Available online at www.sciencedirect.com
ScienceDirect
Discrete Applied Mathematics 156 (2008) 3430–3442
DISCRETE APPLIED MATHEMATICS
www.elsevier.com/locate/dam

Constant memory routing in quasi-planar and quasi-polyhedral graphs[☆]

Evangelos Kranakis[a], Tim Mott[b], Ladislav Stacho[b,*]

[a] School of Computer Science, Carleton University, 1125 Colonel By Drive, Ottawa, Ontario, Canada K1S 5B6
[b] Department of Mathematics, Simon Fraser University, 8888 University Drive, Burnaby, British Columbia, Canada V5A 1S6

Received 13 October 2006; received in revised form 20 January 2008; accepted 24 January 2008
Available online 14 March 2008

Abstract

We address the problem of online route discovery for a class of graphs that can be embedded either in two or in three-dimensional space. In two dimensions we propose the class of *quasi-planar* graphs and in three dimensions the class of *quasi-polyhedral* graphs. In the former case such graphs are geometrically embedded in $\mathbb{R}^2$ and have an underlying backbone that is planar with convex faces; however within each face arbitrary edges (with arbitrary crossings) are allowed. In the latter case, these graphs are geometrically embedded in $\mathbb{R}^3$ and consist of a backbone of convex polyhedra and arbitrary edges within each polyhedron. In both cases we provide a routing algorithm that guarantees delivery. Our algorithms need only "remember" the source and destination nodes and one (respectively, two) reference nodes used to store information about the underlying face (respectively, polyhedron) currently being traversed. The existence of the backbone is used only in proofs of correctness of the routing algorithm; the particular choice is irrelevant and does not affect the behaviour of the algorithm.
Crown Copyright © 2008 Published by Elsevier B.V. All rights reserved.

Keywords: Ad hoc network; Routing; Planar graph; Local algorithm

1. Introduction

Ad hoc networks are widely being adopted today in many sectors of the economy in order to enhance communication and sensor capabilities. A particular case in point is that of sensor networks which are employed in many sectors (such as transportation, agriculture, personal and institutional security, radiology, medicine, and manufacturing) that benefit greatly from increased surveillance. Given that the nodes of such a network are expected to spontaneously create an impromptu connected system that dynamically adapts to device failure and degradation, manages movement of nodes, and may even react to changes in task and network requirements, it is not surprising that a predefined topological structure is not feasible.

Formally, the ad hoc network is represented as a unit disk graph (whereby two nodes are adjacent if and only if they are within distance one). Since it is usually difficult to attain the required communication efficiency with such a

☆ This is an expanded version of a paper that was presented at the 2nd IEEE PerCom Workshop on Pervasive Wireless Networking (PWN 2006).
* Corresponding author. Tel.: +1 604 2914816; fax: +1 604 291 4947.
E-mail address: lstacho@sfu.ca (L. Stacho).

doi:10.1016/j.dam.2008.01.027

network representation, it is useful to work with a simplified topological structure within the ad hoc network. Such a structure must not only span the entire network but also maintain a sufficient number of the old links in order to sustain connectivity. A model typically adapted for this purpose is a planar (*i.e.*, without crossings) spanner of the ad hoc network. To be useful for our purposes it must be possible to construct the spanner from the original network locally and in a distributed manner.

The most efficient way to accomplish communication exchange efficiently between a given pair of nodes of an ad hoc network is to discover a route (*i.e.*, a path) between them. Path finding, or *routing*, is a fundamental problem in the field of ad hoc communication networks. The inherent mobility of the nodes of an ad hoc network and the lack of a predesigned topology imply that packets must navigate the network using only local information and constant memory. Moreover, it is vital that route discovery strategies use only local information and are easily adaptable to network changes. This means that at a vertex v, a routing algorithm must base its next move on v, its neighbourhood $N(v)$, and a small number of bits (typically $O(\log n)$) of stored information. Such an algorithm is said to be *local*, or *online*.

An important technique for discovering routes between two nodes in an ad hoc network involves application of a *face routing* algorithm on a planar spanner of the wireless network [11,5]. 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 [5,11,2,8,13,14,3]. In such algorithms the emphasis is not on minimising the number of hops but rather on guaranteeing packet delivery. Recent research has concentrated on extending these ideas from planar networks to more complex networks. In particular, [6] addresses the problem in directed planar networks, while [12] provides a general survey. We also note that related to routing is traversal which is addressed in several papers by Avis and Fukuda [1], Bose and Morin [4], Chavez et al. [7], Czyczowicz et al. [9], Gold et al. [10], Peuquet and Marble [15,16]. However, traversal is less efficient than routing for message delivery.

1.1. *Results and contribution of the paper*

In this paper we represent a network as a *geometric graph*, that is, a graph G with vertices V in $\mathbb{R}^2$ or $\mathbb{R}^3$, where each vertex is aware of its coordinates. Edges in G are line segments with (distinct) endpoints in V.

We address the problem of online route discovery in a class of graphs that is richer than planar. In two dimensions the class of these graphs is a subclass of *quasi-planar* graphs defined in [7]. We will continue using the same name for the subclass. Intuitively speaking, such graphs are geometrically embedded into $\mathbb{R}^2$ and have underlying planar backbones with convex faces. However, within each face, arbitrary edges are allowed. The superclass of quasi-planar graphs from [7] contains graphs with not necessarily convex backbone and arbitrary graphs within each face. In three dimensions we define a new class of graphs, *quasi-polyhedral* graphs, which extends the notion of quasi-planar graphs into $\mathbb{R}^3$. The backbones of these graphs are collections of convex polyhedra, and arbitrary edges are allowed within each polyhedron. It is important to note that for the purposes of our algorithms only the existence of a backbone is essential. In contrast to the techniques described in the introduction, our algorithms do not explicitly know or determine which edges belong to the backbone; its existence is used only in proofs of correctness of the algorithms.

We will extend the well-known right-hand rule routing algorithm [11,5] for planar graphs to quasi-planar. Furthermore we extend our techniques to a routing algorithm for quasi-polyhedral graphs. Our algorithm for quasi-planar graphs needs only remember the source and destination vertices and one reference vertex used to store information about the underlying face currently being traversed. Our algorithm for quasi-polyhedral graphs requires enough memory to store the source and destination vertices, and two reference vertices. If not all polyhedra have triangular faces, the algorithm also requires memory to store the normal to a given plane.

In addition to using very little memory, our quasi-planar routing algorithm is also robust: at each node, it constructs a set of candidates for its next local destination, and can use any rule or heuristic to choose from this set. This provides more flexibility than, for example, the standard Greedy algorithm, which has only one option from any node.

