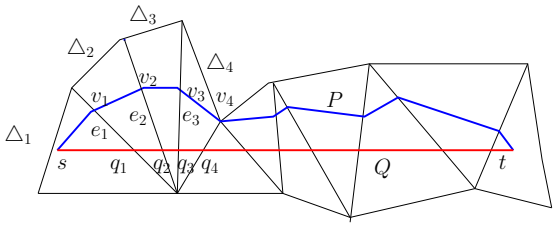


Fig. 5. The greedy path intersects the edges at points $v_1, v_2, \cdots$, while the shortest path intersects them at points $q_1, q_2, \cdots$ etc.

Any two successive triangles $\triangle_i$ and $\triangle_{i+1}$ in sequence S share a common edge, e_i; let l_i be the line containing e_i. Let v_i (resp., q_i) denote the intersection of P (resp., Q) with e_i, and let $r_i = q_i v_i$; see Figure 5.

Some segments of P (e.g., $v_1 v_2$) intersect the corresponding edge (here e_2) at an interior point, and at a right angle; we refer to these as *orthogonal segments*, or *o−segments*. Other

segments (e.g., v_3v_4) intersect at a boundary of the edge; we refer to these as *boundary segments*, or $b-$*segments.*

We will modify the greedy path P to a different path P', which is easier to analyze. This new path P' is necessarily longer than the original path, therefore an upper bound on its length is an upper bound on the length of P.

Modification: Replace b-segments. Let us start with $P' = P$ and modify through the following steps. For each b-segment $v_{i-1}v_i$, we take $v_{i-1}u_i$ as the perpendicular line from v_{i-1} to l_i. Then we replace $v_{i-1}v_i$ by an o-segment $(v_{i-1}u_i)$ and a segment u_iv_i. This replacement is shown in Figure 6. The segment u_iv_i is tangential to the edge e_i and we call it an l-*segment.*

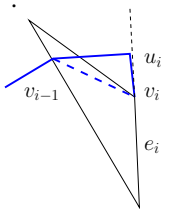


Fig. 6. We replace a b-segment such as $v_{i-1}v_i$ by an o-segment $(v_{i-1}u_i)$ and an l-segment u_iv_i.

After all such replacements, P' has no b-segments, and only has o and l segments. Applying triangle inequality at each place shows $|P'| \geq |P|$. For simplicity, let us refer to P' as P in the rest of the section. Note that v_i is now the point where the greedy path *leaves* the edge e_i.

The quantities most important to our analysis are shown in Figure 7. The angular turn made by the shortest path Q at point q_i is β_i. Equivalently, it is the *curvature* at q_i. The angle between the straight line $q_{i-1}q_i$ and l_{i-1} is α_i. The circular arc of radius r_{i-1} centered at q_{i-1} intersects straight line q_iq_{i-1} at g_{i-1}. It intersects the straight line through q_{i-1} that is parallel to q_iq_{i+1} at h_{i-1}. The arc of radius q_ig_{i-1} centered at q_i intersects l_i at w_i.

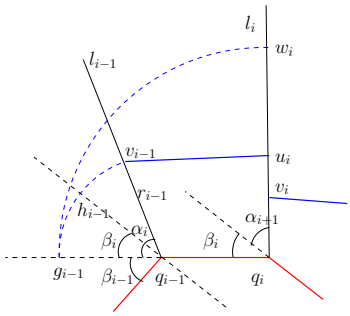


Fig. 7. Bounding the greedy path.

Theorem 2.4. *The total length, $|P| = \sum_{i=1}^{m} |P_{i-1,i}|$, of a growing spiral greedy path can be bounded by the sum of three terms: $|P| \leq A + B + C$, where:*

$$A = (2\pi + 1) \sum_{i=1}^{m} |Q_{i-1,i}|,$$

$$B = \sum_{i-1}^{m} |r_{i-1}|\beta_i,$$

$$C = |r_{m-1}|\alpha_{m+1} + \sum_{i=1}^{m-1} \alpha_{i+1}(|r_{i-1}| - |r_i|).$$

Note that A is simply $(2\pi + 1)|Q|$. Therefore, it remains to estimate B and C. It can be shown that between successive edges of S, the distance of P from Q along the edges e_i does not increase beyond the length of Q between these edges:

Lemma 2.5. $|r_i| - |r_{i-1}| \leq |Q_{i-1,i}|$, *for* $i = 2, \ldots m$.

