Asymptotically Optimal Geometric Mobile Ad-Hoc Routing

Asymptotically Optimal Geometric Mobile Ad-Hoc Routing

Fabian Kuhn, Roger Wattenhofer, Aaron Zollinger
{kuhn,wattenhofer,zollinger}@inf.ethz.ch
Department of Computer Science, ETH Zurich, 8092 Zurich, Switzerland

Abstract

In this paper we present AFR, a new geometric mobile ad-hoc routing algorithm. The algorithm is completely distributed; nodes only need to communicate with direct neighbors in their transmission range. We show that if a best route has cost c, AFR finds a route and terminates with cost $O(c^2)$ in the worst case. AFR is the first algorithm with cost bounded by a function of the optimal route. We also give a tight lower bound by showing that *any* geometric routing algorithm has worst-case cost $\Omega(c^2)$. Thus AFR is asymptotically optimal. We give a non-geometric algorithm that also matches the lower bound, but needs some memory at each node. This establishes an intriguing trade-off between geometry and memory.

1 Introduction

A mobile ad-hoc network consists of mobile nodes equipped with a wireless radio. We think of mobile nodes as points in the Euclidean plane. Two nodes can directly communicate with each other if and only if they are within transmission range of each other. Throughout this paper we assume that all nodes have the same transmission range R[1]. Two nodes with distance greater than R can communicate by relaying their messages through a series of intermediate nodes; this process is called multi-hop routing.

In this paper we study so-called geometric routing; in networks that support geometric routing a) each node is equipped with a location service, i.e. each node knows its Euclidean coordinates, b) each node knows all the neighbor nodes (nodes within transmission range R) and their coordinates, and c) the sender of a message knows the coordinates of the destination.

In addition to the standard assumptions a), b) and c), we take for granted that mobile nodes are not arbitrarily close to each other, i.e. d) there is a positive constant d_0 such that the distance between any pair of nodes is at least d_0. This is motivated by the fact that there are physical limitations on how close together two mobile nodes can be placed. Further, distances between neighboring nodes in an ad-hoc network will typically be in the order of the transmission range.

In this paper we present a new geometric routing algorithm which borrows from the eminent *Face Routing* algorithm by Kranakis, Singh, and Urrutia [KSU99]. As it is the tradition in the community, we give our algorithm a name: AFR which stands for *Adaptive Face Routing*[2]. Our algorithm is completely local; nodes only exchange messages with their direct neighbors, i.e. nodes in their transmission range R. We show that if a best route has cost c, our algorithm finds a route and terminates with cost $O(c^2)$ in the worst case. This bound holds for many prominent cost models such as distance, energy, or the link metric. Note that the distance of the best route (the sum of the distances of the single hops) can be arbitrarily larger than the Euclidean distance of source and destination. Our algorithm is the first algorithm that is bounded by a function of the optimal route; the original *Face Routing* algorithm and all other geometric routing algorithms are only bounded by a function of the number of nodes.

Moreover we show that *any* geometric routing algo-

[1] In the technical part of the paper we simplify the presentation by scaling the coordinates of the system such that $R = 1$.

[2] Is it a coincidence that AFR also reflects our first names?

rithm has cost $\Omega(c^2)$. This tight lower bound proves that our algorithm is asymptotically optimal[3]. The lower bound also holds for randomized algorithms. Apart from the theoretical relevance of our results, we feel that our algorithm has practical potential, especially as a fall-back mechanism for greedy geometric routing algorithms (which are efficient in an average case).

It is surprising that the cost of geometric routing algorithms is quadratic in the cost of the best route. We show that this bound can also be achieved by a simple non-geometric routing algorithm. In exchange for the missing location service we give the algorithm some extra memory at each node. We show that this algorithm also has cost $O(c^2)$, which, contrary to intuition, proves that in the worst case a GPS is about as useful as some extra bits of memory.

The paper is organized as follows. In the next section we discuss the related work. In Section 3 we formally model mobile ad-hoc networks and geometric routing algorithms. In Section 4 we present and analyze our geometric routing algorithm AFR. We give a matching lower bound in Section 5. Section 6 concludes and discusses the paper.

2 Related Work

Traditionally, multi-hop routing for mobile ad-hoc networks can be classified into proactive and reactive algorithms. Proactive routing algorithms copycat the behavior of wireline routing algorithms: Each node in the mobile ad-hoc network maintains a routing table that lays down how to forward a message. Mobile nodes locally change the topology of the network, which in turn provokes updates to the routing tables throughout the network. Proactive routing algorithms are efficient only if the ratio of mobility over communication is low. If the nodes in the network are reasonably mobile, the overhead of control messages to update the routing tables becomes unacceptably high. Also storing large routing tables at cheap mobile nodes might be prohibitively expensive. Reactive routing algorithms on the other hand find routes on demand only. The advantage is that there is no fixed cost for bureaucracy. However, whenever a node needs to send a message to another node, the sender needs to flood the network in order to find the receiver and a route to it. Although there are a myriad of (often obvious and sometimes helpful) optimization tricks, the flooding process can still use up a significant amount of scarce wireless bandwidth. Reviews of routing algorithms in mobile ad hoc networks in general can be found in [BMJ+98] and [RT99].

Over a decade ago researchers started to advocate equipping every node with a location information system [TK84, HL86, Fin87]; Each node knows its geometric coordinates [HB01]. If the (approximate) coordinates of the destination are known too, a message can simply be sent/forwarded to the "best" direction. This approach is called directional, geometric, geographic, location-, or position-based routing. With the growing availability of global positioning systems (GPS or Galileo), it can easily be imagined to have a corresponding receiver at each node [IN96]. Even if this is not the case, one can conceive that nodes calculate their position with a local scheme; a research area that has recently been well studied [SHS01]. Geometric routing only works if nodes know the location of the destination. Clearly, the (approximate) location of the destination changes much less frequently than the structure of the underlying graph. In this sense it is certainly less expensive to keep the approximate locations of the destinations than of the whole graph. In the area of peer-to-peer networking a lot of data structures have been presented that store this type of information in an efficient way. It would be possible to use an overlay peer-to-peer network to maintain the position of all destinations [LJD+00]. Last but not least one could imagine that we want to send a message to any node in a given area, a routing concept that is known as geocasting [NI97, KV98]. Overviews of geometric routing algorithms are given in [RS96, GSB01, MWH01].