2. Quasi-planar routing in $\mathbb{R}^2$

Let $G = (V, E, F)$ be a planar graph with vertex set V, edge set E, and face set F. A *convex embedding* of G is a straight-line embedding into the plane such that the boundary of every face is a convex polygon; we will associate G with its convex embedding. Note that not every planar graph has a convex embedding. For the remainder of the paper we assume that such a graph G has no three collinear vertices. This assumption will be used in our algorithms.

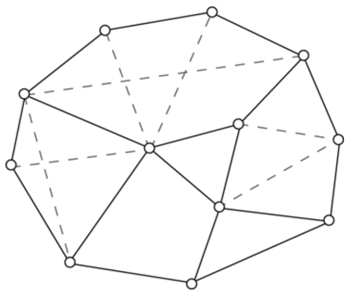

Fig. 1. A quasi-planar graph and one of its underlying planar graphs (shown with solid lines).

Let $G = (V, E, F)$ be a convex embedding, and construct a new graph Q by adding chords to the faces of G except for the outer face f_O. That is $Q = (V, E \cup E')$, where each edge $e \in E'$ joins two vertices of some face $f \in F \setminus \{f_O\}$. We call such a graph Q a *quasi-planar graph*: there may be many crossing edges, but a facial structure remains. Fig. 1 illustrates an example of a quasi-planar graph.

We refer to G as an *underlying planar graph* of Q, and say that the faces $f_i \in F$ of G are *underlying faces* of Q. Note that an underlying planar graph is not necessarily unique for a given quasi-planar graph. For the purposes of our routing algorithm it is enough to know that such a graph G exists; the particular choice of G is irrelevant and will not affect the behaviour of the algorithm. The existence of the graph G is used only in proofs of correctness of the algorithm.

For vertices u, v, and w, we denote by $\angle uvw$ the counterclockwise angle from u to w about v. Similarly, *cone*(u, v, w) denotes the cone with apex v and supporting lines through u and w, with interior angle $\angle uvw$. For both $\angle uvw$ and *cone*(u, v, w) we require that v does not coincide with u or w.

Define $\mathbf{cw}(u, v)$ to be the first clockwise neighbour of u starting from the direction uv. Note that uv is not required to be an edge. Similarly, $\mathbf{ccw}(u, v)$ is the first counterclockwise neighbour of u starting from the direction uv. These two functions can be computed locally, as long as $uv \in E$ or the location of v is known.

The edges uv_1 and uv_2 are *radially adjacent* if $v_2 = \mathbf{cw}(u, v_1)$ or $v_2 = \mathbf{ccw}(u, v_1)$. Observe that if $uv_1, uv_2 \in E$ are radially adjacent edges then some underlying face f contains u, v_1, and v_2. Depending on the choice of the underlying planar graph G, the edges uv_i may be outer edges or chords of f, but again, this distinction is not important.

Let $u, v, w_1, w_2, \ldots, w_p \in V$. Then $w_1, w_2, \ldots, w_p$ form a *clockwise sequence* around u from v if they are the first p consecutive clockwise neighbours of u starting from the direction determined by v. Note that v is not necessarily adjacent to u. A *counterclockwise sequence* is defined analogously.

We denote by uv the line segment through vertices u and v; it will be clear from context whether uv refers to an edge or a line segment. The line segment st separates the vertex set into two subsets V_A and V_B that we can think of as containing vertices "above" and "below" st, respectively. Specifically, $V_A = \{v \in V : 0 < \angle tsv < \pi\}$ and $V_B = \{v \in V : \pi < \angle tsv < 2\pi\}$, and $V = \{s, t\} \cup V_A \cup V_B$.[1] Since G is represented by a convex embedding and using the assumption that $st \notin E$, it follows that both V_A and V_B are non-empty. If a vertex v knows the geometric locations of s and t, it is a fast local computation to determine whether $v \in V_A$ or $v \in V_B$. Finally, for any vertex v of G, $N(v)$ denotes the set of neighbours of G.

Lemma 2.1. *Let Q be a quasi-planar graph with $s, t \in V$ given, and let $v \in V_A$. If $N(v) \cap V_A = \emptyset$ then $vs, vt \in E$. Similarly, for a vertex $v \in V_B$, if $N(v) \cap V_B = \emptyset$ then $vs, vt \in E$.*

Proof. We argue by contradiction: suppose there exists a vertex $v \in V_A$ such that $N(v) \cap V_A = \emptyset$, and $vs \notin E$. Index the neighbours $u_1, u_2, \ldots, u_p$ of v such that $\angle u_1vu_2 < \angle u_1vu_3 < \cdots < \angle u_1vu_p$. By convexity of the outer underlying face, it follows that no vertex lies outside *cone*(u_1, v, u_p). Therefore, s is contained within the convex hull of $\{v, u_i, u_{i+1}\}$ for some i. But v, u_i, and u_{i+1} are all on the same underlying face, which, being convex, must have an empty interior. This shows that v must be adjacent to s; similarly, $vt \in E$.

The same argument applies to a vertex in V_B. ■

[1] The definitions of V_A and V_B depend on the choice of s, t; however, their reference will be omitted as they can easily be understood in context.

2.1. *The* QUASI-PLANAR *algorithm*

We now describe an $O(1)$-memory routing algorithm that guarantees delivery on quasi-planar graphs. We will omit reference to the choice of underlying planar graph for a given quasi-planar graph; the results hold for any such choice. The QUASI-PLANAR algorithm traverses vertices within the underlying faces intersecting st, alternately using the left- and right-hand rules (*i.e.*, using the functions **ccw** and **cw**) when $v \in V_A$ and $v \in V_B$, respectively; see Algorithm 1.

Algorithm 1 Quasi-Planar Routing

```
1:  procedure QUASI-PLANAR($Q, s, t, \mathcal{R}$)
2:      $v \leftarrow$ ccw$(s, t)$
3:      $x \leftarrow$ cw$(s, t)$
4:      while $vt \notin E$ do
5:          if $v \in V_A$ then
6:              Find the counterclockwise sequence $b_1, b_2, \ldots, b_p, a$ around $v$ from $x$, where $p \geq 0$,
    $a \in V_A$ and $b_i \in V_B$, $1 \leq i \leq p$.
7:              if $\mathcal{R}(v, x) = a$ then
8:                  $x \leftarrow b_p$                                  ▷ if $p = 0$, we let $b_p = x$
9:                  $v \leftarrow a$
10:             else                          ▷ in this case $\mathcal{R}(v, x) = b_k$ for some $k$,  $1 \leq k \leq p$
11:                 $x \leftarrow v$
12:                 $v \leftarrow b_k$
13:             end if
14:         else  $v \in V_B$
15:             Find the clockwise sequence $a_1, a_2, \ldots, a_q, b$ around $v$ from $x$, where $q \geq 0$, $b \in V_B$
    and $a_i \in V_A$, $1 \leq i \leq q$.
16:             if $\mathcal{R}(v, x) = b$ then
17:                 $x \leftarrow a_q$                                  ▷ if $q = 0$, we let $a_q = x$
18:                 $v \leftarrow b$
19:             else                          ▷ in this case $\mathcal{R}(v, x) = a_k$ for some $k$,  $1 \leq k \leq q$
20:                 $x \leftarrow v$
21:                 $v \leftarrow a_k$
22:             end if
23:         end if
24:     end while
25:     $v \leftarrow t$
26: end procedure
```

Routing from s to t is trivial when $s = t$ or $st \in E$; we therefore assume that s and t are distinct and non-adjacent, and for brevity in the following algorithm we refrain from explicitly checking for the trivial cases.