From this result it follows that:

Lemma 2.6. $C \leq 2\pi|Q|$.

Finally, we need a bound for term $B = \sum_{i-1}^{m} |r_{i-1}|\beta_i$. According to Figure 7, $|r_{i-1}|\beta_i = |\ \overset{\frown}{g_{i-1}h_{i-1}}\ |$: the length of the arc $\overset{\frown}{g_{i_1}h_{i-1}}$. Intuitively, this is the cost the greedy path incurs at turns. The shortest path Q makes a sharp turn at q_{i-1}, while the greedy path P, which is at a distance of about r_{i-1}, has to travel a longer distance along the outside of the turn. We analyze this cost using the shortest path inside the wedge bounded by q_ig_i and q_ih_i.

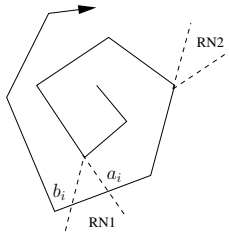


Fig. 8. Shortest path spiral and two types of wedges. RN1 wedges intersect some part of the shortest path while RN2 wedges do not.

Figure 8 shows two types of possibilities for such wedges. At the outer points of the spiral, the wedges do not intersect the shortest path Q; we say they are of type $RN2$. At the inner points of a spiral, the resulting wedges intersect one or more outer layers of Q; we call these of type $RN1$, and denote by a_i and b_i the nearest intersections of Q with the two rays. If $\xi_i = Q_{a_ib_i}$ is part of the shortest path between these points, then it can be shown that over the $RN1$ type wedges: $\sum_{RN1} |r_i|\beta_{i+1} \leq \sum_{RN1} \frac{\pi}{2}\xi_i$. The right side can be shown to be no more than $(\pi/2)|Q|$. While for $RN2$ type wedges, it is easy to show that the total angle is bounded by $\sum_{RN2} \beta_i \leq 3\pi$. Therefore, we get the resultant property that:

Lemma 2.7. *For wedges of types $RN1$ and $RN2$,*

$$\sum_{RN1} |r_i|\beta_{i+1} \leq \tfrac{\pi}{2}|Q|,$$

$$\sum_{RN2} |r_i|\beta_{i+1} \leq 3\pi|Q|.$$

Therefore, it follows that $B \leq \left(\frac{\pi}{2} + 3\pi\right)|Q|$. Using similar arguments (omitted from this abstract), if Q is a growing (or similarly, shrinking) spiral, $|P| \leq A + B + C \leq \left(\frac{15}{2}\pi + 1\right)|Q|$, while the length for a combined spiral is bounded by $|P| \leq 2(A + B + C)$. From the arguments that a factor that holds as upper bound between intersections holds for the entire path, we get the final theorem:

Theorem 2.8. *The length of the greedy path is at most* $(15\pi + 2)$ *times the length of the shortest path.*

This is Theorem 2.2 with the constant $\rho = 15\pi + 2$; this is a worst-case bound and likely can be improved. Also, our simulations show that in practice the stretch factor is much less than the theoretical bound. In the special case when Q is a straight line, we obtain a much smaller bound of $\pi + 1$; in fact, for this case, this bound is tight (see [14]).

Theorem 2.9. *If the shortest path Q is a straight line, the length of the greedy path is at most $(\pi + 1)$ times the length of the shortest path.*

We had assumed that no triangles appear twice in the triangle sequence S, which is the only practically interesting case. However, Theorem 2.2 will still hold, with possibly a larger constant, as long as a triangle is not repeated more than a constant number of times.