The first geometric routing algorithms were purely greedy: The message is always forwarded to the node that is closest to the destination [TK84, HL86, Fin87]. It was shown that even simple location configurations do not guarantee that a message will reach its destination when forwarded greedily. For exam-

[3]The constant between the lower and the upper bound depends on the cost model, but can generally become quite large.

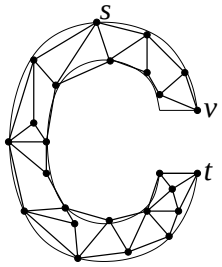

Figure 1: Greedy routing fails with nodes distributed on the letter "C".

ple, we are given a network with nodes that are distributed "on" the letter "C" (see figure 1). Assume that the northernmost node s of "C" wants to send a message to destination t (the southeastern tip of "C"). With greedy routing the message is forwarded from the source to the best neighbor, i.e. in the southeastern direction. At node v (the north western tip of "C") there is no neighbor node closer to the destination, and the routing algorithm fails. To circumvent the gap of the "C", the source should have sent the message to the west. It has been shown that many other definitions of "best" neighbor (e.g. best angle a.k.a. *Compass Routing* in [KSU99]) do not guarantee delivery either.

The first geometric routing algorithm that guarantees delivery is the *Face Routing* algorithm, proposed in a seminal paper by Kranakis, Singh, and Urrutia [KSU99] (in their short paper they call the algorithm *Compass Routing II*). The *Face Routing* algorithm is a building block of our routing algorithm AFR and will therefore be discussed in more detail later. The *Face Routing* algorithm guarantees that the message will arrive at the destination and terminates in $O(n)$ steps, where n is the number of nodes in the network. This is not satisfactory, since already a very simple flooding algorithm will terminate in $O(n)$ steps. In case the source and the destination are close, we would like to have an algorithm that terminates earlier. In particular, we are interested in the competitive ratio of the route found by the algorithm over the best possible route.

There have been other suggestions for geometric routing algorithms with guaranteed delivery [BMSU99, DSW01], but in the worst case (to the best of our knowledge) none of them is better than the original *Face Routing* algorithm. Other (partly non-deterministic) greedy routing algorithms have been shown to find the destination on special planar graphs, such as triangulations or convex subdivisions [BMB+00], without any performance guarantees.

It has been shown that the shortest path between two nodes on a Delaunay triangulation is only a small constant factor longer than their distance [DFS90]. It has even been shown that indeed there is a competitive routing algorithm for Delaunay triangulations [BM99]. However, nodes can only communicate within transmission range R: Delaunay triangulation is not applicable since edges can be arbitrarily long in Delaunay triangulations. Accordingly, there have been attempts to approximate the Delaunay triangulation locally [LCW02] but no better bound on the performance of routing algorithms can be given for such a construction.

A more detailed discussion of geometric routing can be found in [Urr02].

3 Model

This section introduces the notation and the model we use throughout the paper. We consider routing algorithms on Euclidean graphs, i.e. weighted graphs where edge weights represent Euclidean distances between the adjacent nodes in a particular embedding in the plane. As usual, a graph G is defined as a pair $G := (V, E)$ where V denotes the set of nodes and $E \subseteq V^2$ denotes the set of edges. The number of nodes is called $n := |V|$ and the Euclidean length of an edge $e \in E$ is denoted by $c_d(e)$. A path $p := v_1, \ldots, v_k$ for $v_i \in V$ is a list of nodes such that two consecutive nodes are adjacent in G, i.e. $(v_i, v_{i+1}) \in E$. Note that edges can be traversed multiple times when walking along p. Where convenient, we also denote a path p by the corresponding list of edges.

As mentioned in the introduction, we consider the standard model for ad-hoc networks where all nodes have the same limited transmission ranges. This leads to the definition of the unit disk graph (UDG).

Definition 3.1. (Unit Disk Graph) *Let $V \subset \mathbb{R}^2$ be a set of points in the 2-dimensional plane. The Euclidean graph with edges between all nodes with distance at most 1 is called the unit disk graph.*

We also make the natural assumption that the distance between nodes is limited from below.

Definition 3.2. ($\Omega(1)$-model) *If the distance between any two nodes is bounded from below by a term of order $\Omega(1)$, i.e. there is a positive constant d_0 such that d_0 is a lower-bound on the distance between any two nodes, this is referred to as the $\Omega(1)$-model.*

This paper mainly focuses on geometric ad-hoc routing algorithms which can be defined as follows.

Definition 3.3. (Geometric Ad-Hoc Routing Algorithm) *Let $G = (V, E)$ be a Euclidean graph. The aim of a geometric ad-hoc routing algorithm $\mathcal{A}$ is to transmit a message from a source $s \in V$ to a destination $t \in V$ by sending packets over the edges of G while complying with the following conditions:*

- *Initially all nodes $v \in V$ know their geometric position as well as the geometric position of all of their neighbors in G.*
- *The source s knows the position of the destination t.*
- *The nodes are not allowed to store anything except for temporarily storing packets before transmitting them.*
- *The additional information which can be stored in a packet is limited by $O(\log n)$ bits, i.e. information about $O(1)$ nodes is allowed.*

In the literature geometric ad-hoc routing has been given various other names, such as $O(1)$-memory routing algorithm in [BM99, BMB+00], local routing algorithm in [KSU99] or position-based routing. Due to the storage restrictions, geometric ad-hoc routing algorithms are inherently local.

For our analysis we are interested in three different cost models: the link distance metric (the number of hops), the Euclidean distance metric (the total traversed Euclidean distance) and the energy metric (the total energy used). Each cost model implies an edge weight function. As already defined, the Euclidean length of an edge is denoted by $c_d(e)$. In the link distance metric all edges have weight 1 ($c_\ell(e) \equiv 1$), and the energy weight of an edge is defined as the square of the Euclidean length ($c_E(e) := {c_d}^2(e)$). The cost of a path $p = e_1, \ldots, e_k$ is defined as the sum of the costs of its edges:

$$c_\tau(p) := \sum_{i=1}^{k} c_\tau(e_i), \text{ for } \tau \in \{d, \ell, E\}.$$

The cost $c_\tau(\mathcal{A})$ of an algorithm $\mathcal{A}$ is defined analogously as the sum over the costs of all edges which are traversed during the execution of an algorithm on a particular graph G[4].

