Route Discovery with Constant Memory in Oriented Planar Geometric Networks*

January 31, 2006

E. CHÁVEZ[1], S. DOBREV[2], E. KRANAKIS[3], J. OPATRNY[4]

L. STACHO[5] and J. URRUTIA[6]

Abstract

*This is a revised and expanded version of a paper that appeared in proceedings of Algosensors 2004, Springer Verlag, LNCS, Vol. 3121, pp. 147-156, S. Nikoletseas, J. Rolim, editors.

[1]Escuela de Ciencias Fisico-Matemáticas de la Universidad Michoacana de San Nicolás de Hidalgo, México.

[2]School of Information Technology and Engineering (SITE), University of Ottawa, 800 King Eduard, Ottawa, Ontario, Canada, K1N 6N5. Research supported in part by NSERC (Natural Science and Engineering Research Council of Canada) grant.

[3]School of Computer Science, Carleton University, 1125 Colonel By Drive, Ottawa, Ontario, Canada K1S 5B6. Research supported in part by NSERC (Natural Science and Engineering Research Council of Canada) and MITACS (Mathematics of Information Technology and Complex Systems) grants.

[4]Department of Computer Science, Concordia University, 1455 de Maisonneuve Blvd West, Montréal, Québec, Canada, H3G 1M8. Research supported in part by NSERC (Natural Science and Engineering Research Council of Canada) grant.

[5]Department of Mathematics, Simon Fraser University, 8888 University Drive, Burnaby, British Columbia, Canada, V5A 1S6. Research supported in part by NSERC (Natural Science and Engineering Research Council of Canada) grant.

[6]Instituto de Matemáticas, Universidad Nacional Autónoma de México, Área de la investigación cientifica, Circuito Exterior, Ciudad Universitaria, Coyoacán 04510, México, D.F. México.

We address the problem of discovering routes in strongly connected planar geometric networks with directed links. Motivated by the necessity for establishing communication in wireless ad hoc networks in which the only information available to a vertex is its immediate neighborhood, we are considering routing algorithms that use the neighborhood information of a vertex for routing with constant memory only. We solve the problem for three types of directed planar geometric networks: *Eulerian* (in which every vertex has the same number of incoming and outgoing edges), *Outerplanar* (in which a single face contains all vertices of the network) and *Strongly Face Connected*, a new class of geometric networks which we define in the paper, consisting of several faces, each face being a strongly connected outerplanar graph.

Keywords: constant memory routing, oriented planar network, Eulerian network, face routing

1 Introduction

A wireless ad hoc network is a collection of vertices in which a vertex can communicate directly with the vertices within its broadcast range. In ad hoc networks, no communication infrastructure is available and communication with a vertex outside the broadcast range is achieved by multi-hop routing, whereby intermediate vertices cooperate by forwarding packets to the destination. If no assumption on the positions of the vertices of the network can be made then routing may have to be done by *flooding* the network with packets, which may increase overall con-

gestion in the network and thereby cause unnecessary waste of resources, such as energy consumption. To avoid flooding, several authors proposed the use of *location information* for routing of packets, where each vertex knows its own location and the location of the destination vertex.

A position-based ad hoc network can be represented by a geometric graph in which vertices correspond to their locations in the plane and edges are straight lines connecting any pair of vertices that can communicate directly. In many wireless networks it is assumed that any communication link between vertices is bidirectional in the sense that packets may flow in any direction between any pair of adjacent vertices. In this case such a network can be represented by an undirected geometric graph. This type of connectivity arises in a network in which all wireless hosts have the same transmission power, i.e., identical transmission ranges, in which the network is called unit disk graph.

Most studies on routing in ad hoc networks have dealt with undirected geometric graphs. A most important problem is how to discover a route (among the many potential candidates) that must accommodate three rather contradictory goals: (1) avoiding flooding the network, (2) minimizing the number of hops in the path, (3) using only geographically local information. Although it is generally accepted that flooding must be avoided in order to increase the network lifetime, limitations on the knowledge of the networks make it necessary that in practice one may need to relax the second goal in favour of the third. Indeed, this is the case in geometric planar networks whereby algorithms are given for discovering routes (see Kranakis et al. [8]) which do not minimize the number of hops but

guarantee the delivery of the packets. Thus another important consideration is to construct a planar geometric network from a given, possibly more complicated, wireless network. This question is addressed in Bose et al. [5] as well as in several subsequent papers including Li et al. [11, 12]. The basic idea is to locally preprocess the wireless network in order to abstract a planar geometric network over which the algorithm in Kranakis et al. [8] can be applied. There has been extensive literature related to discovering routes in position-based, wireless ad hoc networks when the underlying graph is an undirected planar geometric network, e.g., see Bose et al. [5], Kranakis et al. [8], Kuhn et al. [9, 10]. A problem related to routing is traversal which is addressed in several papers: Avis and Fukuda [1], Bose and Morin [4], Chavez et al. [6], Gold et al. [7], Peuquet and Marble [13, 14]. However, traversing all the vertices of a graph may be an "overkill" especially if all one requires is a single path from a source to a destination host.

The bidirectional communication assumption represents an "idealistic" scenario that is not necessarily valid in general ad hoc networks. This is due to several factors that may cause "communication asymmetries", including obstacles that may create signal interference and/or diminish the strength of a signal during propagation in an asymmetric manner or even some wireless hosts with diminishing transmission ranges. Such scenarios have been modelled in the literature (see Barriere et al. [2]). In these situations a vertex can be fully aware of the positions of its neighbors, but it may not be able to send messages to some of its neighbors while it can receive messages from them, or the other way around. Therefore a fundamental issue is whether for such a graph it is possible to have

a deterministic routing algorithm that for each vertex takes into account only the positions of its neighbors, uses only constant memory, and at the same time guarantees delivery of a packet. If the underlying network is either not planar or the links are not necessarily bidirectional the techniques outlined above may fail to route and/or traverse the network using only "geographically" local information. In general, routing in arbitrary graphs should be expected to be a difficult problem because it may be difficult to avoid "loops" based only on geographically local information. Therefore there is a need for reexamination of the structure of the underlying network that gives rise to our basic communication model in order to accomplish routing satisfying the previously stated conditions.