C. Greedy Routing at the Top Level

At the top level, we have a tree T, dual to the triangulation of Σ after cut edges are removed. Routing at this level deals with finding a sequence of triangles that contains the shortest path of the specified homotopy type. Routing in a tree using a greedy algorithm can be done in various ways. We describe two such methods below and show how to add homotopy types on top of it.

Homotopic routing by hyperbolic embedding. To support greedy routing in the tree T, we can embed T in the hyperbolic plane using the method of Kleinberg [17]. Here we explain how to augment it for routing of a given homotopy type.

To support routing of a given homotopy type, we need to attach copies of the tree along cut edges such that routes through cut edges can be found in a greedy manner. Each cut edge connects two vertices of T that are always leaves. We can attach a copy of T to the open end of each duplicate cut edge. The continuity that was lost in removing the cut edges to create T is now restored.

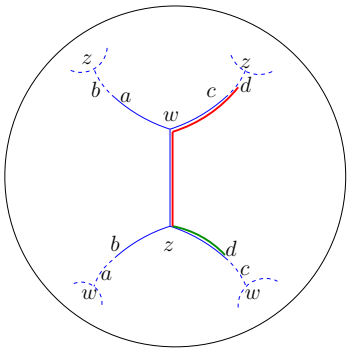


Fig. 9. The tree T is shown in solid blue inside the Poincare disk. The red and green paths from Figure 3 are shown as red and green paths to different images of d. Together they represent a loop.

For example, the edge ab in Figure 3 maps to two different leaves b and a. Let us attach a copy of T by connecting edge ab at a and b, respectively, to maintain continuity of the original $\mathcal{D}$. That is, a and b are now neighbors once again, while maintaining all other neighbor relations. However, this creates a new set of cut leaves with discontinuity where we need to attach copies of T for continuity. This can be carried out indefinitely to get a tree $\mathcal{T}$ with infinitely number of copies of T. This tree $\mathcal{T}$ is the *Universal Covering Space* of the graph $\mathcal{D}$. Each node of T maps to many copies in the universal covering space $\mathcal{T}$. Paths of different types are obtained by simply routing to different images of the destination in $\mathcal{T}$. In particular, the shortest path for a given homotopy type maps to a simple path in the $\mathcal{T}$ connecting the source to the proper image of the destination. This simple path can be found by using a greedy routing algorithm using a embedding of $\mathcal{T}$ in hyperbolic virtual coordinates, as explained below.

The universal covering space of any graph is a tree [12], and an infinite tree of bounded degree has a greedy, $(1 + \varepsilon)$-distortion embedding in the hyperbolic plane that can be computed by a distributed algorithm [17, 23]. Figure 9 shows an example of the embedding in the Poincare disk.

To perform the top-level routing, we need to identify this suitable image of the destination corresponding to the given homotopy type. Recall that a homotopy type is determined by the sequence in which a path crosses cut edges. For example, the red path above crosses the cut edge cd in direction $\overrightarrow{cd}$. We define a corresponding transformation in the hyperbolic plane: the transformation g_{cd} maps the cd edge at the bottom right to the cd edge at the top right. Then the two destinations d and $g_{cd}(d)$ corresponds to different homotopy types shown by the green and red paths.

A path to $g_{cd}(d)$ must cross the cut edge cd in direction $\overrightarrow{cd}$, and thus is equivalent to that homotopy type. Transformations such as g_{cd} form a group called the group of *deck transformations*. The generators of this group can be composed together: $g_{ab}g_{cd}(w) = g_{ab}(g_{cd}(w))$. Therefore, a homotopy type can be represented by a sequence of generators, $g = g_1 g_2 g_3 \ldots$, which represents crossing the corresponding cut edges in the given order and directions. Thus, for our purposes, given a homotopy type g and a final destination d, we will select as our destination $g(d)$ and route to it by greedy routing in the hyperbolic metric. Each generator is an isometry of the hyperbolic plane and can be written as a Möbius transformation: $z \to e^{i\theta}\frac{z - z_0}{1 - \bar{z}_0 z}$, with only two parameters: a complex number z_0 and an angle θ. A composition g can be written in the same form, and therefore needs only two parameters as well.