As is typical of other algorithms using the face routing technique, the QUASI-PLANAR algorithm only requires enough memory to remember s, t, and one other reference vertex x; this latter vertex is used to store information about the current underlying face. Whenever the current vertex v is in V_A, x will be in V_B, and *vice versa*.

Finally, QUASI-PLANAR requires a rule $\mathcal{R}$ that will determine the next vertex from the neighbours of the current vertex v. First suppose $v \in V_A$, and hence $x \in V_B$. Let $b_0 = x$, and let $b_1, b_2, \ldots, b_p, a$ be a counterclockwise sequence around v from x, where $p \geq 0$, $b_i \in V_B$, and $a \in V_A$. Although the set $\{b_1, b_2, \ldots, b_p\}$ may be empty (that is, $p = 0$ is possible), Lemma 2.1 guarantees the existence of a. We require that the function $\mathcal{R}(v, x)$ evaluate to an element from the (non-empty) set $\{b_1, b_2, \ldots, b_p, a\}$; see Fig. 2.

For sake of simplicity, we abuse notation and also refer to $\mathcal{R}(v, x)$ when $v \in V_B$ and $x \in V_A$. That is, $\mathcal{R}(v, x) \in \{a_1, a_2, \ldots, a_q, b\}$ where $a_1, a_2, \ldots, a_q, b$ is a clockwise sequence around v from x, $q \geq 0$, $a_i \in V_A$, and $b \in V_B$. We again let $a_0 = x$.

As we will prove shortly, the particular choice of $\mathcal{R}$ does not affect the correctness of the algorithm on quasi-planar graphs. Incidentally, observe that by choosing $\mathcal{R}(v, x) = a$ for all (v, x), the algorithm can emulate standard face routing on the underlying planar graph having the maximum number of faces. A more effective rule for most applications is naturally to choose the vertex from $\{b_1, \ldots, b_p, a\}$ closest in Euclidean distance to the destination t.

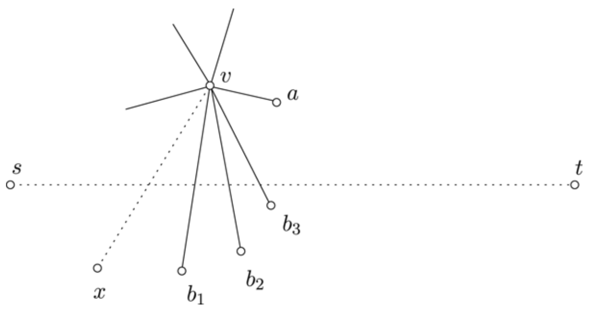

Fig. 2. The current vertex is v; candidates for the next vertex are $\{b_1, \dots, b_p, a\}$.

More complex rules are possible if the algorithm has access to other (non-geometric) information at each node, such as network load.

Theorem 2.2. *Given a quasi-planar graph Q and distinct, non-adjacent vertices $s, t \in V(Q)$, the* QUASI-PLANAR *algorithm successfully routes from s to t.*

Proof. We will show that v and x are on the same underlying face during the execution of QUASI-PLANAR. Furthermore, let l_k denote the point of intersection of vx with st after the kth iteration of the `while` loop. We will also show that if $v \neq t$ after the kth iteration, then l_k exists and $s = l_0 \prec \cdots \prec l_k \prec t$ where $\prec$ is the natural ordering along st.

The intersection points l_k are determined by pairs of distinct vertices in Q, so the sequence $l_0, l_1, \dots$ has at most $\binom{|V|}{2}$ terms. The `while` loop iterates as long as $vt \notin E$, resulting in a new intersection point with each iteration. Therefore, since this sequence of points is finite, it follows that after some iteration, $vt \in E$. The `while` loop then terminates and v reaches t at step 25.

Thus, it remains to prove the above two claims (Claims 1 and 4 in what follows). We proceed by induction on k, the number of iterations of the `while` loop in steps 4–24.

Claim 1. *Vertices v and x are on the same underlying face.*

Proof. This is certainly true after steps 2 and 3. For $k \geq 1$, first suppose $v \in V_A$. If $\mathcal{R}(v, x) = a$, then the argument is as follows. The vertex a is the first neighbour of v counterclockwise from vb_p, so after the updates $x \leftarrow b_p$ and $v \leftarrow a$, the vertices v and x will be on the same underlying face. If $\mathcal{R}(v, x) = b_k$, $1 \leq k \leq p$, then after the update, v and x will be adjacent and hence must be on the same underlying face.

If $v \in V_B$ the argument is similar. ■

Claim 2. *If $v \in V_A$, then for every $0 \leq i < p$, the vertices v, b_i, b_{i+1} are on the same underlying face. Moreover, the vertices v, b_p, a are on the same underlying face. Similarly if $v \in V_B$, then for every $0 \leq i < q$, the vertices v, a_i, a_{i+1} are on the same underlying face, and the vertices v, a_q, b are on the same underlying face.*

Proof. We only consider the case $v \in V_A$ in detail; the other case is similar. For $i \geq 1$ the statement follows since b_{i+1} is the first neighbour of v counterclockwise from vb_i. Thus, suppose $i = 0$, so we must show that v, x, and b_1 are on the same underlying face. If $vx \in E$, the argument is the same as above: vx and vb_1 are radially adjacent edges. On the other hand, if $vx \notin E$, let $u = \mathbf{cw}(v, b_1)$. Then the vertices v, b_1, u lie on the same underlying face f. Now, since x is contained in $cone(u, v, b_1)$, and from Claim 1, it follows that x also lies on f.

