TRAVERSAL OF A QUASI-PLANAR SUBDIVISION WITHOUT USING MARK BITS

E. Chávez[1], Š. Dobrev[2], E. Kranakis[3], J. Opatrný[4], L. Stacho[5] and J. Urrutia[6]

Abstract. The problem of traversal of planar subdivisions or other graph-like structures without using mark bits is central to many real-world applications [7, 8, 11, 13, 12, 17, 18]. First such algorithms developed were able to traverse triangulated subdivisions [10]. Later these algorithm were extended to traverse vertices of an arrangement or a convex polytope [3]. The research progress culminated to an algorithm that can traverse any planar subdivision [6, 9]. In this paper, we extend the notion of planar subdivision to quasi-planar subdivision in which we allow many edges to cross each other. We generalize the algorithm from [9] to traverse any quasi-planar subdivision that satisfies a simple requirement. If we use techniques from [6] the worst case running time of our algorithm will be $O(|E|\log|E|)$; which matches with the running time of the traversal algorithm for planar subdivisions [6].

1. Introduction

Graph traversal is a fundamental problem in graph algorithms: Given a starting vertex, can we systematically traverse the entire graph reaching every vertex reachable from the starting one? Many elementary graph algorithms involve making traversal of the graph (e.g., connected component, tree and cycle detection, graph coloring) in order to update their knowledge as they visit each edge and vertex. There have been several studies on traversal in the literature. The best known polynomial traversal algorithm for undirected graphs needs $O(\log^2 n)$ space—$O(\log n)$ variables each storing an address ($O(\log n)$ bits) of a vertex [4, 16]. One can even drop the space to $O(\log^{3/2} n)$ but then time will not be polynomial [15]. Very recently, this was improved to $O(\log^{4/3} n)$ space [2]. There are also randomized $O(\log n)$ space and expected polynomial time traversal algorithms known [1]. Note that $\Omega(\log n)$ space is necessary for any

[1]Escuela de Ciencias Físico-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 científica, Circuito Exterior, Ciudad Universitaria, Coyoacán 04510, México, D.F. México.

traversal algorithm, hence $O(\log n)$ space traversal algorithms are referred to as algorithms *with no extra memory* or *without using mark bits*. For lower bounds on time-space tradeoff of the traversal problem see [5]. In this paper, we are solely interested in traversal algorithms with no extra memory. We gain the improvement of factor $\log n$ compared to the algorithm from [4] by assuming the graph is geometric and satisfies a simple topological condition. This is a significant improvement since, so far, such algorithms have been known only for planar geometric graphs [3, 6, 9, 10].

Our motivation for studying this problem comes from wireless computing: a set of vertices forms spontaneously an ad hoc wireless network. The vertices being aware only of their own geographic location are required to perform fundamental network tasks such as route discovery and broadcasting under various performance parameters such as minimum number of hops, lowest energy consumption, etc. The problem has been considered in several papers including [7, 8, 11, 13, 12]. In all cases, it is assumed that the underlying ad hoc network is preprocessed in order to produce a planar spanner over which a route discovery algorithm can be performed. In this paper we go beyond the existing literature by defining a new class of networks over which the fundamental task of graph traversal can be performed efficiently without prior knowledge of the whole network but rather based solely on local knowledge of the geographic location of the nodes.

A *planar subdivision* is a partitioning of the plane $\mathbb{E}^2$ into a set V of vertices (points), a set E of edges (line segments), and a set F of faces (polygons). In this paper we always consider only finite partitions. Furthermore, we assume that no edge passes through any vertex except its end-vertices. A combinatorial abstraction of a planar subdivision is the planar graph $G = (V, E)$ together with its straight-line embedding into the plane. We will often identify the planar subdivision with its planar graph G in this paper. A subdivision is *connected* if its graph is connected.