An analogous method of universal covering spaces used in [26] embeds the *entire* network in the hyperbolic plane using an expansive Ricci Flow computation. In contrast, we need to embed only a coarse tree.

Routing with compact labels of T. An alternative method of finding routing paths in $\mathcal{D}$ is to use compact routing labels for the finite tree T. In a tree of size m – for example, as described in [21] – this method assigns label $\zeta(p)$ of size $O(\log m)$ to each node p. Given a destination label t, the source s can find a neighbor p that is closer to t by merely comparing $\zeta(p)$ and $\zeta(t)$. Using ζ, we can perform homotopic routing as follows. Suppose $ab, cd, \ldots$ is the sequence of cut edges for the type. We route from s to a using ζ, route from a to b locally, and route from b to c using ζ again, and so on.

D. Implementation in a Sensor Network

In this section we describe the implementation issues in a sensor network. This includes preprocessing to prepare the necessary information for both top-level and bottom-level routing algorithms. We also elaborate on how to implement the bottom-level routing in a network setting.

Preprocessing. We assume that the polygon Σ (e.g., a map of the deployment domain) in which the network is deployed is already known before the deployment/initialization of the

network. The triangulation of Σ is done using a centralized algorithm offline by the network owner/operator. A polygon with k vertices and h holes can be triangulated in $O(k+h\log^{1+\epsilon} h)$ time [2], or in $O(k\log^* k + h\log k)$ expected time [24]. h cut edges are selected to cut the holes open. We also prepare the coordinates for top-level routing in the tree T (and the universal covering space $\mathcal{T}$) obtained from the triangulation after the removal of the cut edges in a centralized manner. The running time is dependent on k, h, the complexity of Σ instead of the size of the sensor network.

Information about the triangles as well as the coordinates of nodes of $\mathcal{T}$ are then disseminated to the network using a network-wide broadcast such that each sensor node only stores the following information: 1) the Euclidean coordinates of the triangle containing itself and the (at most 3) adjacent triangles; 2) the virtual coordinates of the triangle for top-level routing and that of (at most 3) adjacent triangles; and 3) the cut edges for issuing homotopic routing. With such information each node is able to perform top-level greedy routing given a specified homotopy type using only local information. The storage requirement for each node is $O(h)$. The message size in the broadcast procedure is $O(k)$, the size of Σ. Notice that this procedure is only done once for the entire network lifetime.

To specify a destination, we need to specify the triangle containing it and its Euclidean coordinates. The former is for top-level routing and the later is for bottom-level routing when the message gets to the final triangle. Such information for the destination can be obtained through standard location services.

Bottom-level routing in a sensor network. To perform bottom-level routing, we implement the greedy strategy, described in the continuous setting in Section II-B. There are two cases. In the first case, the destination is not in the current triangle $\triangle$. The top-level routing suggests the next triangle $\triangle'$ to be visited. We find a path in the network to the geometric diagonal shared by $\triangle$ and $\triangle'$. We simply use geographical greedy routing towards the diagonal, i.e., forwarding the message to the neighbor whose Euclidean distance is closest to the next triangle. Since we assumed that sensors are densely deployed, geographical greedy routing inside a triangle-shape domain should have high delivery rate. In the rare situation that the message gets stuck, for example due to small and temporary routing holes, we simply flood it inside the triangle $\triangle$. In the second case, the destination is inside the current triangle $\triangle$; again, simple geographical greedy routing is employed and, in rare situations, flooding is used when a message gets stuck.

III. Simulations

We have implemented our algorithm and run simulations to investigate the observed stretch factor and load balancing. Here, we are interested in evaluating the quality of the bottom level of our algorithm; the top level simply produces the unique sequence of triangles to be used at the bottom level. We compare to three algorithms for routing in a network within a sequence of geometric triangles: shortest path (SP) routing, the landmark-based greedy routing algorithm used in GLIDER [10], and the greedy algorithm using hyperbolic virtual coordinates in [26].