Lemma 3.1. *In the $\Omega(1)$-model, the Euclidean distance, the link distance, and the energy metrics of a path $p = e_1, \ldots, e_k$ are equal up to a constant factor on the unit disk graph*[5]*.*

Proof. The cost of p in the link distance metric is $c_\ell(p) = k$. We have that $d_0 \leq c_d(e) \leq 1$ for all edges $e \in E$. Therefore, the Euclidean distance and the energy costs of p are upper-bounded by k and lower-bounded by $c_d(p) \geq d_0 k$ and $c_E(p) \geq d_0^2 k$, respectively. □

4 AFR: Adaptive Face Routing

In this section, we describe our algorithm AFR which is asymptotically optimal for unit disk graphs in the $\Omega(1)$-model. Our algorithm is an extension of the *Face Routing* algorithm introduced by Kranakis *et al.* [KSU99] (in the original paper the algorithm is called *Compass Routing II*).

Face Routing and AFR work on planar graphs. We use the term planar graph for a specific embedding of a planar graph, i.e. we consider Euclidean planar

[4]For the energy metric it is usually assumed that a node can send a message simultaneously to different neighbors using only the energy corresponding to the farthest of those neighbors. We neglect this because it does not change our results.

[5]More generally, all metrics whose edge weight functions are polynomial in the Euclidean distance weight are equal up to a constant factor on the unit disk graph in the $\Omega(1)$-model. This formulation would include hybrid models as well as energy metrics with exponents other than 2.

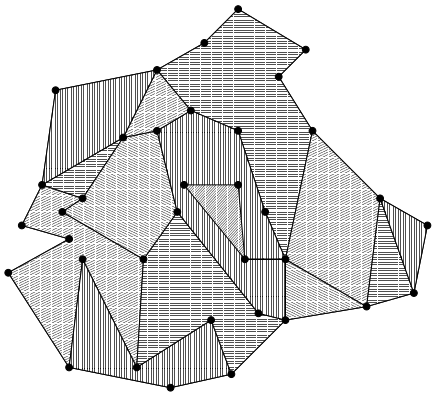

Figure 2: The faces of a planar graph (the white region is the infinite outer face).

graphs. In this case, the nodes and edges of a planar graph G partition the Euclidean plane into contiguous regions called the f faces of G (see Figure 2 as an illustration). Note that we get $f-1$ finite faces in the interior of G and one infinite face around G.

The main idea of the *Face Routing* algorithm is to walk along the faces which are intersected by the line segment $\overline{st}$ between the source s and the destination t. For completeness we describe the algorithm in detail (see Figure 3).

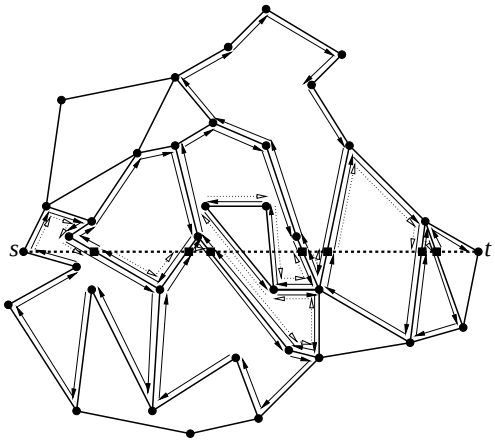

Figure 3: The *Face Routing* algorithm

Face Routing:

0. Start at s and let F be the face which is incident to s and which is intersected by $\overline{st}$ in the immediate region of s.

1. Explore the boundary of F by traversing its edges and remember the intersection point p of $\overline{st}$ with the edges of F which is nearest to t. After traversing all edges, go back to p. If we reach t while traversing the boundary of F, we are done.

2. p divides $\overline{st}$ into two line segments where $\overline{pt}$ is the not yet "traversed" part of $\overline{st}$. Update F to be the face which is incident to p and which is intersected by the line segment $\overline{pt}$ in the immediate region of p. Go back to step 1.

In order to simplify the subsequent proofs, we show that *Face Routing* terminates in linear time.

Lemma 4.1. *The* Face Routing *algorithm reaches the destination t after traversing at most* $O(n)$ *edges where n is the number of nodes.*

Proof. First we show that the algorithm terminates. By the choices of the faces F in step 0 and 2, respectively, we see that in step 1 we always find a point p which is nearer to t than the previous p where we start the tour around F. Therefore we are coming nearer to t with each iteration, and since there are only finitely many intersections between $\overline{st}$ and the edges of G, we reach t in a finite number of iterations.

For the performance analysis, we see that by choosing p as the $\overline{st}$-"face boundary" intersection which is nearest to t, we will never traverse the same face twice. Now, we partition the edges E into two subsets E_1 and E_2 where E_1 are the edges which are incident to only one face (the same face lies on both sides of the edge) and E_2 are the edges which are incident to two faces (the edge lies between two different faces). During the exploration of a face F in step 2, an edge of E_2 is traversed at most twice and an edge of E_1 is traversed at most four times. Since the edges of E_1 appear in only one face and the edges of E_2 appear in two faces, all edges of E are traversed at most four times during the whole algorithm. Each face in a planar connected graph (with at least 4 nodes) has at least three edges on its boundary. This together with

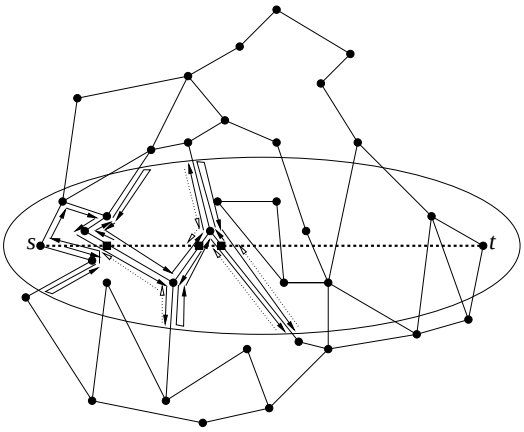

Figure 4: *Bounded Face Routing* (no success: $\widehat{c_d}$ is chosen too small)

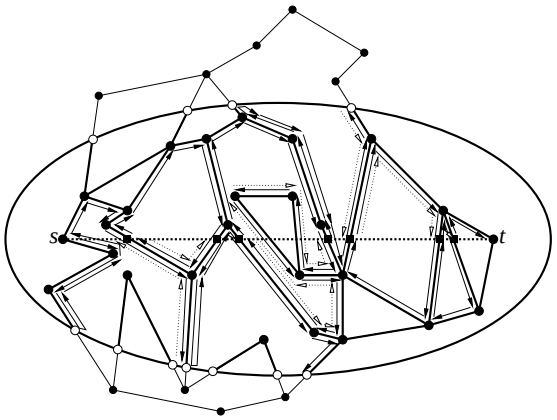

Figure 5: Successful *Bounded Face Routing*

the Euler polyhedral formula ($n - m + f = 2$) yields that the number of edges m is bounded by $m \leq 3n-6$ which proves the lemma. □

In order to obtain our new algorithm AFR, we are now going to change *Face Routing* in two steps. In a first step we assume that an upper-bound $\widehat{c_d}$ on the (Euclidean) length $c_d(p^*)$ of a shortest route p^* from s to t on graph G is known to s at the beginning. We present a geometric ad-hoc routing algorithm which reaches t with link distance cost at most $O(\widehat{c_d}^2)$.