In this paper we make a first step toward addressing the problem of discovering routes in networks that can be represented by strongly connected planar geometric graphs whose vertices correspond to their locations in the plane and whose links are directed edges. We assume that at any given time each vertex either has or can acquire information on the positions of the vertices connected to it by its incoming and outgoing edges. In addition, we only require that routing decisions be made using this local information available at each vertex for routing.

There are several situations where such a scenario of unidirectional links may arise. For example, suppose that vertices have already established bidirectional links with neighbors. Such links may then may be disturbed because of any of the following reasons.

1. Due to irregularities in transmission ranges, power supply quality, signal interference, etc., the signal that vertex u gets from some vertices might not

be strong enough to receive messages without errors while these vertices can receive signals from u without problems. Thus vertices may agree on a one-way transmission in these cases.

2. If a vertex receives packets for re-transmissions from all vertices in its neighborhood, it could create a large amount of traffic in the vertex, more than it can handle. Thus a vertex can decide to receive messages only from some vertices of its neighborhood and send messages only to some of them.

3. A vertex could be required not to accept any message from some neighbors for security reasons (untrusted neighbors), but it can send them messages. For the same security reasons a vertex might send out messages in a coded form that is understood by some neighbor vertices only and is necessarily ignored by other neighbors.

It is worth mentioning that all the current literature on routing and localization is highly dependent on planarity and bidirectional links of the underlying spanning subgraph (spanner). Overcoming these assumptions has never been addressed in the past. Therefore we believe that our approach is a step in the right direction toward addressing the problem of discovering routes locally in more complex network settings.

An outline of the paper is as follows. After specifying the model in Section 2, we consider some specific types of directed planar geometric networks. In Section 3, we look at Eulerian planar geometric networks in which every vertex has the same number of incoming and outgoing edges. In Section 4, we investigate

outerplanar geometric networks whereby a single face contains all the vertices of the network. This is followed by Section 5 in which we solve the routing problem on a new class of planar geometric networks, called *strongly face connected*, consisting of several faces, each face being a strongly connected outerplanar graph.

2 Model

A *planar geometric network* is a planar graph $G = (V, E, F)$ with vertex set V, edge set E and the face set F together with its straight line embedding into the plane $\mathbb{E}^2$. If we do not need an explicit reference to F, sometimes we denote G as $G = (V, E)$ only. In this paper, we always consider finite graphs. Furthermore, we assume that no edge passes through any vertex except its end-vertices. A geometric network is *connected* if its graph is connected. An *orientation* $\vec{}$ of a planar geometric network G is an assignment of a direction to every edge e of G. For an edge e with endpoints u and v, we write $e = (u, v)$ if its direction is from u to v. The geometric network together with its orientation is denoted by $\vec{G}$. A geometric network with a given orientation is *strongly connected* if for every ordered pair of its vertices, there is a (directed) path joining them.

We assume that every vertex v is assigned a pair $[x, y]$ of coordinates in the plane where x is its horizontal and y is its vertical coordinate.

Consider a connected planar geometric network $\vec{G} = (V, E)$. We say $\vec{G}$ is *Eulerian* if for each vertex $u \in V$, the size of $N^+(u) = \{x : (u, x) \in E\}$ equals the size of $N^-(u) = \{y : (y, u) \in E\}$; i.e., the number of edges leaving u equals

the number of edges entering u. A planar geometric network G is *outerplanar* if one of the faces (called the outerface) contains all the vertices. A *strongly face connected* network $\vec{G} = (V, E, F)$ is obtained from a geometric planar network by adding new directed edges into the faces of the latter so that every such face (except the outerface) together with the added edges induces a strongly connected outerplanar graph in $\vec{G}$.

3 Route Discovery in Oriented Eulerian Planar Geometric Networks

First consider a planar geometric network G without any orientation. Given a vertex v on a face f in G, the boundary of f can be traversed in the counterclockwise (clockwise if f is the outer face) direction using the well-known right hand rule [3] which states that it is possible to visit every wall in a maze by keeping your right hand on the wall while walking forward. Treating this face traversal technique as a subroutine, Kranakis et al. [8] give an algorithm for routing in a planar geometric network from a vertex s to a vertex t in which a vertex on the route forwards the packet to exactly one of its neighbors using only information about the immediate neighborhood of the vertex. This algorithm guarantees delivery of the packet to the destination if the underlying graph is connected.

If we impose an orientation $\vec{}$ on G, then this algorithm does not work in all cases since some edges may be directed in a direction opposite to the direction in which a face is being traversed. In this section, we describe a simple technique to

overcome this difficulty if the directed graph satisfies some conditions. In particular, we propose a method for routing a message to the other end of an oppositely directed edge in Eulerian geometric networks.

Suppose now that $\vec{}$ is an orientation such that $\vec{G}$ is an Eulerian planar geometric network. For a given vertex u of $\vec{G}$, we order edges (u, x) where $x \in N^+(u)$ clockwise around u starting with the edge forming the smallest angle with the vertical line passing through u. Similarly, we order edges (y, u) where $y \in N^-(u)$ clockwise around u; see Figure 1. Clearly, these orderings are unique and can be determined locally at each vertex.

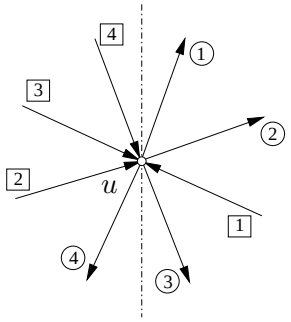

Figure 1: Circled numbers represent the ordering on outgoing edges, squared numbers on incoming ones.