GLIDER: A landmark-based 2-level routing method. GLIDER [10] starts by selecting some nodes as *landmarks*. Then, these landmarks flood the network such that every node in the network can identify its nearest landmark. Nodes with the same nearest landmark are said to be in the same Voronoi cell. Each node is informed of the dual graph of the Voronoi decomposition and thus knows the path in the dual to any cell. When a message needs to move from cell A to cell B to cell C, the message moves along a shortest path toward the landmark of B, until it enters B. At that point, it switches to a shortest path toward the landmark of C, etc.

Hyperbolic routing using Ricci flow. The hyperbolic routing algorithm in [26] uses hyperbolic Ricci flow to compute an embedding of the network in the hyperbolic space. Similarly, the universal covering space is embedded so that homotopic routing is supported. We remark that our method uses a very coarse triangulation on the top level, rather than embedding the entire network. In addition, our method guarantees bounded stretch, while hyperbolic routing has no worst case guarantee.

Setting. While both the GLIDER scheme and our algorithm utilize network decompositions, the decompositions are different in nature. Thus, for consistent comparison, we slightly modify the GLIDER scheme to utilize the partition of the given triangulation (instead of the Voronoi partition), treating the triangles as cells, and setting the landmark of a triangle to be the node nearest to the centroid of that triangle. Further, all paths in comparison have the same homotopy type.

The simulation results reported in here are for unit-disk graphs. The results for quasi-unit-disk graphs and other relaxations of radio models are similar (and thus omitted from this extended abstract). The nodes are distributed as a perturbed grid.

Our main observations from the simulations are:

- Our greedy routing method almost always finds routes with a stretch of at most 3, which is much better than the theoretical bounds, GLIDER and hyperbolic routing. Its performance is consistent and is not sensitive to the actual triangulation of Σ used.
- Our algorithm distributes traffic load more evenly than the GLIDER 2-level routing method and causes fewer hotspots.
- Although there is no theoretical guarantee on the stretch for the GLIDER algorithm and the hyperbolic routing scheme, the paths they generate have stretch typically no more than 5 in all of the examples we have tested. This is in agreement with results reported in [10] and [26].

Path Length. We first consider a network of 8713 sensors deployed in a rectangle with three concave holes. The average degree is 23. We first triangulate this region as shown in Figure 10. We then take random sequences of triangles from this network, select random source and destination from them, execute routing using different methods and measure the

stretch and node loads. Each experiment is repeated $10,000$ times.

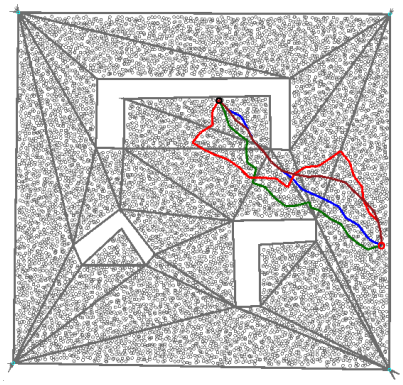


Fig. 10. The rectangular domain with 8713 nodes distributed on a perturbed grid. Comparing a shortest path (blue), the GLIDER path (red), the hyperbolic routing path (brown), and our greedy route (green). The starting point is circled by red, the end point is circled by black.

Figure 11 shows the distribution of the stretch of our algorithm, GLIDER, and hyperbolic routing over the shortest path algorithm. Of the three algorithms, 90% of the paths have stretch factor below 1.5, which is far better than the theoretical bound. Moreover, in this experiment our algorithm produces no stretch greater than 3, which outperforms the other two algorithms.