Bounded Face Routing (BFR[$\widehat{c_d}$]): Let E be the ellipse which is defined by the locus of all points the sum of whose distances from s and t is $\widehat{c_d}$, i.e. E is an ellipse with foci s and t. By the definition of E, the shortest path (in $\mathbb{R}^2$) from s to t via a point q outside E is longer than $\widehat{c_d}$. Therefore, the best path from s to t on G is completely inside or on E. We change step 1 of *Face Routing* such that we always stay within E.

0. Start at s and let F be the face which is incident to s and which is intersected by $\overline{st}$ in the immediate region of s.

1. As before, we explore the face F and remember the best intersection between $\overline{st}$ and the edges of F in p. We start the exploration of F as in *Face Routing* by starting to walk into one of the two possible directions. We continue until we come around the whole face F as in the normal *Face Routing* algorithm or until we would cross the boundary of E. In the latter case, we turn around and walk back until we get to the boundary of E again. In any case we are then going back to p. If the exploration of F does not yield a better p, i.e. if p has the same value as in the previous iteration, *Bounded Face Routing* does not find a route to t and we restart BFR to find a route back from p to the source s. Otherwise, proceed with step 2.

2. p divides $\overline{st}$ into two line segments where $\overline{pt}$ is the not yet "traversed" part of $\overline{st}$. Update F to be the face which is incident to p and which is intersected by the line segment $\overline{pt}$ in the immediate region of p. Go back to step 1.

Figure 4 shows an example where $\widehat{c_d}$ is chosen too small, Figure 5 shows a successful execution of the *Bounded Face Routing* algorithm.

Lemma 4.2. *If the length of an optimal path p^* (w.r.t. the Euclidean distance metric) between s and t in graph G in the $\Omega(1)$-model is upper-bounded by a constant $\widehat{c_d} \geq c_d(p^*)$,* Bounded Face Routing *finds a path from s to t. If* Bounded Face Routing *does not succeed in finding a route to t, it does succeed in*

returning to s. In any case, Bounded Face Routing *terminates with link distance cost at most* $O(\widehat{c_d}^2)$.

Proof. We show that whenever there is a path from s to t which is completely inside or on E, *Bounded Face Routing* finds a route from s to t by traversing at most $O(\widehat{c_d}^2)$ edges. The lemma then follows.

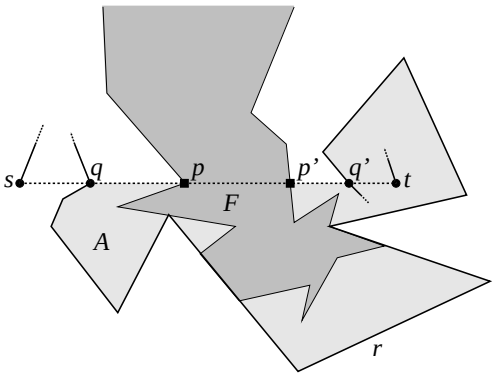

Figure 6: If there is a path from s to t inside E, BFR succeeds in routing from s to t (E is not drawn on the picture).

Suppose that there is a path r from s to t where r lies inside or on E. First we show that in this case BFR finds a route from s to t. Consider a point p on $\overline{st}$ from which we start to traverse a face F. We have to show that we find a point p' on $\overline{st}$ which is nearer to t than p while exploring face F. Assume that F does not completely lie inside the ellipse E since otherwise we find p' as in the normal *Face Routing* algorithm. Let q be the last intersection between path r and $\overline{st}$ before p and let q' be the first intersection between r and $\overline{st}$ after p (see Figure 6 as an illustration). The part of the path r which is between q and q' and the line segment $\overline{qq'}$ together define a polygon. We denote the area which is covered by this polygon by A. To traverse the boundary of F we can leave p in two possible directions where one of them points into A. During the traversal we will in any case take both directions. While walking along the boundary of F, we cannot cross the path r because the edges of r are part of the planar graph of which F is a face. In order to leave A, we therefore have to cross $\overline{st}$ at a point $p' \neq p$. p' must be nearer to t than p because otherwise the boundary of F would cross itself.

As a second step we show that each edge inside E is traversed at most four times during the execution of the BFR algorithm. In order to prove this, we consider the graph G' which is defined as follows. Prune everything of G which is outside the ellipse E. At the intersections between edges of G and E, we introduce new nodes and we take the segments of E between those new nodes as additional "edges"[6]. As an illustration, all edges of G' are drawn with thick lines in Figure 5. Now consider step 1 of BFR as exploring a face F of G' instead of exploring a face of G. Let p be the intersection between F and $\overline{st}$ where we start the traversal of F and let p' be the $\overline{st}$-"face boundary"-intersection which is closest to t. If there is a path from s to t inside E, there must also be a path between p and p' which is inside E. Assume that this is not the case. The part of the boundary of F which includes p and the part of the boundary of F which includes p' would then only be connected by the additional edges on E in G'. Thus, F would then separate E into two parts, one of which containing s, the other one containing t. Therefore step 1 of our algorithm BFR yields p' as a new point on $\overline{st}$, i.e. BFR is in a sense equivalent to *Face Routing* on G'. Hence, in an execution of BFR each face of G' is visited at most once. During the exploration of a face F in step 1 of BFR each edge is traversed at most twice, no matter if we walk around F as in the normal *Face Routing* algorithm or if we hit E and have to turn around (the edges whose sides belong to the same face can again be traversed four times). Therefore, we conclude that each edge inside E is traversed at most four times.