The same reasoning shows that v, b_p, and a are on the same underlying face. ■

Claim 3. *If $v \in V_A$, then $\angle svx < \angle svt$, and similarly if $v \in V_B$, then $\angle xvs < \angle tvs$. That is, the line segments vx and st intersect.*

Proof. First, when $k = 0$, note that from the assumptions that $st \notin E$ and no three vertices are collinear, it follows from the convexity of the underlying faces that $v \in V_A$ and $x \in V_B$ exist and are well-defined after the initialisation (steps 2–3). By choice of v and x, it is clear that $v = \mathbf{ccw}(s, x)$, so s, v, and x all lie on a common underlying face f. If $\angle svx > \angle svt$, there are two possibilities: either $\pi < \angle xsv < 2\pi$, or t is in the convex hull of s, v, and x. Because

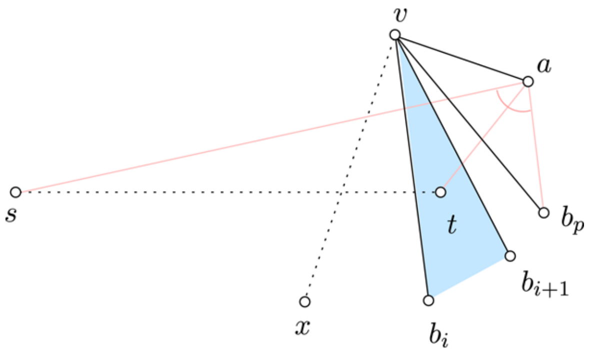

Fig. 3. Invalid position for t when $\mathcal{R}(v, x) = a$.

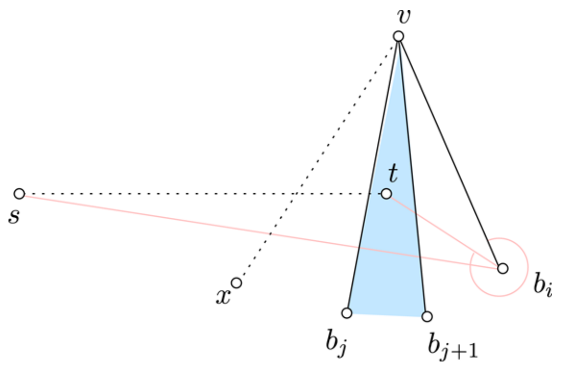

Fig. 4. Invalid position for t when $\mathcal{R}(v, x) = b_i$.

f is convex and the angle $\angle xsv$ is an interior angle, $0 < \angle xsv < \pi$, eliminating the first case. On the other hand, t cannot be in the interior of f, so t is not in the convex hull of s, v, and x. Therefore $\angle svx < \angle svt$, establishing the basis of the induction.

Now assume that after k iterations of the `while` loop, the desired property holds. By symmetry, we may without loss of generality assume that currently $v \in V_A$, and consequently $x \in V_B$.

During the $(k + 1)$th iteration, first suppose that $\mathcal{R}(v, x) = a$. Then v and x will be assigned a and b_p respectively, so we must show that $\angle sab_p < \angle sat$. Towards a contradiction, suppose that $\angle sat < \angle sab_p$. Then t lies within the convex hull of v, b_i, and b_{i+1} for some $0 \le i < p$, or within the convex hull of v, b_p, and a; see Fig. 3. But each of these triples lies on an underlying face, by Claim 2, which by convexity cannot contain t, a contradiction.

If, on the other hand, $\mathcal{R}(v, x) = b_i$ for some $i > 0$, then v and x will be assigned b_i and v, respectively, and we must show that $\angle tb_iv < \angle tb_is$. To this end, suppose that $\angle tb_is < \angle tb_iv$. Then either $\pi < \angle xvt < 2\pi$ or $0 < \angle xvt < xvb_i$. The first case contradicts the induction step, so suppose that $0 < \angle xvt < xvb_i$. Then for some $0 \le j < i$, t lies within the convex hull of the vertices v, b_j, b_{j+1}, as shown in Fig. 4. However, by Claim 2, this is impossible. ■

Claim 4. *Suppose $v \ne t$. Then $s = l_0 \prec \cdots \prec l_k \prec t$ where $\prec$ is the natural ordering along st.*

Proof. It follows from Claim 3 that l_j is well-defined (*i.e.*, the intersection of vx with st exists) and that $s \preceq l_j \prec t$ for all $0 \le j \le k$. We now assume for some $0 \le j < k$ that $v \in V_A$; the case $v \in V_B$ is similar.

Since all underlying faces are convex, the angle between any radially adjacent edges is less than π. Therefore, the point of intersection of st with vb_i precedes that of st with vb_{i+1} for all $0 \le i < p$, and the point of intersection of st with vb_p precedes that of st with b_pa. Regardless of the choice of $\mathcal{R}(v, x)$, we must then have $l_j \prec l_{j+1}$. ■

This concludes the proof of Theorem 2.2. ■

Observe that QUASI-PLANAR runs in polynomial time since there are at most $\binom{|V|}{2}$ intersections l_k. Moreover, the algorithm only uses those underlying faces crossing the line segment st.

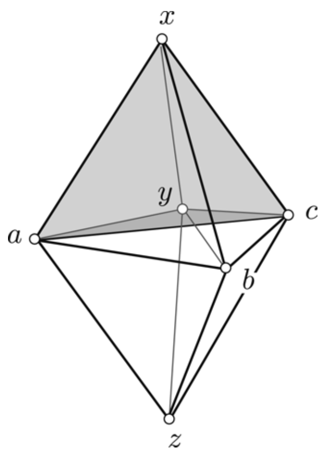

Fig. 5. The pairwise adjacent vertices a, b, c compose a 3-cycle, but not a face. There are six polyhedra in the graph: the tetrahedra with vertex sets $\{a, b, x, y\}$, $\{b, c, x, y\}$, $\{c, a, x, y\}$, $\{a, b, y, z\}$, $\{b, c, y, z\}$, and $\{c, a, y, z\}$. The polyhedron $\{c, a, x, y\}$ is shaded in the figure.

3. Quasi-polyhedral routing in $\mathbb{R}^3$

In this section we extend the notion of quasi-planar graphs to *quasi-polyhedral graphs* in $\mathbb{R}^3$, and describe a routing algorithm on these graphs. It is theoretically interesting that we can guarantee delivery in $\mathbb{R}^3$ with an online algorithm, since the geometry becomes much more subtle — compare, for example, the (trivial) problem of traversing a polygon in $\mathbb{R}^2$ to traversing a polyhedron in $\mathbb{R}^3$.

3.1. *Quasi-polyhedral graphs*

Let V be a set of vertices in $\mathbb{R}^3$, not all coplanar, and let P_O be the convex hull of V. Consider a geometric graph $G = (V, E)$. If the edges of G determine a set of convex polyhedra such that any two polyhedra are either disjoint or intersect in exactly one vertex, edge, or face, and if moreover their union is P_O, then we say G is a *polyhedral graph*. We use $\mathcal{P}$ to denote the set of these polyhedra along with P_O, and call P_O the *outer polyhedron* of G. Note that $\mathcal{P}$ is not necessarily uniquely determined by (V, E), but this is not important for our purposes.