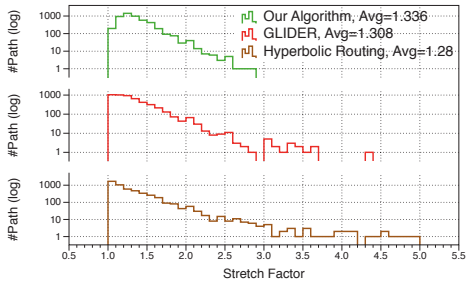


Fig. 11. Histogram of stretch factors of paths produced by our algorithm, GLIDER, and hyperbolic routing. Our algorithm has a typical stretch less than 1.5, and is always less than 3, while GLIDER and hyperbolic routing have some paths of length more than 3 times the shortest path length.

Load Balancing. Geographical routing using face routing to get around holes is known to have a tendency to congregate near hole boundaries, creating traffic hotspots that slow down routing and drain sensor batteries. It has been shown that both GLIDER and hyperbolic routing improved load balancing in this aspect [10, 26], since both of them take long de-tours around holes.

We show the load distribution as a histogram in Figure 12 for different algorithms. All three algorithms perform reasonably well in this respect. Our algorithm is slightly better than GLIDER and slightly worse than hyperbolic routing.

Various Domains. We tested the performance of our algorithm in the six different domains shown in Figure 13. In Figure 13(a) we tested the fan shape, in which it is expected that our algorithm performs close to the worst case stretch bound – our route approximately follows a semi-circle, resulting in a stretch factor of about $\pi/2$. We also tested various domains with obstacles.

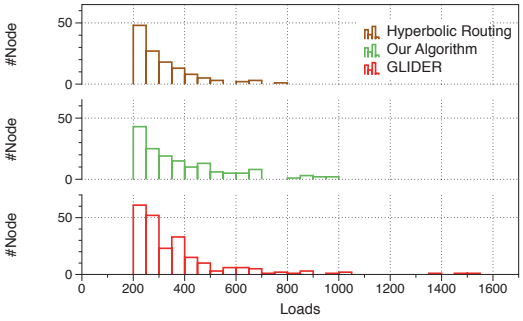


Fig. 12. A load comparison between paths produced by hyperbolic routing, our algorithm and GLIDER. Only nodes with load above 200 are shown here.

We chose $10,000$ random routing paths within each network. Table I shows that the average stretch of the greedy routing method is always less than 1.5, irrespective of the network geometry.

Networks	Stretch
Fan shape domain	1.412191984
Spiral shape domain	1.271866423
Polygon domain with 1 hole	1.338248974
Rectangular domain with 1 hole	1.328246153
Rectangular domain with 5 holes	1.313282006
Rectangular domain with 7 holes	1.319227400

TABLE I. Stretch in different networks of Figure 13

Dependency on Triangulations. To check the dependence of the quality of results on the nature of triangulations, we repeated the experiment for different triangulations with randomly distributed interior vertices, and found that the standard deviation of the stretch of our method and GLIDER are both small, about 0.013 and 0.018, respectively, implying that the performances of these methods are largely independent of the triangulation.

IV. Conclusion

We presented an algorithm to perform homotopic routing in large scale sensor networks. The greedy routing method is extendable to general decompositions of networks beyond triangles, and can be used to improve routing performance in other decomposition-based routing methods. Future work will utilize short homotopic paths for generating resilient backup paths, specific types of loops for surveillance, etc.

Acknowledgements. We thank Michael Biro for helpful discussions. J. Mitchell is partially supported by NSF (CCF-1018388). Jie Gao and Chien-Chun Ni acknowledge supports by NSF through grants DMS-1221339, CNS-1217823 and CNS-1116881.