As a last step, we have to prove that there are only $O(\widehat{c_d}^2)$ edges of G inside E. Since G is a planar graph, we know that the number of edges is linear in the number of nodes ($m \leq 3n - 6$ as shown in the proof of Lemma 4.1). We consider the $\Omega(1)$-model where the Euclidean distance between any pair of nodes is at least d_0. Thus, the circles of radius $d_0/2$ around all nodes do not intersect each other. The length of the

[6]We do not consider that those additional edges are no straight lines. By adding some additional new nodes on E and connecting all new nodes by straight line segments, we could also construct G' to be a real Euclidean planar graph.

semimajor axis a of the ellipse E is $\widehat{c_d}/2$, and thus, the area E is smaller than πa^2. The number of nodes n' inside E is therefore bounded by

$$n' \leq \frac{\pi a^2}{\pi \left(\frac{d_0}{2}\right)^2} + \mathrm{O}(a) = \frac{\widehat{c_d}^2}{d_0^2} + \mathrm{O}(a) \in \mathrm{O}\left(\widehat{c_d}^2\right).$$

We have now proven that if there is a path from s to t inside E, then algorithm BFR terminates after traversing at most $\mathrm{O}(\widehat{c_d}^2)$ edges. The only thing which remains open in order to conclude the proof of Lemma 4.2 is that an unsuccessful execution of BFR also terminates after traversing at most $\mathrm{O}(\widehat{c_d}^2)$ edges. Let F_i, $1 \leq i \leq k$, be the faces which are visited during the execution of the algorithm. F_k is the face where we do not find a better point on $\overline{st}$, i.e. F_k is the face which divides E into two parts. From the above analysis it is clear that the first $k-1$ faces are only visited once. F_k is explored at most twice, once to find the best accessible intersection with $\overline{st}$ and once to see that no further improvement can be made. Hence, all edges are traversed at most eight times until we arrive at the point p on $\overline{st}$ where we have to turn around[7]. For our way back we know that there is a path from p to s which lies inside E and therefore we arrive at s after visiting every edge at most another four times. □

We are now coming to the definition of AFR. The problem with *Bounded Face Routing* is that usually no upper-bound on the length of the best route is known. In AFRwe apply a standard trick to get around this.

AFR Adaptive Face Routing We begin by determining an estimate $\widetilde{c_d}$ for the unknown value $\mathrm{c_d}(p^*)$, e.g. $\widetilde{c_d} := 2\overline{st}$. The algorithm then runs *Bounded Face Routing* with exponentially growing $\widetilde{c_d}$ until eventually the destination t is reached:

1. Execute BFR$[\widetilde{c_d}]$.
2. If the BFR execution of step 1 succeeded, we are done; otherwise, we double the estimate for the length of the shortest path ($\widetilde{c_d} := 2\widetilde{c_d}$) and go back to step 1.

[7] It is possible to explore face F_k only once as well but for our asymptotic analysis, we ignore this optimization.

Lemma 4.3. *Let p^* be a shortest path from node s to node t on the planar graph G.* Adaptive Face Routing *finds a path from s to t while traversing at most $\mathrm{O}(\mathrm{c_d}^2(p^*))$ edges.*

Proof. We denote the first estimate $\widetilde{c_d}$ on the optimal path length by $\widetilde{c}_{d,0}$ and the consecutive estimates by $\widetilde{c}_{d,i} := 2^i \widetilde{c}_{d,0}$. Furthermore, we define k such that $\widetilde{c}_{d,k-1} < \mathrm{c_d}(p^*) \leq \widetilde{c}_{d,k}$. For the cost of BFR$[\widetilde{c_d}]$ we have $\mathrm{c}_\ell(\mathrm{BFR}[\widetilde{c_d}]) \in \mathrm{O}(\widetilde{c_d}^2)$ and therefore

$$\mathrm{c}_\ell(\mathrm{BFR}[\widetilde{c_d}]) \leq \lambda \cdot \widetilde{c_d}^2$$

for a constant λ (and sufficiently large $\widetilde{c_d}$). The total cost of algorithm AFR can therefore be bounded by

$$\begin{aligned}
\mathrm{c}_\ell(\mathrm{AFR}) &\leq \sum_{i=0}^{k} \mathrm{c}_\ell(\mathrm{BFR}[\widetilde{c}_{d,i}]) \leq \sum_{i=0}^{k} \lambda \left(2^i \widetilde{c}_{d,0}\right)^2 \\
&= \lambda \widetilde{c}_{d,0}^2 \frac{4^{k+1}-1}{3} < \frac{16}{3} \lambda \left(2^{k-1} \widetilde{c}_{d,0}\right)^2 \\
&< \frac{16}{3} \lambda \cdot \mathrm{c_d}^2(p^*) \in \mathrm{O}\left(\mathrm{c_d}^2(p^*)\right).
\end{aligned}$$

□

For the remainder of this section we show how to apply AFR to the unit disk graph. We need a planar subgraph of the unit disk graph, since AFR requires a planar graph. There are various suggestions on how to construct a planar subgraph of the unit disk graph in a distributed way. Often the intersection between the UDG and the relative neighborhood graph (RNG [Tou80]) or the Gabriel Graph (GG [GS69]), respectively, have been proposed. In the RNG an edge between nodes u and v is present iff no other node w is closer to u and to v than u is to v. In the gabriel graph an edge between u and v is present iff no other node w is inside or on the circle with diameter $\overline{uv}$. The Relative Neighborhood Graph and the Gabriel Graph are easily constructed in a distributed manner. There have been other suggestions, such as the intersection between the Delaunay triangulation and the unit disk graph [LCW02]. All mentioned graphs are connected provided that the unit disk graph is connected as well. We use the Gabriel Graph, since it meets all requirements as shown in the following lemma.

Lemma 4.4. *In the $\Omega(1)$-modelthe shortest path for any of the considered metrics (Euclidean, link distance, and energy) on the Gabriel Graph intersected with the unit disk graph is only by a constant longer than the shortest path on the unit disk graph for the respective metric.*

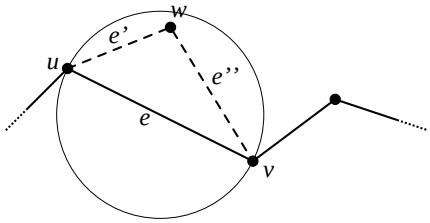

Figure 7: The unit disk graph contains an energy optimal path.