Let F be the set of all faces determined by $\mathcal{P}$. We say $f \in F$ is a face of the polyhedron $P \in \mathcal{P}$ if $f \cap P = f$. A polyhedral graph G may now be described by the 4-tuple $(V, E, F, \mathcal{P})$.

For three distinct, not necessarily adjacent vertices $a, b, c \in V$, denote by $\triangle abc$ the triangle with vertices a, b, c. A *3-cycle abc* is a triple of pairwise adjacent vertices $a, b, c \in V$.

As in the previous section, we will assume that no three vertices are collinear. To simplify the presentation of the routing algorithm, we now also assume that no four vertices are coplanar, so that every face in F is a triangle. Note, however, that not every 3-cycle is a face; *e.g.*, see Fig. 5. It is straightforward to extend the routing procedure for a more general case where coplanar vertices are allowed if they constitute a face of a polyhedron. This is accomplished by storing the normal of the plane through three given vertices so that further vertices in that plane can be detected.

As an analogue of quasi-planar graphs, we now add chords to a polyhedral graph, so long as the chords join vertices on the same polyhedron (except the outer polyhedron P_O). That is, for some polyhedral graph $G = (V, E, F, \mathcal{P})$, construct $Q = (V, E \cup E', F, \mathcal{P})$, where each edge in E' joins two vertices of a polyhedron $P \in \mathcal{P} \setminus \{P_O\}$. We say that Q is a *quasi-polyhedral graph*, and that G is an *underlying polyhedral graph* of Q (G is not necessarily unique for Q). For brevity, we will usually use the term *polyhedron* rather than the more formal *underlying polyhedron*.

3.2. *The* QUASI-POLYHEDRAL *algorithm*

Similarly to the planar face routing algorithms, QUASI-POLYHEDRAL travels only through polyhedra intersecting the line segment st. Whereas QUASI-PLANAR uses only one reference vertex x, QUASI-POLYHEDRAL stores two reference vertices x and y, maintaining the properties that v, x, y are on the same polyhedron P, and that $\triangle vxy$ intersects st.

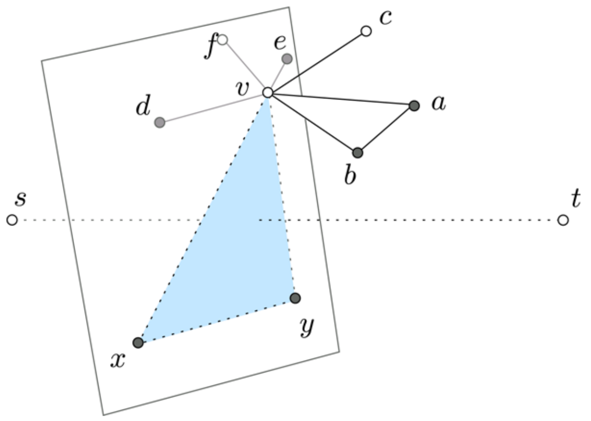

Fig. 6. Candidates for the next vertex include a and b, which are feasible and forward. The other neighbours of v are not candidates since they are infeasible (*e.g.*, c, f) or backward (*e.g.*, d, e, f). The diagram also depicts the plane through $\triangle vxy$. The dashed vertices are feasible.

We will call a neighbour u of v *feasible* if there exists a polyhedron $P \in \mathcal{P}$ whose vertices include v, x, y, and u; otherwise u is *infeasible*. A *feasible face* is a face whose vertices are all feasible; a face with at least one infeasible vertex is an *infeasible face*. Note that a feasible vertex can be incident to many infeasible faces. A vertex $u \in N(v)$ is said to be a *forward vertex* if u is separated from s by the plane through $\triangle vxy$. Otherwise, u is a *backward vertex*. An example illustrating these definitions is depicted in Fig. 6. To determine the first move from s (*i.e.*, the first location of vertex v) and the initial reference vertices x and y, QUASI-POLYHEDRAL uses a subroutine FIND FEASIBLE INITIALISATION (FFINIT). Then QUASI-POLYHEDRAL progresses towards t in each iteration, using a subroutine FIND FORWARD FEASIBLE NEIGHBOUR (FFF) to choose the next vertex from the feasible forward neighbours of the current vertex v. These subroutines are similar to the corresponding computations in QUASI-PLANAR; in particular, FFINIT is analogous to steps 2–3 and FFF to steps 6 and 15. However, there is a subtle geometric complication that requires some explanation, so we delay the descriptions of these subroutines until Section 3.3.

Assuming the correctness of FFF and FFINIT for now, we prove that QUASI-POLYHEDRAL (Algorithm 2) successfully routes on quasi-polyhedral graphs.

Algorithm 2 Quasi-Polyhedral Routing

```
1:  procedure QUASI-POLYHEDRAL($Q, s, t, \mathcal{R}$)
2:      $\{v, x, y\} \leftarrow$ FFINIT($Q, s, t$)
3:      while $vt \notin E$ do
4:          $w \leftarrow$ FFF($Q, s, t, v, x, y$)
5:          if $\triangle wxy$ intersects $st$ then
6:              $v \leftarrow w$
7:          else if $\triangle vwy$ intersects $st$ then
8:              $x \leftarrow y$
9:              $y \leftarrow v$
10:             $v \leftarrow w$
11:         else    $\triangle vxw$ intersects $st$
12:             $y \leftarrow x$
13:             $x \leftarrow v$
14:             $v \leftarrow w$
15:         end if
16:     end while
17:     $v \leftarrow t$
18: end procedure
```

Theorem 3.1. *Given a quasi-polyhedral graph Q and distinct, non-adjacent vertices $s, t \in V(Q)$, the* QUASI-POLYHEDRAL *algorithm successfully routes from s to t.*

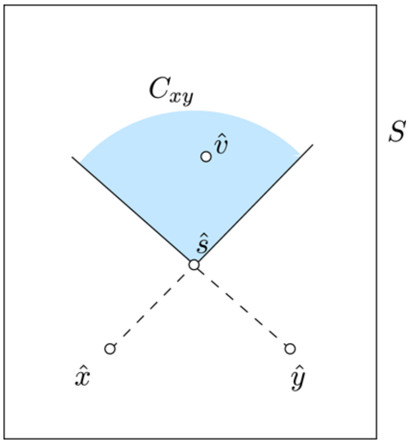

Fig. 7. The cone C_{xy}.

Proof. The proof is structured analogously to the proof of Theorem 2.2. We will show that v, x, and y are on the same underlying polyhedron during the execution of QUASI-POLYHEDRAL. Furthermore, let l_k denote the point of intersection of $\triangle vxy$ with st after the kth iteration of the `while` loop. We will also show that if $v \neq t$ after the kth iteration, then l_k exists and $s = l_0 \prec \cdots \prec l_k \prec t$ where $\prec$ is the natural ordering along st.