References

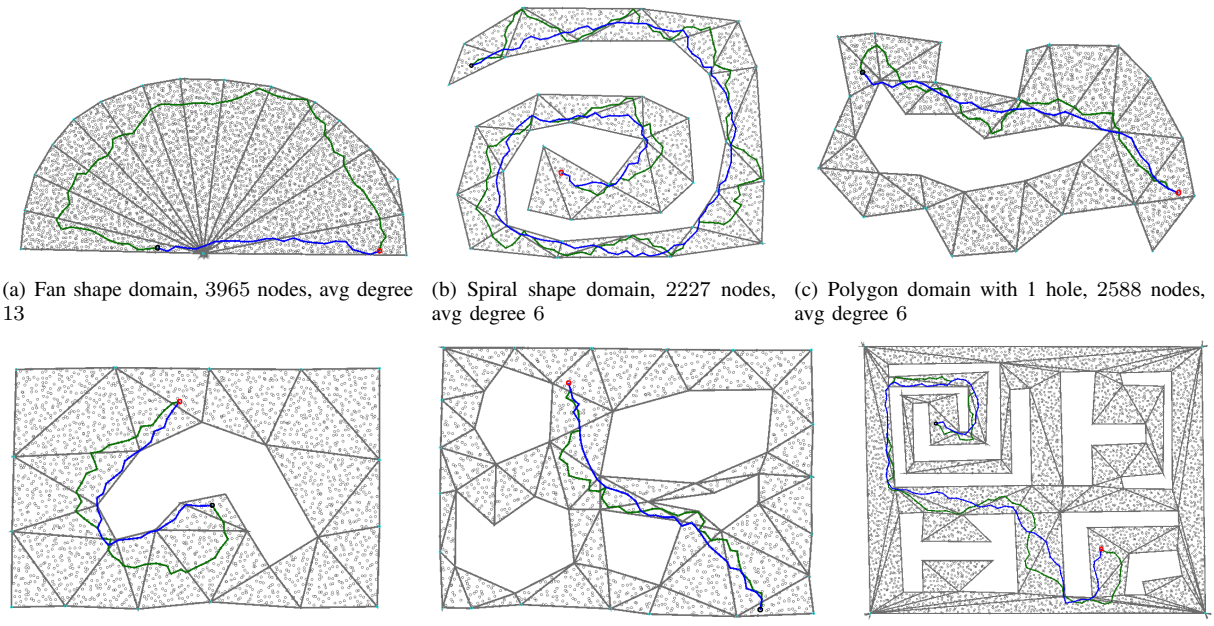
[1] N. M. Amato, M. T. Goodrich, and E. A. Ramos. A randomized algorithm for triangulating a simple polygon in linear time. *Discrete Comput. Geom.*, pages 245–265, 2001.

[2] R. Bar-Yehuda and B. Chazelle. Triangulating disjoint Jordan chains. *Internat. J. Comput. Geom. Appl.*, 4(4):475–481, 1994.

[3] P. Berman. On-line searching and navigation. In *Online Algorithms*, pages 232–241, 1996.

[4] P. Bose, A. Brodnik, S. Carlsson, E. D. Demaine, R. Fleischer, A. López-Ortiz, P. Morin, and J. I. Munro. Online routing in convex subdivisions. *Int. J. Comput. Geometry Appl.*, 12(4):283–296, 2002.

[5] P. Bose and P. Morin. Online routing in triangulations. In *Proceedings of the 10th International Symposium on Algorithms and Computation (ISAAC '99)*, pages 113–122, 1999.



(a) Fan shape domain, 3965 nodes, avg degree 13

(b) Spiral shape domain, 2227 nodes, avg degree 6

(c) Polygon domain with 1 hole, 2588 nodes, avg degree 6

(d) Rectangular domain with 1 hole, 2403 nodes, avg degree 6

(e) Rectangular domain with 5 holes, 1883 nodes, avg degree 6

(f) Rectangular domain with 7 holes, 5432 nodes, avg degree 7

Fig. 13. Six different domains and the shortest path (blue) and the path (green) obtained using our algorithm, the starting point is circled by red, the end point is circled by black.

[6] P. Bose and P. Morin. Competitive online routing in geometric graphs. *Theor. Comput. Sci.*, 324(2-3):273–288, 2004.

[7] P. Bose and P. Morin. Online routing in triangulations. *SIAM J. Comput.*, 33(4):937–951, 2004.