Planar subdivisions are finding more and more applications into various real-world problems. For example, they are the basic spatial vector data structure in many geographic information systems [17, 18]. Further recent applications of planar subdivisions can be found in the area of ad hoc wireless networks which we already mentioned above. A fundamental task performed on planar subdivisions is the traversal. Traversing a subdivision involves reporting each vertex, edge, and face of G exactly once, so that some operation can be applied to each. The usual approach to the problem involves a DFS (Depth First Search) of the primal (vertices and edges) or dual (faces and edges) graph. Unfortunately, this technique cannot be implemented without using mark bits on the vertices, edges, or faces, and a stack or queue. If the data structure used to represent the subdivision G does not have extra memory allocated (which is the case for many real-world applications, i.e. the hosts in ad hoc wireless networks are usually very simple devices with limited memory), then an auxiliary array must be allocated and some form of hashing is required to map vertex/edge/face records to array indices. The DFS approach has also another drawback—the traversal cannot be performed simultaneously by more than one thread of execution without some locking mechanism, and of course the memory requirements are increasing.

These problems stimulated an extensive research on traversing planar subdivisions or other graph-like structures without the use of mark bits. One of the first such algorithms developed was for the traversal of a triangulated subdivision [10]. The main idea was to choose one starting point and then define for each triangle unique starting edge through which the triangle can be entered. With careful order and choices one can make sure that each triangle in the subdivision is reported exactly once. In fact, an order is defined on triangles and triangles are reported in this order. This technique is the basis of all subsequent results: In [3], authors describe an algorithm for traversal of vertices of an arrangement or a convex polytope. In [9], authors extend the algorithm to arbitrary planar subdivisions, and very recently, in [6] the running time of this algorithm was improved.

Generally speaking, all algorithms described in [3, 6, 9, 10] use geometric properties of planar subdivisions. In this paper, we look at planar subdivisions as combinatorial objects—graphs consisting of vertices, edges and cycles which give the notion of faces. This allows us to generalize results from [9, 6] to graphs that do not necessarily represent planar subdivisions in $\mathbb{E}^2$. In particular, we define a notion of a quasi-planar subdivision which generalizes the notion of planar subdivision and give an algorithm for traversing quasi-planar subdivisions without the use of mark bits. The worst case running time of our algorithm is $O(|E|\log|E|)$ where E is the number of edges in the quasi-planar subdivision G. Note that if G is a planar subdivision, then $|E| = O(|V|)$ and the running time of our algorithm matches the running time of the best known planar subdivision traversal algorithm [6]. The implementation of our algorithm only requires that every vertex knows its and its neighbors coordinates.

2. Quasi-Planar Subdivisions

In this section, we generalize the notion of planar subdivision and its traversal.

A *quasi-planar subdivision* is a graph $G = (V, E)$ with vertices embedded in the plane and partitioned into $V_p \cup V_c = V$ so that

- vertices in V_p induce a connected planar graph P,
- the outer-face of P does not contain any vertex from V_c or edge of $G - P$, and
- no edge of P is crossed by any other edge of G.

An example of a quasi planar subdivision is depicted in Figure 1.

We will refer to the graph P as an *underlying planar subgraph* and to its faces as *underlying faces*. The notion of vertices and edges is explicit in the definition of quasi-planar subdivision, however, the notion of faces is not. To define the notion of a face, we need to introduce some basic functions on quasi-planar subdivisions. Note that our algorithms do not need to know the partition of V into V_p and V_c. Such a partition is used only in the proofs of correctness of algorithms.

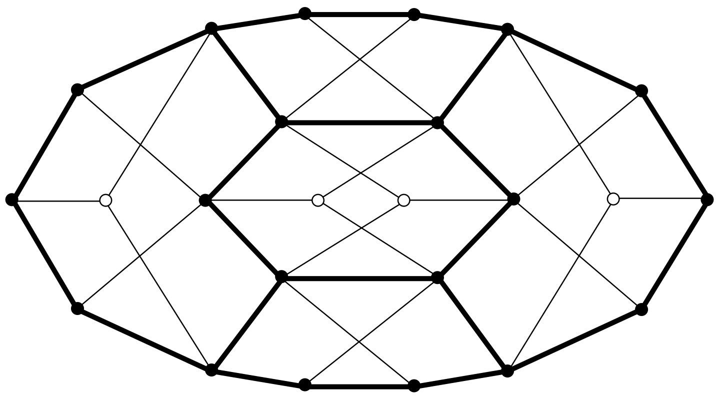

FIGURE 1. An example of a quasi-planar subdivision that satisfies the Left-Neighbor Rule. The filled vertices are in V_p and bold edges are edges of the underlying planar subgraph P.