We define the function $\mathbf{succ}()$ on the incoming edges at a vertex u as follows. If e is the i-th incoming edge into u in $\vec{G}$ then $\mathbf{succ}(e)$ is the i-th outgoing edge from u. For an illustration of the function see Figure 2. Clearly, this function is easy to compute using only local information.

Obviously, the function $\mathbf{succ}()$ is a bijection from the set of incoming edges to the set of outgoing edges at each vertex u of G. Thus for every edge $e = (u, v)$

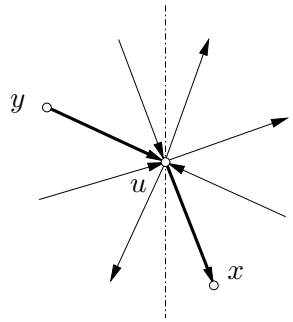

Figure 2: In this example the incoming edge (y, u) is third, so the chosen outgoing edge (u, x) is also third. Both these edges are depicted bold.

of $\vec{G}$, we can define a closed walk by starting from $e = (u, v)$ and then repeatedly applying the function $\mathbf{succ}()$ until we arrive at the same edge $e = (u, v)$. Since $\vec{G}$ is Eulerian, the walk is well defined and finite. We call such a walk a *quasi-face* of $\vec{G}$.

In the following, we use the route discovery algorithm from [8] for planar geometric networks. We modify it so that it works on an Eulerian planar geometric network. For this, we only need to extend the face traversal routine as follows:

> Whenever the face traversal routine wants to traverse an edge $e = (u, v)$ in the reverse direction, i.e., from v to u, we traverse instead the following edges in the order $\mathbf{succ}(e), \mathbf{succ}(e)^2, \ldots, \mathbf{succ}(e)^k$ until $\mathbf{succ}(e)^{k+1} = (u, v)$. After traversing $\mathbf{succ}(e)^k$, the routine resumes the original traversal of the face from u. In our notation, $\mathbf{succ}(e)^k = \mathbf{succ}(\mathbf{succ}(e)^{k-1})$.

This modification obviously guarantees (in Eulerian geometric networks) that

all edges of the face can be visited. In the sequel we present an algorithm that is applicable to any pair s (source) and t (target) vertices. The algorithm can be applied to any vertex s over a given Eulerian network $\vec{G} = (V, E)$. Vertex s must know the target vertex t and every other vertex requires knowledge only of its immediate outgoing neighbors; however no vertex is required to know the entire graph $\vec{G} = (V, E)$.

Algorithm 1 **EULERIAN GEOMETRIC NETWORK ROUTE.**

Starting vertex: s

Destination vertex: t

1: $v \leftarrow s$ {Current vertex = starting vertex.}
2: **repeat**
3: Let f be a face of G with v on its boundary that intersects the line segment v-t.
4: **for all** edges e of f **do**
5: **if** $e \cap v\text{-}t = p$ and $\mathbf{dist}(p, t) < \mathbf{dist}(v, t)$ **then**
6: $v \leftarrow p$
7: **end if**
8: **end for**
9: Traverse f until reaching the edge xy containing the point p.
10: **until** $v = t$

Theorem 1. *Let $\vec{G} = (V, E)$ be a connected Eulerian geometric network. Let s and t be two vertices of $\vec{G}$. Algorithm 1 will find a (directed) path from s to t in at*

most $O(n^2)$ steps, where $n = |V|$.

Proof. Follows from the proof of the correctness of the traversal algorithm for planar geometric networks [8] and from the discussion above. The bound on the number of steps follows from the $O(n)$ bound on the number of steps of the algorithm from [8] and the fact that every edge on the route could be oriented oppositely to the direction of traversal and may need up to $O(n)$ steps to route around. □

Notice that in this algorithm, at any given time, the packet that is being delivered to the destination resides in exactly one vertex of the network.

Examples which show that the bound cannot be improved in general are easy to construct. They typically include a large face that needs to be traversed whose boundary contains $\Theta(n)$ edges which are all oriented the opposite way to the direction of the face traversal.

4 Route Discovery in Strongly Connected Outerplanar Geometric Networks

Recall that a planar geometric network G is *outerplanar* if one of the elements in F, the outerface, contains all the vertices. We will assume that this face is a convex polygon in $\mathbb{E}^2$. For a given triple of vertices x, y, and z, let $V_{\curvearrowleft}(x, y, z)$ [$V_{\curvearrowright}(x, y, z)$, resp.] denote the ordered set of vertices distinct from x and z that are encountered while moving from y counterclockwise [clockwise, resp.] around

the outerface of G until x [z, resp.] is reached. The ordering follows the order of encounters; see Figure 3.

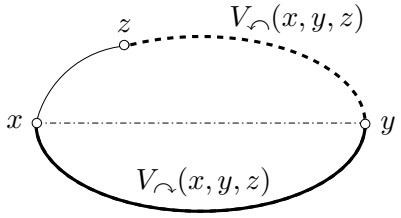

Figure 3: The dashed part of the outer face represents the vertices in $V_{\curvearrowleft}(x,y,z)$ and the bold solid part represents vertices in $V_{\curvearrowright}(x,y,z)$, respectively. Note that y belongs to both these sets and is in fact the first element of those sets.

Consider an orientation $\vec{}$ of a geometric network G. Let $N_{\curvearrowleft}(x,y,z) = V_{\curvearrowleft}(x,y,z) \cap N^+(x)$ and let $N_{\curvearrowright}(x,y,z) = V_{\curvearrowright}(x,y,z) \cap N^+(x)$. If $N_{\curvearrowleft}(x,y,z) \neq \emptyset$, let $v_{\curvearrowleft}(x,y,z)$ denote the first vertex in $N_{\curvearrowleft}(x,y,z)$. Similarly we define $v_{\curvearrowright}(x,y,z)$ as the first vertex in $N_{\curvearrowright}(x,y,z)$, if it exists.