The intersection points l_k are determined by triples of distinct vertices in Q, so the sequence $l_0, l_1, \ldots$ has at most $\binom{|V|}{3}$ terms. The `while` loop iterates as long as $vt \notin E$, resulting in a new intersection point with each iteration. Therefore, since this sequence of points is finite, it follows that after some iteration, $vt \in E$. The `while` loop then terminates and v reaches t at step 17.

Therefore, it remains to prove the above two statements (Claims 5 and 7 in what follows). We proceed by induction on k, the number of iterations of the `while` loop in steps 3–16.

Claim 5. *Vertices v, x, and y are on the same underlying polyhedron.*

Proof. For $k = 0$, this follows from the choice of x and y from FFINIT in step 2. For $k \geq 1$, FFF finds a feasible vertex w in step 4. By definition, w is on the same polyhedron as v, x, and y. Steps 5–15 only permute the vertices v, x, and y, and one of them is assigned w. This maintains the desired property. ■

Claim 6. *The intersection point l_k is well-defined, i.e., the triangle $\triangle vxy$ intersects the line segment st.*

Proof. Let ℓ be the line through st. We will first prove that $\triangle vxy$ intersects ℓ, then use this to show that the point of intersection lies on the line segment st.

When $k = 0$, $\triangle vxy$ intersects ℓ at s since $v = s$. For $k > 0$, suppose that $\triangle vxy$ intersected ℓ after the $k-1$st iteration of the `while` loop. Let w be the vertex chosen by FFF in step 4 during the kth iteration of the `while` loop. We will show that at least one of the triangles $\triangle wxy$, $\triangle vwy$, $\triangle vxw$ intersects ℓ.

Project V onto a plane S perpendicular to ℓ, denoting the image of a vertex u by $\hat{u}$. Then the line ℓ is projected onto one point $\hat{s}$. The images $\hat{v}$, $\hat{x}$, and $\hat{y}$ are distinct since $\triangle vxy$ intersects ℓ, and no four vertices are coplanar.

Let C_{xy} be the reflection of $cone(\hat{x}, \hat{s}, \hat{y})$ through its axis of symmetry across $\hat{s}$, as shown in Fig. 7. For any $u \in V$, it is clear that $\triangle uxy$ intersects st if and only if C_{xy} contains $\hat{u}$.

Define C_{vx} and C_{vy} similarly. Then $C_{xy} \cup C_{vx} \cup C_{vy} = S$, so at least one of $\triangle \hat{w}\hat{x}\hat{y}$, $\triangle \hat{v}\hat{w}\hat{y}$, $\triangle \hat{v}\hat{x}\hat{w}$ contains $\hat{s}$. Finally, a triangle intersects ℓ in the original graph if and only if its projection onto S contains $\hat{s}$.

It follows from steps 5–15 that $\triangle vxy$ intersects ℓ at the end of the kth iteration; call the point of intersection l_k. We now show that l_k must lie on the line segment st.

Since $l_0 = s$, we can assume that $k > 0$ and that l_{k-1} lies on st. Let w be the vertex chosen by FFF in step 4. Then, since w is a forward feasible neighbour of v, the vertices v, x, y, and w lie on a polyhedron P; also, w and t are on the same side of the plane through $\triangle vxy$. Therefore, if l_k does not lie on st, t must be contained in P, a contradiction. It follows that one of $\triangle wxy$, $\triangle vwy$, $\triangle vxw$ intersects st, so w will replace one of v, x, y in steps 5–15 such that the desired property is maintained. ■

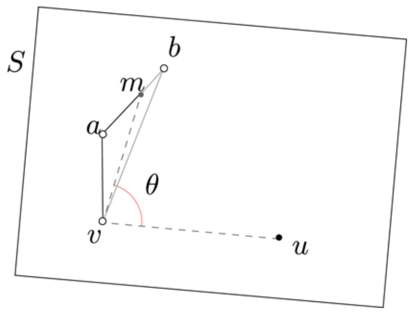

Fig. 8. S is an oriented plane through v, and intersecting ab; u is a point on S. The angle on S from vu to the 3-cycle vab is $\theta = \angle uvm$ where $m = ab \cap S$.

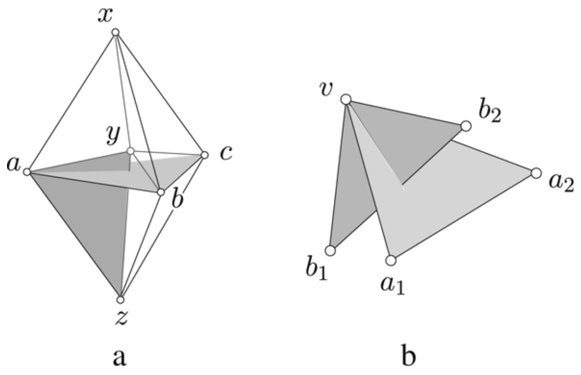

Fig. 9. (a) The edge yz of the 3-cycle ayz intersects the triangle $\triangle abc$. (b) The 3-cycle va_1a_2 dominates vb_1b_2.

Claim 7. *Suppose $v \neq t$. Then $s = l_0 \prec \cdots \prec l_k \prec t$ where $\prec$ is the natural ordering along st.*

Proof. It follows from Claim 6 that l_j is well-defined (*i.e.*, the intersection of $\triangle vxy$ with st exists) and that $s \preceq l_j \prec t$ for all $0 \leq j \leq k$.

Since all underlying polyhedra are convex, the angle between $\triangle wxy$ and $\triangle vxy$ is less than π. The same holds with respect to $\triangle vxy$ for triangles $\triangle vwy$ and $\triangle vxw$. Therefore, $l_j \prec l_{j+1}$ for all $0 \leq j < k$. ■

This concludes the proof of Theorem 3.1. ■

3.3. *The* FFINIT *and* FFF *subroutines*

In this section we describe both the FIND FEASIBLE INITIALISATION (FFINIT) and FIND FEASIBLE FORWARD NEIGHBOUR (FFF) algorithms and prove their correctness. First we need some definitions.

An *oriented plane* in $\mathbb{R}^3$ is a plane S along with two spanning vectors **a** and **b** that play the roles of the standard unit vectors $[10]^T$ and $[01]^T$, respectively, in $\mathbb{R}^2$. We say that S has *orientation* $(\mathbf{a}, \mathbf{b})$. The orientation makes it possible to measure clockwise and counterclockwise angles on S.

Let $C = vab$ be a 3-cycle, and let S be an oriented plane through v intersecting ab at some point m. Let u be a point (not necessarily a vertex) on S. Then $\angle_S uvC$ denotes the counterclockwise angle $\angle uvm$ from u to m around v, as measured on S; similarly $\angle_S Cvu$ denotes the angle $\angle mvu$. See Fig. 8.