Proof. We show that at least one best path with respect to the energy metric on the UDG is also contained in $\mathrm{GG} \cap \mathrm{UDG}$. Suppose that $e = (u, v)$ is an edge of an energy optimal path p on the UDG. Further suppose that e is not contained in $\mathrm{GG} \cap \mathrm{UDG}$. Then there is a node w in or on the circle with diameter $\overline{uv}$ (see Figure 7). The edges $e' = (u, w)$ and $e'' = (v, w)$ are also edges of the UDG and because w is in the described circle, we have $e'^2 + e''^2 \leq e^2$. If w were inside the circle with diameter $\overline{uv}$, the energy for the path $p' := p \setminus \{e\} \cup \{e', e''\}$ would be smaller than the energy of p and p no energy optimal path. If w is on the above circle, p' is an energy optimal path as well and the argument applies recursively. Using Lemma 3.1, we see that the optimal path costs with respect to the Euclidean and the link distance metrics are only by a constant factor greater than the energy cost of p. This concludes the proof. □

Lemma 4.4 directly leads to Theorem 4.5.

Theorem 4.5. *Let p^*_τ for $\tau \in \{\mathrm{d}, \ell, \mathrm{E}\}$ be an optimal path with respect to the corresponding metric on the unit disk graph in the $\Omega(1)$-model. We have*

$$\forall \tau \in \{\mathrm{d}, \ell, \mathrm{E}\} : \mathrm{c}_\tau(\mathrm{AFR}) \in \mathrm{O}\big(\mathrm{c}^2_\tau(p^*_\tau)\big)$$

when applying AFR on $\mathrm{GG} \cap \mathrm{UDG}$ in the $\Omega(1)$-model.

Proof. The theorem directly follows from Lemma 3.1, Lemma 4.3, and Lemma 4.4. □

5 Lower Bound

In this section we give a constructive lower bound for geometric ad-hoc routing algorithms.

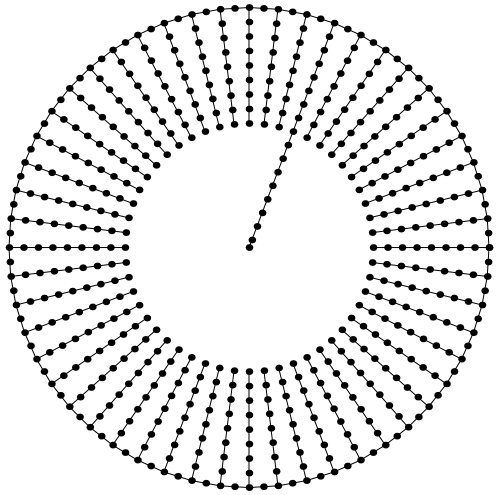

Figure 8: Lower bound graph

Theorem 5.1. *Let the cost of a best route for a given source destination pair be c. Then any deterministic (randomized) geometric ad-hoc routing algorithm has (expected) cost $\Omega(c^2)$ for link, distance, or energy cost.*

Proof. We construct a family of networks as follows. We are given a positive integer k and define a Euclidean graph G (see Figure 8): On a circle we evenly distribute $2k$ nodes such that the distance between two neighboring points is exactly 1; thus, the circle has radius $r \approx k/\pi$. For every second node of the circle we construct a chain of $\lceil r/2 \rceil - 1$ nodes. The nodes of such a chain are arranged on a line pointing towards the center of the circle; the distance between two neighboring nodes of a chain is exactly 1. Node w is one arbitrary circle node with a chain: The chain of w consists of $\lceil r \rceil$ nodes with distance 1. The last node of the chain of w is the center node; note that

the edge to the center node does not need to have distance 1.

Please note that the unit disk graph consists of the edges on the circle and the edges on the chains only. In particular, there is no edge between two chains because all chains except the w chain end strictly outside radius $r/2$. Note that the graph has k chains with $\Theta(k)$ nodes each.

We route from an arbitrary node on the circle (the source s) to the center of the circle (the destination t). An optimal route between s and t follows the shortest path on the circle until it hits node w, and then directly follows w's chain to t with link cost $c \leq k + r + 1 \in \mathrm{O}(k)$. An ad-hoc routing algorithm with routing tables at each node will find this best route.

A geometric ad-hoc routing algorithm needs to find the "correct" chain w. Since there is no routing information stored at the nodes, this can only be done by exploring the chains. Any deterministic algorithm needs to explore the chains in a deterministic order until it finds the chain w. Thus, an adversary can always place w such that w's chain will be explored as the last one. The algorithm will therefore explore $\Theta(k^2)$ (instead of only $\mathrm{O}(k)$) nodes.

The argument is similar for randomized algorithms. By placing w accordingly (randomly!), an adversary forces the randomized algorithm to explore $\Omega(k)$ chains before chain w with constant factor probability. Then the expected link cost of the algorithm is $\Omega(k^2)$.

Because all edges (but one) in our construction have length 1, the costs in the Euclidean distance, the link distance, and the energy metrics are equal. Thus, the $\Omega(c^2)$ lower bound holds for all three metrics. □

Note that our lower bound does hold generally, not only for $\Omega(1)$-graphs. However, if the graph is not an $\Omega(1)$ graph, there might be a higher (worse) lower bound.

To conclude this section, we present the main theorem of this paper stating that AFR is asymptotically optimal for unit disk graphs in the $\Omega(1)$-model.

Theorem 5.2. *Let c be the cost of an optimal path for a given source destination pair on a unit disk graph in the $\Omega(1)$-model. In the worst case the cost for applying AFR to find a route from the source to the destination is $\Theta(c^2)$. This is asymptotically optimal.*

Proof. Theorem 5.2 is an immediate consequence of Theorem 4.5 and of Theorem 5.1. □

6 Conclusion

In this paper we proved a lower bound for geometric ad-hoc routing algorithms on the unit disk graph. Specifically, we showed that in the worst case the cost of any geometric ad-hoc routing algorithm is quadratic in the cost of an optimal path. This result holds for the Euclidean distance, the link distance, and the energy metric. Furthermore, we gave an algorithm (AFR) which matches this lower bound and is therefore optimal.