In the sequel we present an algorithm for discovering routes. The algorithm can be applied to any vertex s over a given strongly connected outerplanar network. Vertex s must know the target vertex t and every other vertex requires knowledge only of its immediate outgoing neighbors; however no vertex is required to know the entire graph $\vec{G} = (V, E)$.

Algorithm 2 OUTERPLANAR GEOMETRIC NETWORK ROUTE.

Starting vertex: s

Destination vertex: t

1: $v \leftarrow s$ {Current vertex = starting vertex.}

2: $v_{\curvearrowleft}, v_{\curvearrowright} \leftarrow s$ {counterclockwise and clockwise bound = starting vertex.}

3: **while** $v \neq t$ **do**

4: **if** $(v, t) \in E$ **then**

5: $v, v_{\curvearrowleft}, v_{\curvearrowright} \leftarrow t$ {Move to t.}

6: **else if** $N_{\curvearrowleft}(v, t, v_{\curvearrowleft}) \neq \emptyset$ and $N_{\curvearrowright}(v, t, v_{\curvearrowright}) = \emptyset$ **then** {No-choice vertex; greedily move to the only possible counterclockwise direction toward t.}

7: $v, v_{\curvearrowleft} \leftarrow v_{\curvearrowleft}(v, t, v_{\curvearrowleft})$

8: **else if** $N_{\curvearrowleft}(v, t, v_{\curvearrowleft}) = \emptyset$ and $N_{\curvearrowright}(v, t, v_{\curvearrowright}) \neq \emptyset$ **then** {No-choice vertex; greedily move to the only possible clockwise direction toward t.}

9: $v, v_{\curvearrowright} \leftarrow v_{\curvearrowright}(v, t, v_{\curvearrowright})$

10: **else if** $N_{\curvearrowleft}(v, t, v_{\curvearrowleft}) \neq \emptyset$ and $N_{\curvearrowright}(v, t, v_{\curvearrowright}) \neq \emptyset$ **then** {Decision vertex; first take the "counterclockwise" branch but remember the vertex for the backtrack purpose.}

11: $b \leftarrow v$; $v, v_{\curvearrowleft} \leftarrow v_{\curvearrowleft}(v, t, v_{\curvearrowleft})$

12: **else if** $N_{\curvearrowleft}(v, t, v_{\curvearrowleft}) = \emptyset$ and $N_{\curvearrowright}(v, t, v_{\curvearrowright}) = \emptyset$ **then** {Dead-end vertex; backtrack to the last vertex where a decision has been made. No updates to $v_{\curvearrowleft}$ and $v_{\curvearrowright}$ are necessary.}

13: **if** $v \in V_{\curvearrowleft}(t, b, t)$ **then**

14: **while** $v \neq b$ **do**

15: $v \leftarrow v_{\curvearrowleft}(v, b, v)$

16: **end while**

17: **end if**

18: **if** $v \in V_{\curvearrowright}(t, b, t)$ **then**

19: **while** $v \neq b$ **do**

20: $v \leftarrow v_{\curvearrowright}(v, b, v)$

21: **end while**

22: **end if**

23: $v, v_{\curvearrowright} \leftarrow v_{\curvearrowright}(v, t, v_{\curvearrowright})$ {Take the "clockwise" branch toward t.}

24: **end if**

25: **end while**

Note 1. The implementation of tests in lines 6, 8, 10, 12 is simple. Since the vertices of G form a convex polygon, one can easily (and locally—remembering only two best candidates, one for each direction) compute the first vertex in $N^+(v)$ that is in a clockwise (resp. counterclockwise) direction from t or determine that such vertex does not exist. Similarly, for tests in lines 13 and 18 only local information and constant memory are needed.

Lemma 1. *Suppose Algorithm 2 reaches a decision vertex b (line 11). Next suppose that $v_1, v_2, \ldots, v_k$ are vertices reached in subsequent steps and that all are no-choice vertices, i.e., determined at line 7 or 9. Finally suppose that the next vertex reached is a dead-end vertex v_{k+1} (determined at line 12). Then vertices $v_1, v_2, \ldots, v_{k+1}$ are all either in $V_{\curvearrowleft}(t, b, t)$ or in $V_{\curvearrowright}(t, b, t)$.*

Proof. By way of contradiction, suppose i is maximum so that v_i and v_{i+1} are not both in $V_{\curvearrowleft}(t, b, t)$ or in $V_{\curvearrowright}(t, b, t)$. We may suppose $v_i \in V_{\curvearrowright}(t, b, t)$ and $v_{i+1} \in V_{\curvearrowleft}(t, b, t)$ (the other case is analogous); see Figure 4.

Since $\vec{G}$ is strongly connected, there must exist a path from b to t. Every such path must pass either through v_i or v_{i+1}. Since v_i is a no-choice ver-

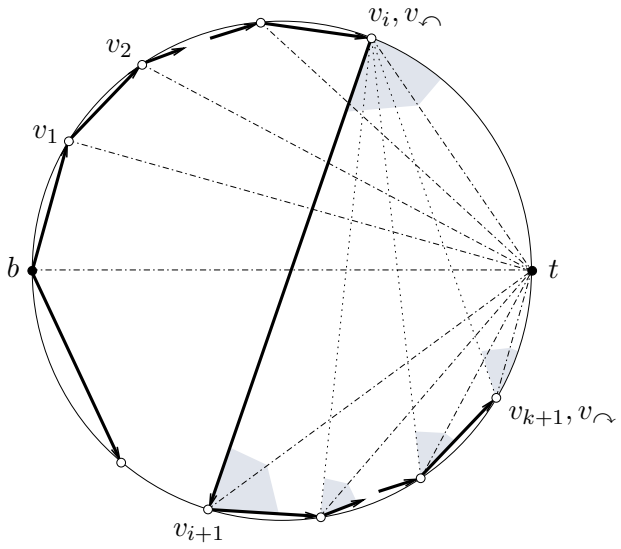

Figure 4: Dashed parts at some vertices represent the area with no outgoing edges from the corresponding vertex in that direction. The exception are the three dotted lines which represent possible edges going into v_i. However these edges cannot help to reach t. Note that $V_{\curvearrowleft}(b,t,b)\setminus\{t\} = V_{\curvearrowright}(t,b,t)\setminus\{b\}$ and $V_{\curvearrowright}(b,t,b)\setminus\{t\} = V_{\curvearrowleft}(t,b,t)\setminus\{b\}$.