This naturally suggests functions $\mathbf{ccw}_S(v, u)$ and $\mathbf{cw}_S(v, u)$ that return the 3-cycle C minimising the non-zero angle $\angle_S uvC$, respectively $\angle_S Cvu$, such that ab intersects S. Note that C is not necessarily unique; for our purposes it is enough to choose a 3-cycle with minimal angle.

We will use these functions in FFINIT and FFF to find initial vertices v, x, y, and candidates for the next vertex, respectively. However, there is one issue to consider before implementing them. Recall Fig. 5, which showed a non-facial 3-cycle. Observe that in this example, yz intersects $\triangle abc$, while no edge intersects $\triangle ayz$, as shown in Fig. 9(a). This indicates a means of identifying some of the "bad" 3-cycles in the graph.

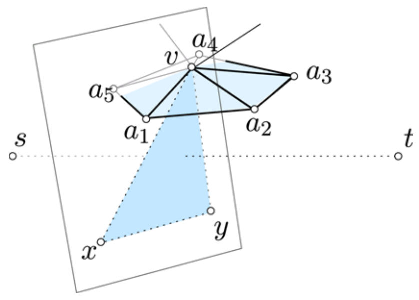

Fig. 10. The five faces $va_1a_2, \ldots, va_5a_1$ are cap faces. The portions of the faces on the forward side of the plane through $\triangle vxy$ are lightly shaded, and the forward portions of the edges are bold.

Let $C_a = va_1a_2$ and $C_b = vb_1b_2$ be 3-cycles, with a_1, a_2, b_1, b_2 distinct. Suppose that the line segment b_1b_2 intersects $\triangle va_1a_2$; see Fig. 9(b). Then we say C_a *dominates* C_b. If a 3-cycle C dominates another 3-cycle, then C is a *dominating* 3-cycle.

We will call the feasible faces incident to v *cap faces*; see Fig. 10. As we show in Lemma 3.2, it is impossible for a cap face to dominate another 3-cycle through v. This allows us to safely ignore all dominating 3-cycles.

Lemma 3.2. *Let $f = va_1a_2$ be a face. Then f does not dominate any other 3-cycle through v.*

Proof. Let f be a face of some polyhedron $P \in \mathcal{P}$. Towards a contradiction suppose f dominates a 3-cycle vb_1b_2. Then b_1b_2 intersects f. Therefore b_1b_2 intersects P, and by convexity of the polyhedra and definition of quasi-polyhedral graphs, at least one of the b_i, say b_1, is contained in P. But since b_1b_2 intersects f, b_2 must be outside P. Since P is convex and b_1b_2 intersects f, $b_1b_2 \notin E \cup E'$, a contradiction. ■

3.3.1. FIND FEASIBLE INITIALISATION *(*FFINIT*)*

The FFINIT subroutine proceeds as follows. First, it chooses an arbitrary oriented plane S through st. Then, by repeated application of $\mathbf{ccw}_S$, it finds the first non-dominating 3-cycle sa_1a_2 counterclockwise from t around s. It then finds the first non-dominating 3-cycle sb_1b_2 clockwise from t around s. The vertices a_1, a_2, b_1, b_2 lie on the same polyhedron, and three of them must form a triangle intersecting st. These three will be the initial assignments to v, x, and y.

Algorithm 3 Find Feasible Initialisation
```
1:  procedure FFINIT($Q, s, t$)
2:      Let $S$ be an oriented plane through $st$.
3:      $l \leftarrow t$
4:      repeat
5:          $sa_1a_2 \leftarrow \mathbf{ccw}_S(s, l)$
6:          $l \leftarrow a_1a_2 \cap S$
7:      until $sa_1a_2$ is not a dominating cycle
8:      $l \leftarrow t$
9:      repeat
10:         $sb_1b_2 \leftarrow \mathbf{cw}_S(s, l)$
11:         $l \leftarrow b_1b_2 \cap S$
12:     until $sb_1b_2$ is not a dominating cycle
13:     return three of $\{a_1, a_2, b_1, b_2\}$ that form a triangle intersecting $st$.
14: end procedure
```

Theorem 3.3. *Let P be the polyhedron through s that intersects $st \setminus s$. The* FFINIT *algorithm returns three vertices lying on P, and the triangle formed by these vertices intersects st.*

Proof. Let S be an oriented plane through st. For simplicity, we may assume that S passes through no vertices other than s and t, so that there are exactly two cap faces of P that intersect $S \setminus s$. Call these cap faces f_a and f_b, where $0 < \angle_S tsf_a < \pi$ and $0 < \angle_S f_bst < \pi$.

By Lemma 3.2, f_a and f_b do not dominate any 3-cycle through s, so both `repeat` loops terminate.

Let $C = sa_1a_2$ be the first non-dominating 3-cycle counterclockwise from t around s, *i.e.*, the final 3-cycle determined by the `repeat` loop in steps 4–7. We will show that a_1 and a_2 lie on P.

If C is a cap face, we are done; therefore, suppose that C is not a cap face. It follows that $0 < \angle_S tsC < \angle_S tsf_a$. Then, since P intersects $st \setminus s$ and $f_a \setminus s$, and P is convex, P intersects $\triangle C \setminus s$. Now, towards a contradiction, suppose that a_1 does not lie on P. Since P intersects $\triangle C \setminus s$, and a_1 is not on P and lies on C, $\triangle C \setminus s$ must intersect some cap face f. By Lemma 3.2, f does not dominate C, so C must dominate f. But this is impossible by choice of C. Therefore a_1 must lie on P. The same reasoning shows that a_2, b_1 and b_2 lie on P.

We now show that (at least) one of the four triangles $\triangle a_1a_2b_1$, $\triangle a_1a_2b_2$, $\triangle a_1b_1b_2$, $\triangle a_2b_1b_2$ intersects st. Project V onto a plane T perpendicular to st, denoting the image of a vertex u by $\hat{u}$. Then st is projected onto one point $\hat{s}$, and the plane S is projected onto a line $\hat{S}$. Both $\hat{a_1}\hat{a_2}$ and $\hat{b_1}\hat{b_2}$ intersect $\hat{S}$, and the points of intersection lie on different sides of $\hat{s}$. Therefore, $\hat{s}$ is in the interior of the quadrilateral with vertices $\hat{a_1}, \hat{a_2}, \hat{b_1}, \hat{b_2}$, so one of the four triangles $\triangle\hat{a_1}\hat{a_2}\hat{b_1}$, $\triangle\hat{a_1}\hat{a_2}\hat{b_2}$, $\triangle\hat{a_1}\hat{b_1}\hat{b_2}$, $\triangle\hat{a_2}\hat{b_1}\hat{b_2}$ contains $\hat{s}$. Since P is convex and intersects $st \setminus s$, the corresponding vertices in V form a triangle intersecting the line through st, and in particular, the point of intersection must lie on the line segment st. ■

3.3.2. FIND FEASIBLE FORWARD NEIGHBOUR *(*FFF*)*

Suppose v, x, and y are on the same polyhedron P, and that $\triangle vxy$ intersects st. To find a feasible forward neighbour of v, FFF uses the same technique as FFINIT: it finds the first non-dominating 3-cycle through v counterclockwise from $\triangle vxy$ and returns one of the endpoints.