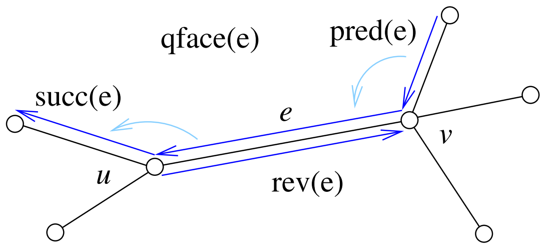

FIGURE 2. Illustration of basic functions on quasi-planar subdivisions.

2.1. **Basic functions on quasi-planar subdivision.** We assume that every vertex u is uniquely determined by pair $[x, y]$ where x is its horizontal coordinate and y is its vertical coordinate. Moreover, we assume that the representation of G is so that every edge $e = uv$ is stored as two oppositely directed edges (u, v) and (v, u). If we need to specify a direction of e, we write either $e = (u, v)$ or $e = (v, u)$, and if the direction is irrelevant, we write $e = uv$. Note that in our algorithm, we will still report each (undirected) edge $e = uv$ exactly once.

For a vertex v, the function **xcor**(v) will return the horizontal coordinate of the vertex v, while the function **ycor**(v) will return the vertical coordinate of v. For an edge $e = (u, v)$, the function **rev**(e) will return a pointer to the edge (v, u). We will sometimes use e^- to denote **rev**(e). Similarly the function **succ**(e) will return a pointer to the edge (v, x) so that (v, x) is the first edge counter-clockwise around v starting from the edge (v, u), and the function **pred**(e) will return a pointer to the edge (y, u) so that (u, y) is the first edge clockwise around u starting from the edge (u, v). For an illustration of these functions see Figure 2. These functions can be easily implemented using so-called doubly-connected edge list structure [14, 19].

Obviously, functions **succ**() and **pred**() are injective, and thus, for every (directed) edge $e = (u, v)$ of G, we can define a closed walk by starting from $e = (u, v)$ and then repeatedly

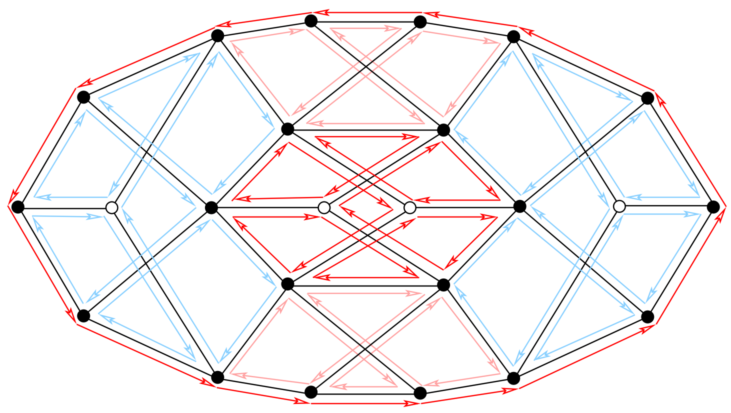

FIGURE 3. A quasi-planar subdivision and its six quasi-faces.

applying the function **succ**() until we arrive at the same edge $e = (u, v)$. Such a walk is called a *quasi-face* of G. The set of all quasi-faces of G is denoted by F. The function **qface**(e) will return a pointer to the quasi-face determined by the (directed) edge $e = (u, v)$. Note that if G is a planar subdivision, then quasi-faces become (regular) faces, and hence the notion of quasi-planar subdivision generalizes the notion of the connected planar subdivision.