tex and since $v_{i+1} \in V_{\curvearrowleft}(t,b,t)$, such a path must always pass through v_{i+1}. By the choice of i, all vertices $v_{i+1}, v_{i+2}, \ldots, v_{k+1}$ are in $V_{\curvearrowleft}(t,b,t)$ and thus such a path must eventually pass through the vertex v_{k+1} and continue to a vertex in $N_{\curvearrowleft}(v_{k+1},t,v_i) \cup N_{\curvearrowright}(v_{k+1},t,v_{k+1})$. However, $N_{\curvearrowleft}(v_{k+1},t,v_i) = \emptyset$ and $N_{\curvearrowright}(v_{k+1},t,v_{k+1}) = \emptyset$, a contradiction. □

Lemma 2. *Suppose Algorithm 2 reaches a dead-end vertex d (line 12). Then it will eventually return to the vertex b (last decision vertex defined in line 11).*

Proof. Suppose $v_1, v_2, \ldots, v_k$ are all vertices reached (in this order) after reaching the decision vertex b and before reaching the dead-end vertex $d = v_{k+1}$. By

Lemma 1, we may assume $v_1, v_2, \ldots, v_{k+1} \in V_{\curvearrowright}(t, b, t)$. The other case is analogous. Since $\vec{G}$ is strongly connected, there must exist a (directed) path from d to b. Suppose by way of contradiction that the algorithm backtracks to some vertex $x \neq b$ for which $N_{\curvearrowright}(x, b, x) = \emptyset$. Suppose furthermore that x lies between v_i and v_{i+1} going from b to t around the outer face; see Figure 5.

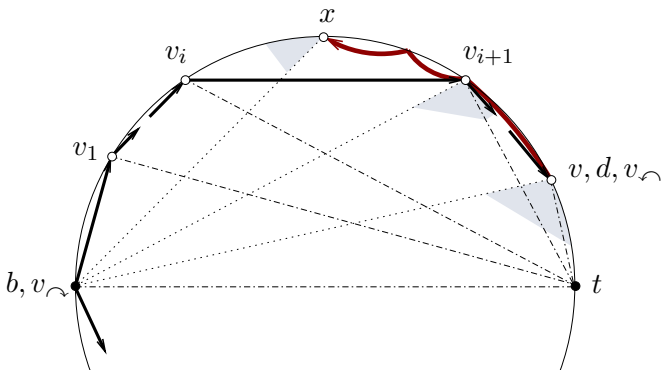

Figure 5: Dashed parts at some vertices represent the area with no outgoing edges from the corresponding vertex in that direction. The bold curve from d to x represents the backtrack path.

By our assumption none of the edges $(v_{i+1}, b), (v_{i+2}, b), \ldots, (v_{k+1}, b)$ exist, for otherwise the backtrack procedure would follow such an edge directly to b. Hence every path from d to b must eventually pass x and continue to a vertex in $N_{\curvearrowright}(x, b, x)$. However $N_{\curvearrowright}(x, b, x) = \emptyset$ by our assumption, a contradiction. □

Lemma 3. *If the "counterclockwise" branch taken at a decision vertex leads to a dead-end vertex, then no dead-end vertex is reached on the "clockwise" branch at that vertex before reaching a new decision vertex.*

Proof. The proof is similar to the two previous proofs. If $b, s_1, s_2, \ldots, s_k$ where s_k is a dead-end vertex is the counterclockwise branch at b and $b, n_1, n_2, \ldots, n_l$

where n_l is a dead-end vertex is the clockwise branch at b, then by Lemma 1, vertices $s_1, s_2, \ldots, s_k \in V_{\curvearrowleft}(b,t,b)$ and vertices $n_1, n_2, \ldots, n_l \in V_{\curvearrowright}(b,t,b)$. Now it is clear that every (directed) path from b to t must pass either through s_k or n_l and then continue to a vertex either in $N_{\curvearrowleft}(s_k,t,b) \cup N_{\curvearrowright}(s_k,t,b)$ or in $N_{\curvearrowleft}(n_l,t,b) \cup N_{\curvearrowright}(n_l,t,b)$. However all these sets are empty by our assumption, a contradiction. □

Lemma 4. *At each step of Algorithm 2 which is not the backtracking step, either $v_{\curvearrowleft}$ or $v_{\curvearrowright}$ is moved closer to t (measured as the graph distance on the outer face of G.*

Proof. This follows directly from the definition of $N_{\curvearrowleft}(v,t,v_{\curvearrowleft})$ and $N_{\curvearrowright}(v,t,v_{\curvearrowright})$ and the updates performed at lines 5, 7, 9, 11, and 23, respectively. □

Theorem 2. *Let $\vec{G} = (V,E,F)$ be a strongly connected outerplanar geometric network. Let s and t be two vertices of $\vec{G}$. Algorithm 2 will find a (directed) path from s to t in at most $2n-1$ steps, where $n = |V|$.*

Proof. The proof that the algorithm will reach the destination vertex t follows from the above lemmas. The fact that no more than $2n-1$ steps are needed follows from the fact that G has at most $2n-1$ edges, that the algorithm processes an edge at each step, and the fact that no edge is processed twice. □

5 Route Discovery in Strongly Face Connected Geometric Networks

Consider a geometric planar network $\vec{G}_c = (V_c, E_c, F_c)$ that is convex embedded into $\mathbb{E}^2$. Let $\tilde{F}_c$ be the set of all faces of G_c excluding the outerface. A *strongly face connected* network $\vec{G} = (V, E, F)$ is obtained from $\vec{G}_c$ by adding new directed edges into faces in $\tilde{F}_c$ so that every face in $\tilde{F}_c$ together with the added edges induces a strongly connected outerplanar graph in $\vec{G}$. Thus, we have $V = V_c$ and $E_c \subseteq E \subseteq E_c \cup \cup_{f \in \tilde{F}_c}\{(u, v) \ : \ u \text{ and } v \text{ are not consecutive on } f\}$. The network G_c is called the *backbone* of $\vec{G}$. An example of a strongly face connected geometric network is depicted in Figure 6. It is not hard to observe that if the orientation on a face in $\tilde{F}_c$ induces two (directed) paths of length at least two, then the face can always be extended by new edges to a strongly connected outerplanar graph.