Algorithm 4 Find Forward Feasible Neighbour

```
1: procedure FFF($Q, s, t, v, x, y$)
2:     Let $l = \triangle vxy \cap st$.
3:     Let $S$ be the plane through $v, l$, and $t$, with orientation $(lt, lv)$.
4:     repeat
5:         $va_1a_2 \leftarrow \mathbf{ccw}_S(v, l)$
6:         $l \leftarrow a_1a_2 \cap S$
7:     until $va_1a_2$ is not a dominating cycle
8:     return a forward vertex from $\{a_1, a_2\}$.
9: end procedure
```

Theorem 3.4. *The* FFF *algorithm finds a forward feasible neighbour of* v.

Proof. First, the point of intersection $l = \triangle vxy \cap st$ is well-defined by the above assumption on the vertices v, x, y. Let P be the polyhedron through v, x, and y; if these vertices lie on two polyhedra, then consider P to be the one whose intersection with st is closer to t. We can therefore imagine $\mathbf{ccw}_S$ to be sweeping through the interior of P to find successive 3-cycles.

The `repeat` loop in steps 4–7 terminates, since a cap face of P is valid by Lemma 3.2. Let $C = va_1a_2$ be the 3-cycle determined by the `repeat` loop. The same methods as in the proof of Theorem 3.3 show that a_1 and a_2 must lie on P.

Finally, by convexity of P and the choice of orientation of S, $0 < \angle_S lvC < \pi$, so $(C \setminus v) \cap S$ lies in the forward region, *i.e.*, every point of $(C \setminus v) \cap S$ is separated from s by $\triangle vxy$. Therefore, since $a_1a_2 \subset (C \setminus v)$, a_1 and a_2 cannot both be backward vertices. ■

Note that both FFINIT and FFF run in polynomial time, and only use v, x, y, s, and t for their computations. Thus, QUASI-POLYHEDRAL runs in polynomial time; also, the algorithm visits only those vertices on underlying polyhedra properly intersecting st.

Acknowledgements

Many thanks to the Morelia group for numerous inspiring conversations. The first and third authors research were supported in part by NSERC (Natural Science and Engineering Research Council of Canada) grant. The first author's research was also supported in part by MITACS (Mathematics of Information Technology and Complex Systems) grant.

References

[1] D. Avis, K. Fukuda, A pivoting algorithm for convex hulls and vertex enumeration of arrangements and polyhedra, in: Proc. of 7th Annu. ACM Sympos. Comput. Geom., 1991, pp. 98–104.
[2] P. Boone, E. Chávez, E. Gleitzky, E. Kranakis, J. Opatrný, G. Salazar, J. Urrutia, Morelia test: Improving the efficiency of the Gabriel test and face routing in ad hoc networks, in: SIROCCO, in: LNCS, vol. 3104, Springer, 2004.
[3] P. Bose, A. Brodnik, S. Carlsson, E. Demaine, R. Fleischer, A. Lopez, P. Morin, I. Munro, Online routing in convex subdivisions, in: International Symposium on Algorithms and Computation, ISAAC, in: LNCS, Springer, 2000, pp. 47–59.
[4] P. Bose, P. Morin, An improved algorithm for subdivision traversal without extra storage, in: Proceedings of Annual International Symposium on Algorithms and Computation (Taipei, 2000), International Journal of Computational Geometry & Applications 12 (4) (2002) 297–308.
[5] P. Bose, P. Morin, I. Stojmenovic, J. Urrutia, Routing with guaranteed delivery in ad hoc wireless networks, Wireless Networks 7 (2001) 609–616.
[6] E. Chávez, S. Dobrev, E. Kranakis, J. Opatrný, L. Stacho, J. Urrutia, Route discovery with constant memory in oriented planar geometric networks, in: Algosensors, in: LNCS, vol. 3121, Springer, 2004, pp. 147–156.
[7] E. Chavez, S. Dobrev, E. Kranakis, J. Opatrny, L. Stacho, J. Urrutia, Traversal of a quasi-planar subdivision without using mark bits, in: 4th International Workshop on Algorithms for Wireless, Mobile, Ad Hoc and Sensor Networks, WMAN'04, Santa Fe, New Mexico, 2004, Journal of Interconnection Networks 5 (4) (2004) 395–408.
[8] E. Chávez, S. Dobrev, E. Kranakis, J. Opatrný, L. Stacho, J. Urrutia, Local construction of planar spanners in unit disk graphs with irregular transmission ranges, in: 7th Latin American Theoretical Informatics Symposium, LATIN'06, March 2006, Valdivia, Chile.
[9] J. Czyczowicz, E. Kranakis, N. Santoro, J. Urrutia, Traversal of geometric planar networks using a mobile agent with constant memory (in preparation).
[10] C. Gold, U. Maydell, J. Ramsden, Automated contour mapping using triangular element data structures and an interpolant over each irregular triangular domain, Computer Graphic 11 (2) (1977) 170–175.
[11] E. Kranakis, H. Singh, J. Urrutia, Compass routing on geometric networks, in: Proc. of 11th Canadian Conference on Computational Geometry, August 1999, pp. 51–54.
[12] E. Kranakis, L. Stacho, Routing and traversal via location awareness in ad-hoc networks, in: A. Boukerche (Ed.), Handbook of Algorithms for Wireless and Mobile Computing, CRC Press, 2006.
[13] F. Kuhn, R. Wattenhofer, Y. Zhang, A. Zollinger, Geometric ad hoc routing: Of theory and practice, in: Proc. of the 22nd ACM Symposium on the Principles of Distributed Computing, PODC, July 2003.
[14] F. Kuhn, R. Wattenhofer, A. Zollinger, Worst-case optimal and average-case efficient geometric ad hoc routing, in: Proc. of the 4th ACM International Symposium on Mobile Ad Hoc Networking and Computing, MOBIHOC, June 2003.
[15] D. Peuquet, D. Marble, Arc/info: An example of a contemporary geographic information system, in: Introductory Readings in Geographic Information Systems, Taylor & Francis, 1990, pp. 90–99.
[16] D. Peuquet, D. Marble, Technical description of the dime system, in: Introductory Readings in Geographic Information Systems, Taylor & Francis, 1990, pp. 100–111.