The task of *traversing a quasi-planar subdivision* is to report every vertex, (undirected) edge, and quasi-face exactly once in some order. For general quasi-planar subdivisions this seems to be a hard task if we want to perform it without using mark bits and a stack. In the next section, we will show that it is possible to traverse a quite large class of quasi-planar subdivisions.

Definition 1. We say that a quasi-planar subdivision G satisfies a *Left-Neighbor Rule* if every vertex $v \in V_c$ has a neighbor u so that $\mathbf{xcor}(u) < \mathbf{xcor}(v)$. For an example of G that satisfies the Left-Neighbor Rule see Figure 1.

3. Quasi-Planar Subdivision Traversal Algorithm

In this section, we generalize traversal algorithms from [9, 6] so that it will traverse any quasi-planar subdivision $G = (V, E)$ that satisfies the Left-Neighbor Rule. The general idea of the algorithm is the same as the one in [3, 6, 9, 10]: We define a total order $\preceq$ on all edges in E. Using this order, we define a unique predecessor for every quasi-face in F such that the predecessor relationship imposes a virtual directed tree $G(F)$. The algorithm will search for the root of $G(F)$ and then will report quasi-faces of G in DFS order on the tree $G(F)$. For this we use a well-known tree-traversal technique to traverse $G(F)$ using $O(1)$ additional memory. Note that the tree $G(F)$ is never stored in memory and at any given time the algorithm will remember only a constant number of edges (at most two) of this tree. The tree $G(F)$ is used to prove the correctness of our algorithm.

3.1. **The order $\preceq$, the entry edge, and the virtual tree $G(F)$.** In order to define the virtual tree $G(F)$, we determine a unique edge, called an *entry edge*, in each quasi-face. We first define a total order on all edges in E. We write $u \ll v$ if $(\mathbf{xcor}(u), \mathbf{ycor}(u)) \leq (\mathbf{xcor}(v), \mathbf{ycor}(v))$ by lexicographic comparison of the numeric values using $\leq$. For an edge $e = (u, v)$, let

$$\mathbf{left}(e) = \begin{cases} u, \text{ if } u \ll v \\ v, \text{ otherwise} \end{cases}, \quad \mathbf{right}(e) = \begin{cases} v, \text{ if } u \ll v \\ u, \text{ otherwise} \end{cases}, \text{ and}$$

let $\check{u} = [\mathbf{xcor}(u), \mathbf{ycor}(u) - 1]$. Now let $\mathbf{key}(e)$ be the 5-tuple

$$\mathbf{key}(e) = \Big(\mathbf{xcor}(\mathbf{left}(e)),\ \mathbf{ycor}(\mathbf{left}(e)),\ \measuredangle \check{\mathbf{left}}(e)\, \mathbf{left}(e)\, \mathbf{right}(e),\ \mathbf{xcor}(u),\ \mathbf{ycor}(u)\Big).$$

By $\measuredangle abc$ we always refer to the counter-clockwise angle between rays ba and bc with b being the apex of the angle. It follows by our assumption that edges cannot cross vertices that if two edges $e \neq e'$ have the same first three values in their $\mathbf{key}()$, then $e' = e^-$ and hence their last two values in $\mathbf{key}()$ cannot both be the same. Hence it follows that $e = e'$ if and only if $\mathbf{key}(e) = \mathbf{key}(e')$. We define the total order $\preceq$ by lexicographic comparison of the numeric $\mathbf{key}()$ values using $\leq$. For a quasi-face $f \in F$, we define

$$\mathbf{entry}(f) = e \in f \ :\ e \preceq e' \text{ for all } e' \neq e \in f,$$

i.e., $\mathbf{entry}(f)$ is the minimum edge (with respect to the order $\preceq$) on the quasi-face f. Such an edge e will be called the *entry edge* of f. Note that this function is easy to implement using the function $\mathbf{succ}()$, and the total order $\preceq$ using only $O(1)$ memory. We will use the following function

$$\mathbf{ismin}(e) = \begin{cases} \text{T}, \text{ if } e = \mathbf{entry}(\mathbf{qface}(e)) \text{ and } e^- = \mathbf{entry}(\mathbf{qface}(e^-)) \text{ and } e \prec e^-, \\ \text{F}, \text{ otherwise.} \end{cases}$$

Let $e_0 = (u_0, v_0)$ be the minimum edge in the order $\preceq$. The next lemma shows that using the function $\mathbf{ismin}()$ we can test for the minimum edge e_0 in quasi-planar subdivisions that satisfy the Left-Neighbor Rule.