In this section, we generalize results from the previous section and develop a routing algorithm for strongly face connected networks with a convex backbone. The algorithm does not need to know the backbone; the backbone is needed only for the proof of correctness of the algorithm. The basic idea of the algorithm is to simulate the face routing algorithm on G. A problem that arises in the face routing is that when traversing a face, it contains edge (u, v) of $\vec{G}$ that is directed opposite to the direction of the face traversal, i.e., the message should be delivered from v to u. This situation will be resolved by Algorithm 3, that will be made a part of the face routing algorithm, and which finds a route from v to u. After delivering

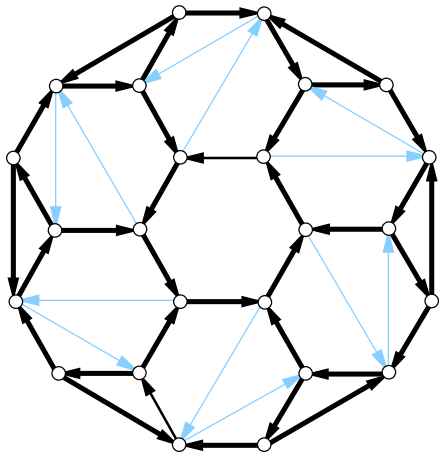

Figure 6: A strongly face connected geometric network. The darker edges represent its backbone.

the message to u, the original face routing resumes.

Let $cone(u, v, w)$ denote the cone with apex v, the supporting rays passing through u and w, respectively, and the positive interior angle $\angle uvw$. We assume that the bounding rays passing through u and w both belong to $cone(u, v, w)$. For technical reasons, we also define

$$cone(u, v, v) = cone(u, v, u') \text{ and } cone(v, v, u) = cone(u', v, u),$$

where u' is the reflection of u through the origin v.

Given vertices x and y, we say that a vertex z is $\overset{\curvearrowleft}{xy}$*-feasible* if

1. its first out-neighbor p counterclockwise starting from the line zx is in $cone(x, y, z)$, and

2. every its in-neighbor w in $cone(x, z, p)$ has an in-neighbor in $cone(w, x, y)$.

See Figure 7.

Similarly, z is $\overset{\frown}{xy}$-*feasible* if its first out-neighbor p clockwise starting from the line zx is in $cone(z, y, x)$, and every its in-neighbor w in $cone(x, z, p)$ has an in-neighbor in $cone(y, x, w)$.

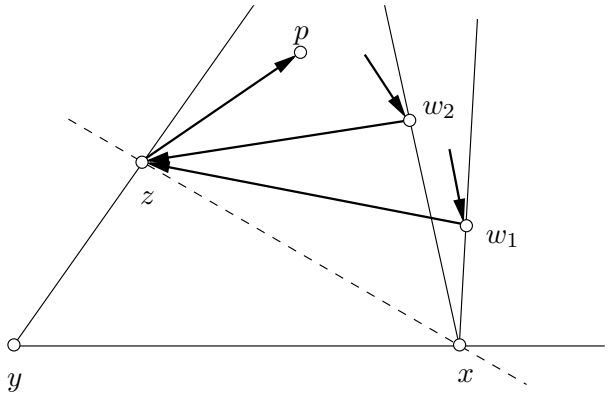

Figure 7: The vertex z is $\overset{\frown}{xy}$-feasible. Its first out-neighbor in $cone(x, y, z)$ is p, and every its in-neighbor in $cone(x, z, p)$ (vertices w_1 and w_2) has an in-neighbor in $cone(w_1, x, y)$ and $cone(w_2, x, y)$, respectively.

Algorithm 3 BYPASSING AN EDGE.

Input: An edge (u, v) in a strongly face connected geometric network $\vec{G}$

1: $w \leftarrow v$ {Current vertex = v.}

2: **while** w is $\overset{\frown}{uv}$-feasible and $(w, u) \notin E$ **do** {Counterclockwise approach.}

3: $w \leftarrow$ the first out-neighbor of w counterclockwise starting from (w, u)

4: **end while**

5: **if** $(w, u) \notin E$ **then** {Counterclockwise approach not possible.}

6: **while** $w \neq v$ **do** {Need to backtrack to the vertex v and perform clockwise approach.}

7: $w \leftarrow$ the first out-neighbor of w clockwise starting from (w, v)

8: **end while**

9: **while** w is $\overset{\frown}{uv}$-feasible and $(w, u) \notin E$ **do** {Clockwise approach.}

10: $w \leftarrow$ the first out-neighbor of w clockwise starting from (w, u)

11: **end while**

12: **end if**

13: $w \leftarrow u$ {Final move to the vertex u.}

Theorem 3. *Let $\vec{G} = (V, E, F)$ be a strongly face connected geometric network. Let (u, v) be a given (directed) edge of $\vec{G}$. On input (u, v), Algorithm 3 will find a (directed) path from v to u in at most $\ell^2 + \ell$ steps where ℓ is the length of a longest face in the backbone of $\vec{G}$.*

Proof. Let $\vec{G}_c = (V_c, E_c, F_c)$ be the backbone of $\vec{G}$. Suppose the edge (u, v) is a chord of a face $f \in F_c$. The case when (u, v) is an edge on the boundary of two faces in F_c is simpler and is also considered separately in the course of the proof below.