It is interesting to see that if we allow the nodes to store $\mathrm{O}(\log n)$ bits, we can achieve the same results even if the source does not know anything about the coordinates of the destination. The lower bound still holds and the upper bound can be achieved by a simple flooding algorithm. The source floods the network (we again take GG $\cap$ UDG) with an initial time to live ttl_0, i.e. all nodes up to depth ttl_0 are reached. The result of the flood (destination reached or not reached) is then echoed back to the source along the same paths in the reverse direction. We iterate the process with exponentially growing time to live until we reach the destination. All nodes which are reached by flooding with TTL ttl are in a circle with radius ttl around the source. In this circle there are $\mathrm{O}(ttl^2)$ nodes and hence also $\mathrm{O}(ttl^2)$ edges each of which is traversed at most 4 times (including the echo process). Therefore, the cost of iteration i (with TTL ttl_i) is $\mathrm{O}(ttl_i^2)$ and the cost of the whole algorithm is quadratic in the cost of the best path for any of the three considered metrics. We find it intriguing that a few storage bits in each node appear to be as good as the geometric information about the destination.

References

[BM99] P. Bose and P. Morin. Online routing in triangulations. In *10th International Symposium on Algorithms and Computation (ISAAC)*, volume 1741 of Springer LNCS, pages 113–122, 1999.

[BMB+00] P. Bose, P. Morin, A. Brodnik, S. Carlsson, E.D. Demaine, R. Fleischer, J.I. Munro, and A. Lopez-Ortiz. Online routing in convex subdivisions. In *International Symposium on Algorithms and Computation (ISAAC)*, pages 47–59, 2000.

[BMJ+98] J. Broch, D.A. Maltz, D.B. Johnson, Y.-C. Hu, and J.Jetcheva. A performance comparison of multi-hop wireless ad hoc network routing protocols. In *Mobile Computing and Networking*, pages 85–97, 1998.

[BMSU99] P. Bose, P. Morin, I. Stojmenovic, and J. Urrutia. Routing with guaranteed delivery in ad hoc wireless networks. In *Proc. of Discrete Algorithms and Methods for Mobility (DIALM'99)*, pages 48–55, 1999.

[DFS90] D. Dobkin, S.J. Friedman, and K.J. Supowit. Delaunay graphs are almost as good as complete graphs. *Discrete & Computational Geometry*, 5:399–407, 1990.

[DSW01] S. Datta, I. Stojmenovic, and J. Wu. Internal node and shortcut based routing with guaranteed delivery in wireless networks. In *Proc. IEEE Int. Conf. on Distributed Computing and Systems Workshops; Cluster Computing, to appear*, pages 461–466, 2001.

[Fin87] G.G. Finn. Routing and addressing problems in large metropolitan-scale internetworks. Technical Report ISI/RR-87-180, USC/ISI, March 1987.

[GS69] K.R. Gabriel and R.R. Sokal. A new statistical approach to geographic variation analysis. *Systematic Zoology*, 18:259–278, 1969.

[GSB01] S. Giordano, I. Stojmenovic, and L. Blazevic. Position based routing algorithms for ad hoc networks: a taxonomy, July 2001.

[HB01] J. Hightower and G. Borriella. Location systems for ubiquitous computing. *IEEE Computer*, 34(8):57–66, 2001.

[HL86] T.C. Hou and V.O.K. Li. Transmission range control in multihop packet radio networks. *IEEE Transactions on Communications*, 34(1):38–44, 1986.

[IN96] T. Imielinski and J. Navas. Gps-based addressing and routing. Technical Report RFC 2009, Computer Science, Rutgers University, November 1996.

[KSU99] E. Kranakis, H. Singh, and J. Urrutia. Compass routing on geometric networks. In *Proc. 11th Canadian Conference on Computational Geometry*, pages 51–54, Vancouver, August 1999.

[KV98] Y. Ko and N. Vaidya. Geocasting in mobile ad hoc networks: Location-based multicast algorithms, 1998.

[LCW02] X.-Y. Li, G. Calinescu, and P.-J. Wan. Distributed construction of planar spanner and routing for ad hoc wireless networks. In *IEEE INFOCOM*, 2002.

[LJD+00] J. Li, J. Jannotti, D. De Couto, D. Karger, and R. Morris. A scalable location service for geographic ad-hoc routing. In *MobiCom '00*, pages 120–130, August 2000.

[MWH01] M. Mauve, J. Widmer, and H. Hartenstein. A survey on position-based routing in mobile ad-hoc networks. In *IEEE Network*, 2001.

[NI97] J.C. Navas and T. Imielinski. Geocast - geographic addressing and routing. In *Mobile Computing and Networking*, pages 66–76, 1997.

[RS96] S. Ramanathan and M. Steenstrup. A survey of routing techniques for mobile communications networks. *Mobile Networks and Applications*, 1(2):89–104, 1996.

[RT99] E. Royer and C. Toh. A review of current routing protocols for ad-hoc mobile wireless networks. In *IEEE Personal Communications*, April 1999.

[SHS01] A. Savvides, C.-C. Han, and M. Srivastava. Dynamic fine-grained localization in ad-hoc networks of sensors. In *MOBICOM '01*, pages 166–179, July 2001.

[TK84] H. Takagi and L. Kleinrock. Optimal transmission ranges for randomly distributed packet radio terminals. *IEEE Transactions on Communications*, 32(3):246–257, 1984.

[Tou80] G. Toussaint. The relative neighborhood graph of a finite planar set. *Pattern Recognition*, 12(4):261–268, 1980.

[Urr02] J. Urrutia. Routing with guaranteed delivery in geometric and wireless networks. In I. Stojmenovic, editor, *Handbook of Wireless Networks and Mobile Computing*, chapter 18, pages 393–406. John Wiley & Sons, 2002.

A AFR Implementation

This appendix contains a detailed distributed implementation of the AFR algorithm in pseudocode notation. On each network node an instance of the algorithm runs independently. Basically each instance waits to react to messages arriving on the network. The message structure is defined as follows:

```
Message {
  State state; // message state
  Point s, t; // source, destination
  boolean cw; // sense of face traversal
  int cDTilde; // ellipse size estimate
  Point pStart, pNext; // intersection points on the line segment st
  int hopsBeforeP, hopsAfterP; // hop counters
  IterationState iterationState; // overall iteration state
}
```

We assume that a number of primitive operations and structures are offered by an underlying software layer for use in our algorithm. In particular, we employ the data structures `Point` and `Edge` representing network node locations or intersection points and edges between network nodes, respectively. We denote by `thisPoint` the position of the network node on which the considered instance of the algorithm is running. Additionally, we define `LineSegment(Point a, Point b)` to describe the line segment between the two points `a` and `b`. We also define the value set types `State`, containing the three values `FACE`, `BOUNDEDFACE`, and `FORWARD`, and `IterationState`, consisting of the values `NORMAL`, `FAILED`, and `DISCONNECTED`.