Lemma 1. *If a quasi-planar subdivision G satisfies the Left-Neighbor Rule, then the function* $\mathbf{ismin}(e) = T$ *if and only if* $e = e_0$.

Proof. The proof can be found in the Appendix A-1. $\square$

Lemma 1 guarantees that we can test for the minimum edge e_0 using only the basic functions $\mathbf{entry}()$, $\mathbf{qface}()$, $\mathbf{rev}()$, and $\mathbf{key}()$. This allows us to implement the following function using only $O(1)$ extra memory.

$$\mathbf{parent}(f) = \begin{cases} \mathbf{qface}(\mathbf{rev}(\mathbf{entry}(f))), \text{ if } \mathbf{entry}(f) \neq e_0, \text{ and} \\ \text{NULL, otherwise.} \end{cases}$$

Let us note that it is possible that $\mathbf{parent}(f) = f$, however, we show that if G satisfies the Left-Neighbor Rule, then this never happen. This rule will also guarantee that $\mathbf{entry}(\mathbf{parent}(f)) \prec \mathbf{entry}(f)$ if $\mathbf{entry}(f) \neq e_0$. Our algorithm will identify the edge e_0 and will treat $\mathbf{qface}(e_0)$ differently than all other quasi-faces.

We now define an auxiliary graph which will be used to prove the correctness of our algorithm. Let $G(F) = (F, E(F))$ with

$$E(F) = \left\{(f, f') : \mathbf{parent}(f) = f'\right\}.$$

We prove that if G satisfies the Left-Neighbor Rule, then $G(F)$ is a rooted tree.

Lemma 2. *If a quasi-planar subdivision G satisfies the Left-Neighbor Rule, then for every quasi-face f so that $\mathbf{entry}(f) \neq e_0$, $\mathbf{entry}(\mathbf{parent}(f)) \prec \mathbf{entry}(f)$.*

Proof. The proof can be found in the Appendix A-2. □

Corollary 1. *If a quasi-planar subdivision G satisfies the Left-Neighbor Rule, then for every quasi-face f, $\mathbf{parent}(f) \neq f$.*

Proof. This follows directly from Lemma 2 for any f so that $\mathbf{entry}(f) \neq e_0$. For $\mathbf{qface}(e_0)$, by definition we have $\mathbf{parent}(\mathbf{qface}(e_0)) = \text{NULL}$. □

Theorem 1. *If a quasi-planar subdivision G satisfies the Left-Neighbor Rule, then the graph $G(F)$ is a rooted tree with the root $\mathbf{qface}(e_0)$.*

Proof. We must show that for every quasi-face $f \in F$ there is unique (directed) path from f to $\mathbf{qface}(e_0)$ in $G(F)$. Since every quasi-face has exactly one entry edge, there cannot exist more than one path from f to $\mathbf{qface}(e_0)$. It remains to show that for every $f \in F$, there exists at least one path f to $\mathbf{qface}(e_0)$ in $G(F)$. Suppose by way of contradiction that for some $f \in F$, there is no such path. Now consider the sequence:

$$C = (f, \mathbf{parent}(f), \mathbf{parent}(f)^2, \mathbf{parent}(f)^3, \ldots \mathbf{parent}(f)^i, \ldots).$$

By our assumption $\mathbf{parent}(f)^i \neq \mathbf{qface}(e_0)$ for $i \geq 1$, hence $\mathbf{entry}(\mathbf{parent}(f)^i) \neq e_0$ for $i \geq 1$. Thus, for $i \geq 1$, $\mathbf{entry}(\mathbf{parent}(f)^{i+1}) \prec \mathbf{entry}(\mathbf{parent}(f)^i)$. Moreover, $\mathbf{entry}(f) \neq e_0$, and hence $\mathbf{entry}(\mathbf{parent}(f)) \prec \mathbf{entry}(f)$ and hence all the terms in the sequence C are distinct. Thus, C is an infinite sequence of distinct quasi-faces of G. This contradicts the assumption that G is a finite subdivision. □