The *$\curvearrowleft$-polygonal* $v - u$ path is the shortest (directed) $v - u$ path all of whose edges are in $cone(u, v, v)$ and together with the edge (u, v) they constitute a convex polygon. We define the *$\curvearrowright$-polygonal* $v - u$ path as the shortest (directed) $v - u$ path all of whose edges are in $cone(v, v, u)$ and together with the edge (u, v) they constitute a convex polygon. Since $\vec{G}$ is strongly face connected, at least one

of the $\curvearrowleft$-polygonal or $\curvearrowright$-polygonal $v - u$ paths exists and is completely contained in f. We prove that either Algorithm 3 finds a $v - u$ path that is completely in $cone(u, v, v)$ and this path will be the $\curvearrowleft$-polygonal $v - u$ path if this polygonal path exists, or it finds the $\curvearrowright$-polygonal $v - u$ path.

First suppose that there exists the $\curvearrowleft$-polygonal $v - u$ path P. We claim that Algorithm 3 finds it. (Note that since $\vec{G}$ is strongly face connected, this case always holds whenever (u, v) is on the boundary of two faces in $\tilde{F}_c$. If (u, v) is on the boundary of the outer face, then there may not be a $\curvearrowleft$-polygonal $v - u$ path but then there must be the $\curvearrowright$-polygonal $v - u$ path. If this is the case, the convexity of the outer face guarantees that $cone(u, v, v)$ does not contain any out-neighbor of v, hence v is not $\overset{\curvearrowleft}{uv}$-feasible and the algorithm performs lines 9, 10, and 11. The proof that this gives the $\curvearrowright$-polygonal $v - u$ path is essentially the same as the following proof for the $\curvearrowleft$-polygonal path. Now we can return to the original case.) Since P is a shortest path and $P + (u, v)$ is convex, for any three consecutive vertices w^-, w, w^+ on P, the vertex w^+ is the first out-neighbor of w counterclockwise starting from the line ww^-. Moreover, any in-neighbor x of w in $cone(w^-, w, w^+)$ is on P and hence has an in-neighbor in $cone(x, u, v)$. Thus, we may conclude that if w^- was $\overset{\curvearrowleft}{uv}$-feasible, then w is $\overset{\curvearrowleft}{uv}$-feasible. Our claim now follows from lines 2, 3, 4, and 13 in Algorithm 3 and by induction since the base case, i.e., v is $\overset{\curvearrowleft}{uv}$-feasible, is true.

Second suppose that there is no $\curvearrowleft$-polygonal $v - u$ path. The convexity of f guarantees that there is no $v - u$ path in $cone(u, v, v)$ which is completely contained in f. If in lines 2, 3, and 4, Algorithm 3 constructs a path from v to an

in-neighbor of u, then in line 13 the path is completed to a $v - u$ path and we are done. We therefore may suppose that the **while** loop in line 2 terminates with the current vertex w not $\overset{\curvearrowleft}{uv}$-feasible. We first claim that w is on the face f.

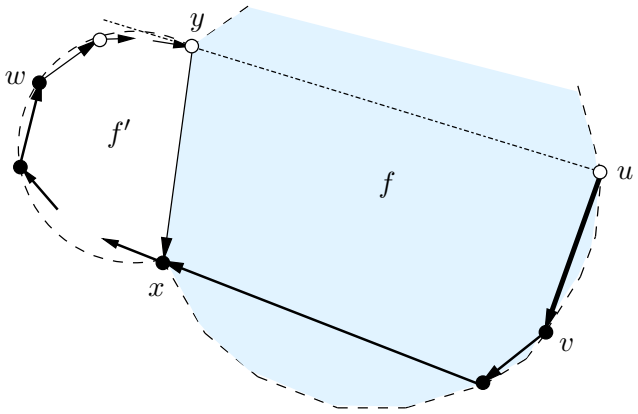

Figure 8: The case when the counterclockwise route fails at the vertex w which is not $\overset{\curvearrowleft}{uv}$-feasible. The dashed region represents the face f. Filled vertices represent those traversed by the algorithm so far.

Suppose this is not the case; see Figure 8. Then w is on a face $f' \neq f$. Let x be the last vertex on the face f encountered by the algorithm before reaching w. Let y be the neighbor of x on the boundary of f and in $cone(u, v, x)$. It follows from our assumption that y is an in-neighbor of x. Since the face g on the other side of the edge xy is strongly connected, there is the $\curvearrowleft$-polygonal $x - y$ path Q in g. Because $Q + (y, x)$ is convex, all vertices encountered by the algorithm between x and w inclusive are on Q. Hence, $g = f'$.

Since x is $\overset{\curvearrowleft}{uv}$-feasible, it follows that y has an in-neighbor in $cone(y, u, v)$. Hence also the polygon $Q + u$ is convex. However this is a contradiction to the fact that w is not $\overset{\curvearrowleft}{uv}$-feasible. Indeed, any in-neighbor of w in the polygon $Q + u$

must be a vertex z on Q that has not been traversed by the algorithm so far, but the convexity of $Q + u$ guarantees that z has an in-neighbor in $cone(z, u, v)$ (which is a vertex on Q). This proves the claim.

Thus we suppose w is the first vertex encountered by the algorithm that is not $\overset{\curvearrowleft}{uv}$-feasible. We proved w is on the face f; see Figure 9.