`State` contains values describing the state of the traveling message in the course of a BFR iteration of the algorithm. Whenever the message is in the `FACE` state, the face traversal phase of the Face Routing algorithm is being performed. After possibly "hitting" the ellipse, the message changes to the `BOUNDEDFACE` state. Whenever the message simply should be forwarded to `m.pNext`, i.e. having fully explored a face or after "hitting" the ellipse for the second time, the message is in `FORWARD` state.

`IterationState` consists of values denoting the overall state of a BFR iteration. If the source receives a return message with `iterationState == FAILED`, the next iteration of AFR with doubled `cDTilde` should be started. `DISCONNECTED` informs the source that the network is disconnected in such a way that the destination is not reachable. In non-error cases the iteration state is `NORMAL`.

In addition to the above structures we define the following primitive operations employed by the algorithm implementation:

```
Message newMessage(State state, Point s, Point t, ... ,
                   IterationState iterationState);
  // factory method allowing for easy creation of a new message
send(Message m, Edge e); // send message m over edge e
```

```
receive(Message m, Edge e);
  // wait for receipt of any message m on edge e, both being
  // read-out parameters
Point nextHop(Edge e); // returns the position of the node
                       // adjacent to thisPoint via edge e
Edge getNextFaceEdge(Point a, Point b);
  // returns one incident edge of the face intersected by the line
  // segment ab in the immediate environment of thisPoint
Edge getAdjacentEdge(Edge e, boolean cw);
  // returns the edge a message arriving on edge e has to be
  // forwarded to for clockwise (cw == true) or counter-clockwise
  // traversal of the corresponding face adjacent to e
dist(Point a, Point b);
  // computes the Euclidean distance between two points a and b
Point intersect(LineSegment s, Edge e);
  // returns the intersection point of the line segment s and
  // the edge e or null if there is no such intersection
boolean pointOnEdge(Point p, Edge e);
  // returns true iff the point p lies on the edge e;
  // both of e's endpoint nodes lie on e
```

Note that all of these operations run locally on each network node and only use information collected from directly neighboring nodes. Also observe that a node will forward the message to itself exactly once if an intermediate destination (`pStart` or `pNext`) lies on the edge to be traversed next and the face traversal direction has to be changed. Finally, note that an error message will return to the source without changing its `iterationState`, since there always exists a path back to the source within the currently considered ellipse.

```
Edge nextEdge;
boolean stopped = false;

while (!stopped) {

  receive(m, e);

  if (thisPoint == m.t) {

    if (m.iterationState == NORMAL) {
      //
      // we have reached the destination
      //
    } else if (m.iterationState == FAILED) {
      // we are back at the original source s,
      // this is a failure message,
```

```
    // restart algorithm with doubled cDTilde
    m = newMessage(FACE, m.t, m.s, true, 2*m.cDTilde,
                   thisPoint, thisPoint, 0, 0, NORMAL);
    nextEdge = getNextFaceEdge(thisPoint, m.t);
    send(m, nextEdge);
  } else if (m.iterationState == DISCONNECTED) {
    // the graph is disconnected
  }
  continue;
}

switch (m.state) {
case FACE: // traditional face routing
  nextEdge = getAdjacentEdge(e, m.cw);
  if (pointOnEdge(m.pStart, nextEdge)) {
    // we have circled the whole face
    if (m.pNext == m.pStart) {
      // graph is disconnected
      // send failure report to m.s
      m = newMessage(FACE, m.t, m.s, true, m.cDTilde,
                     m.pStart, m.pStart, 0, 0, DISCONNECTED);
      nextEdge = getNextFaceEdge(m.pStart, m.s);
    } else {
      if (m.hopsAfterP <= m.hopsBeforeP) {
        nextEdge = e;
        m.cw = !m.cw;
      }
      m.state = FORWARD;
    }
  } else { // we are on our way around the face
    if ((p = intersect(LineSegment(m.pStart, t),
                       nextEdge)) != null) {
      // we have found an intersection p
      if (dist(p, m.t) < dist(m.pNext, m.t)) {
        // the new p is closer
        m.pNext = p;
        m.hopsBeforeP = m.hopsBeforeP + m.hopsAfterP;
        m.hopsAfterP = 1;
      } else
        m.hopsAfterP++;
    } else {
      Point pNextHop = nextHop(nextEdge);
      if ((dist(pNextHop, m.s) +
           dist(pNextHop, m.t)) > m.cDTilde) {
        // the next hop would lie outside the ellipse
        m.state = BOUNDEDFACE;
```

```
        m.cw = !m.cw;
        nextEdge = e;
      } else
        m.hopsAfterP++;
    }
  }
  break;

case BOUNDEDFACE: // bounded face routing
  nextEdge = getAdjacentEdge(e, m.cw);
  if ((p = intersect(LineSegment(m.pStart, t),
                     nextEdge)) != null) {
    // we have found an intersection p
    if (dist(p, m.t) < dist(m.pNext, m.t))
      // the new p is closer
      m.pNext = p;
  } else {
    Point pNextHop = nextHop(nextEdge);
    if ((dist(pNextHop, m.s) +
         dist(pNextHop. m.t)) > m.cDTilde) {
      // the next hop would lie outside the ellipse
      m.state = FORWARD; // travel to m.pNext
      m.cw = !m.cw;
      nextEdge = e;
      if (m.pNext == m.pStart)
        m.iterationState = FAILED;
        // no path within this ellipse
    }
  }
  break;

case FORWARD: // just forward the packet to m.pNext
  nextEdge = getAdjacentEdge(e, m.cw);
  if (pointOnEdge(m.pNext, nextEdge)) { // we are here
    if (m.iterationState == NORMAL) {
      // start next face traversal
      m = newMessage(FACE, m.s, m.t, true, m.cDTilde,
                     m.pNext, m.pNext, 0, 0, NORMAL);
      nextEdge = getNextFaceEdge(m.pNext, m.t);
    } else {
      // iterationState == FAILED,
      // send error message to m.s
      m = newMessage(FAILED, t, s, true, m.cDTilde,
                     m.pNext, m.pNext, 0, 0, FAILED);
      nextEdge = getNextFaceEdge(m.pNext, m.s);
    }
```

```
    } else
      nextEdge = getAdjacentEdge(e, m.cw);
  }

  send(m, nextEdge);
}
```