3.2. **The Algorithm.** In this subsection, we describe Algorithm 1 which performs traversal on any quasi-planar subdivision G. The reader may check that the algorithm is very similar to the one in [6, 9] and in fact it uses the same technique for reporting vertices, (undirected) edges and quasi-faces of G. However to make the algorithm self-contained, we provide all details here. Let $|ab|$ denote the distance between points a and b. Let $\overrightarrow{ab}$ be the direction of the ray originating at a and containing b. Let $cone(a, b, c)$ denote the cone with apex b, the supporting rays passing through a and c, respectively, and the interior angle $\measuredangle abc$. We will assume that the bounding ray passing through a belongs to $cone(a, b, c)$ but the bounding ray passing through c does not. As noted in [6], all the functions used in the algorithm can be easily implemented using only algebraic functions. Using the results from previous section, we can prove

Theorem 2. *Algorithm 1 reports each vertex, (undirected) edge, and quasi-face of a quasi-planar subdivision G that satisfies the Left-Neighbor Rule exactly once in $O(|E| \log |E|)$ time.*

Proof. The proof can be found in the Appendix A-3. □

The Left-Neighbor Rule condition is essential for Algorithm 1 to successfully traverse quasi-planar subdivisions. If a quasi-planar subdivision does not satisfy the Left-Neighbor Rule condition, then the order $\preceq$ is not guaranteed to be a total order on edges of G and hence there may be several locally minimal edges each playing the role of e_0. Then the traversal algorithm would traverse only a subgraph of G. For an example of a quasi-planar subdivision with two locally minimal edges (u, v) and (x, y) see Figure 4.

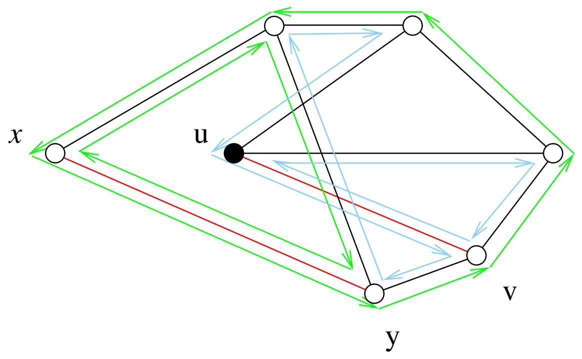

FIGURE 4. The vertex u does not satisfy the Left-Neighbor Rule. Consequently, the function **ismin**() returns T for both edges (u, v) and (x, y).

Algorithm 1 `Traversal of quasi-planar subdivision` $G(V, E)$.

Input: $e = (u, v)$ of $G(V, E)$
Output: List of vertices, edges, and quasi-faces of G in some order.

```
1: repeat {* find the minimum edge $e_0$ *}
2:     $e \leftarrow$ rev($e$)
3:     while $e \neq$ entry(qface($e$)) do
4:         $e \leftarrow$ succ($e$)
5:     end while
6: until $e = e_0$
7: $p \leftarrow \check{\text{left}}(e)$
8: repeat {* start the traversal *}
9:     $e \leftarrow$ succ($e$)
```

```
10:     let $e = (u, v)$ and let **succ**$(e) = (v, w)$
11:     **if** $p$ is contained in $cone(u, v, w)$ **then** {* report $u$ if necessary *}
12:       **report** $u$
13:     **end if**
14:     **if** $|up| < |vp|$ or $(|up| = |vp|$ and $\overrightarrow{up} < \overrightarrow{vp})$ **then** {* report $e$ if necessary *}
15:       **report** $e$
16:     **end if**
17:     **if** $e =$ **entry**(**qface**$(e)$) **then** {* report $e$ and return to parent of **qface**$(e)$ *}
18:       **report qface**$(e)$
19:       $e \leftarrow$ **rev**$(e)$
20:     **else** {* descend to children of **qface**$(e)$ if necessary *}
21:       **if rev**$(e) =$ **entry**(**qface**(**rev**$(e)$)) **then**
22:         $e \leftarrow$ **rev**$(e)$
23:       **end if**
24:     **end if**
25:   **until** $e = e_0$
26:   **report qface**$(e_0)$
```