[8] B. Chazelle. Triangulating a simple polygon in linear time. *Discrete Comput. Geom.*, 6(5):485–524, Aug. 1991.

[9] P. Eades, X. Lin, and N. C. Wormald. Performance guarantees for motion planning with temporal uncertainty. *Australian Computer Journal*, 25(1):21–28, 1993.

[10] Q. Fang, J. Gao, L. Guibas, V. de Silva, and L. Zhang. GLIDER: Gradient landmark-based distributed routing for sensor networks. In *Proc. of the 24th Conference of the IEEE Communication Society (INFOCOM)*, volume 1, pages 339–350, March 2005.

[11] J. Gao and L. Guibas. Geometric algorithms for sensor networks. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 370(1958):27–51, 2012.

[12] A. Hatcher. *Algebraic Topology*. Cambridge University Press, November 2001.

[13] J. Hershberger and J. Snoeyink. Computing minimum length paths of a given homotopy class. *Comput. Geom. Theory Appl.*, 4(2):63–97, June 1994.

[14] K. Huang, C.-C. NI, R. Sarkar, J. Gao, and J. S. B. Mitchell. Bounded stretch homotopic routing using hyperbolic embedding of sensor networks. http://homepages.inf.ed.ac.uk/rsarkar/papers/homotopic-full.pdf.

[15] B. Karp and H. Kung. GPSR: Greedy perimeter stateless routing for wireless networks. In *Proc. of the ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom)*, pages 243–254, 2000.

[16] R. Klein. Walking an unknown street with bounded detour. In *Proceedings of the 32nd annual symposium on Foundations of computer science*, SFCS '91, pages 304–313, Washington, DC, USA, 1991. IEEE Computer Society.

[17] R. Kleinberg. Geographic routing using hyperbolic space. In *Proceedings of the 26th Conference of the IEEE Communications Society (INFOCOM'07)*, pages 1902–1909, 2007.

[18] E. Kranakis, H. Singh, and J. Urrutia. Compass routing on geometric networks. In *Proc. 11th Canadian Conference on Computational Geometry*, pages 51–54, 1999.

[19] V. J. Lumelsky and A. A. Stepanov. Path-planning strategies for a point mobile automaton moving amidst unknown obstacles of arbitrary shape. *Algorithmica*, 2:403–430, 1987.

[20] J. S. Mitchell. Geometric shortest paths and network optimization. In *Handbook of Computational Geometry*, pages 633–701. Elsevier Science Publishers B.V. North-Holland, 1998.

[21] J. Newsome and D. Song. GEM: graph embedding for routing and data-centric storage in sensor networks without geographic information. In *SenSys '03: Proceedings of the 1st international conference on Embedded networked sensor systems*, pages 76–88, 2003.

[22] C. H. Papadimitriou and M. Yannakakis. Shortest paths without a map. *Theor. Comput. Sci.*, 84(1):127–150, July 1991.

[23] R. Sarkar. Low distortion delaunay embedding of trees in hyperbolic plane. In *Proceedings of the 19th international conference on Graph Drawing*, GD'11, pages 355–366, 2011.

[24] R. Seidel. A simple and fast incremental randomized algorithm for computing trapezoidal decompositions and for triangulating polygons. *Comput. Geom. Theory Appl.*, 1(1):51–64, July 1991.

[25] G. Tan, M. Bertier, and A.-M. Kermarrec. Convex partition of sensor networks and its use in virtual coordinate geographic routing. In *INFOCOM*, pages 1746–1754, 2009.

[26] W. Zeng, R. Sarkar, F. Luo, X. D. Gu, and J. Gao. Resilient routing for sensor networks using hyperbolic embedding of universal covering space. In *Proc. of the 29th Annual IEEE Conference on Computer Communications (INFOCOM'10)*, pages 1694–1702, March 2010.

[27] X. Zhu, R. Sarkar, and J. Gao. Segmenting a sensor field: Algorithms and applications in network design. *ACM Trans. Sen. Netw.*, 5(2):12:1–12:32, Apr. 2009.