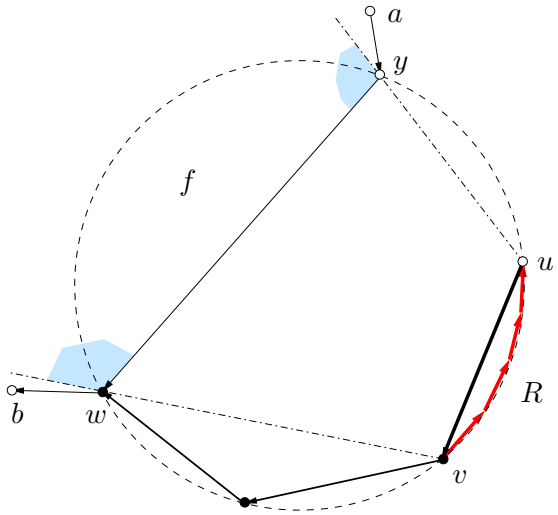

Figure 9: The case when the counterclockwise route fails at the vertex w which is not $\overset{\curvearrowleft}{uv}$-feasible. The dashed region at vertex y represents the region without the in-neighbors of y. The dashed region at vertex w represents the region without the out-neighbor of w.

We may suppose that (w, u) is not edge of $\vec{G}$, since otherwise line 13 guarantees a $v-u$ path. In what follows, we prove that lines 6, 7, and 8 guarantee that the algorithm backtracks to the vertex v. From the definition of feasibility, it follows that either there is an in-neighbor y of w in $cone(u, v, w)$ that does not have any in-neighbors in $cone(y, u, v)$ or w itself has no out-neighbors in $cone(u, v, w)$. Either case implies that there is no (directed) $v - u$ path whose internal vertices are

in $cone(u, v, v)$ and completely contained in the face f. Since f in $\vec{G}$ is strongly connected, there must exists a $w - u$ path in f. Such a path cannot go through y, and hence it must go through v. The proof that lines 6, 7, and 8 backtrack to v is essentially the same as the proof of Lemma 2.

Now from all our assumptions, it follows that there is the $\curvearrowright$-polygonal $v - u$ path R. Moreover, R must be contained in f. Using the same proof as in the first case but replacing lines 2, 3, 4 with 9, 10, 11 we conclude that the algorithm finds the $\curvearrowright$-polygonal $v - u$ path R.

Finally, it remains to show that the length of the $v - u$ walk computed by the algorithm is at most $\ell^2+\ell$. The algorithm, at worst, will visit at most ℓ neighboring faces of f and it will traverse at most ℓ edges in each. Since the backtracking is performed completely in f, the algorithm will go through another at most ℓ edges leading to u. The proof is complete. □

The following theorem follows from the previous discussion. The bound on the number of steps follows from $O(n)$ bound for the face routing from [8] and from Theorem 3.

Theorem 4. *Let $\vec{G} = (V, E, F)$ be a strongly face connected geometric network. Let s and t be two vertices of $\vec{G}$. The face routing algorithm extended by Algorithm 3 will find a (directed) path from s to t in at most $n(\ell^2 + \ell)$ steps, where $n = |V|$ and ℓ is the length of a longest face in a backbone of $\vec{G}$.*

6 Conclusion

Routing in position-based oriented ad hoc networks is much more difficult than routing in non-oriented networks. Except for flooding, there seems to be no simple extension of the known routing algorithms that would be applicable to general oriented networks. In this paper we give routing algorithms for three cases of oriented planar networks: Eulerian, Outerplanar and a generalization of Outerplanar networks which we called Strongly Face Connected. No doubt, the problem of efficient route discovery in oriented ad hoc networks is far from being settled and further progress in this area is needed.

References

[1] D. Avis and K. Fukuda, A pivoting algorithm for convex hulls and vertex enumeration of arrangements and polyhedra, Proc. of 7th Ann ACM Symp on Comput Geom, 1991, pp. 98–104.

[2] L. Barrière, P. Fraigniaud, L. Narayanan, and J. Opatrny, Robust position-based routing in wireless ad hoc networks with irregular transmission ranges, Wireless Communications and Mobile Computing 3 (2003), 141–153.

[3] J. A. Bondy and U. S. R. Murty, Graph theory with applications, American Elsevier Publishing, New York, 1976.

[4] P. Bose and P. Morin, An improved algorithm for subdivision traversal without extra storage, Internat J Comp Geom Appl 12 (2002), 297–308.

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

[6] E. Chavez, S. Dobrev, E. Kranakis, J. Opatrny, L. Stacho, and J. Urrutia, Traversal of a quasi-planar subdivision without using mark bits, Journal of Interconnection Networks 5 (2004), 395–408.

[7] C. Gold, T. Charters, and J. Ramsden, Automated contour mapping using triangular element data structures and an interpolant over each irregular triangular domain, Computer Graphics 11 (1977), 170–175.

[8] E. Kranakis, H. Singh, and J. Urrutia, Compass routing on geometric networks, Proc. of 11th Canadian Conference on Computational Geometry, 1999, pp. 51–54.

[9] F. Kuhn, R. Wattenhofer, Y. Zhang, and A. Zollinger, Geometric ad-hoc routing: Of theory and practice, *Proc 22nd ACM Symposium on the Principles of Distributed Computing*, 2003, pp. 63–72.

[10] F. Kuhn, R. Wattenhofer, and A. Zollinger, Worst-case optimal and average-case efficient geometric ad-hoc routing, Proc 4th ACM International Symposium on Mobile Ad Hoc Networking and Computing, 2003, pp. 267–278.

[11] X.-Y. Li, G. Calinescu, and P.-J. Wan, Distributed construction of planar spanner and routing for ad hoc wireless networks, Proc 25th Conference on Computer Communications, 2002, pp. 23–27.

[12] X.-Y. Li, G. Calinescu, P.-J. Wan, and Y. Wang, Localized Delaunay triangulation with application in ad hoc wireless networks, IEEE Transactions on Parallel and Distributed Systems 14 (2003), 1035–1047.

[13] D. Peuquet and D. Marble, "ARC/INFO: an example of a contemporary geographic information system", Introductory Readings in Geographic Information Systems, D. Peuquet and D. Marble (Editors), Taylor & Francis, 1990, pp. 90–99.

[14] U.S. Bureau of the Census, "Technical description of the DIME system", Introductory Readings in Geographic Information Systems, D. Peuquet and D. Marble (Editors), Taylor & Francis, 1990, pp. 100–111.