4. Concluding Remarks

We have generalized a graph traversal algorithm for geometric planar subdivisions [9] (a graph is geometric if every vertex knows its geometric coordinates). The main idea in our algorithm is that of extending the notion of a face in planar subdivision into a closed walk in symmetric directed graph (i.e. directed graph which with every edge (u, v) also contains the edge (v, u)). Thus, our algorithm can traverse (in polynomial time and $O(\log n)$ space) a much wider class of geometric graphs which satisfy a simple geometric condition. The best known polynomial traversal algorithm for non-geometric graphs needs $O(\log^2 n)$ space. One interesting problem remains: Can the geometric condition be dropped from our algorithm by using a more sophisticated approach to define a total order on edges? If so, this would manifest the essential difference between geometric and non-geometric graphs from the graph traversal point of view.

References

[1] R. Aleliunas, R. M. Karp, R. J. Lipton, L. Lovász, and C. Rackoff. Random walks, universal traversal sequences, and the complexity of maze problems. In *20th Annual Symposium on Foundations of Computer Science (San Juan, Puerto Rico, 1979)*, pages 218–223. IEEE, New York, 1979.

[2] R. Armoni, A. Ta-Shma, A. Wigderson, and S. Zhou. An $o(\log^{4/3}(n))$ space algorithm for (s,t) connectivity in undirected graphs. *Journal of the ACM*, 47:294–311.

[3] D. Avis and K. Fukuda. A pivoting algorithm for convex hulls and vertex enumeration of arrangements and polyhedra. In *Proc. of 7th Annu. ACM Sympos. Comput. Geom.*, pages 98–104, 1991.

[4] G. Barnes and W. L. Ruzzo. Undirected s-t connectivity in polynomial time and sublinear space. *Comput. Complexity*, 6(1):1–28, 1996/97.

[5] P. Beame, A. Borodin, P. Raghavan, W. L. Ruzzo, and M. Tompa. A time-space tradeoff for undirected graph traversal by walking automata. *SIAM J. Comput.*, 28(3):1051–1072 (electronic), 1999.

[6] P. Bose and P. Morin. An improved algorithm for subdivision traversal without extra storage. *Internat. J. Comput. Geom. Appl.*, 12(4):297–308, 2002. Annual International Symposium on Algorithms and Computation (Taipei, 2000).

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

[8] J. Czyczowicz, E. Kranakis, N. Santoro, and J. Urrutia. Traversal of geometric planar networks using a mobile agent with constant memory. in preparation.

[9] M. de Berg, M. van Kreveld, R. van Oostrum, and M. Overmars. Simple traversal of a subdivision without extra storage. *International Journal of Geographic Information Systems*, 11:359–373, 1997.

[10] C. Gold, U. Maydell, and J. Ramsden. Automated contour mapping using triangular element data structures and an interpolant over each irregular triangular domain. *Computer Graphic*, 11(2):170–175, 1977.

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

[12] F. Kuhn, R. Wattenhofe, Y. Zhang, and 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.

[13] F. Kuhn, R. Wattenhofe, and 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.

[14] D. E. Muller and F. P. Preparata. Finding the intersection of two convex polyhedra. *Theoret. Comput. Sci.*, 7(2):217–236, 1978.

[15] E. Nisan, E. Szemeredi, and A. Wigderson. Undirecter connectivity in $o(\log^{1.5} n)$ space. In *Proc. of 33rd Annual Symposium on Fundations of Computer Science*, pages 24–29. IEEE, October 1992.

[16] N. Nisan. RL $\subseteq$ SC. *Comput. Complexity*, 4(1):1–11, 1994.

[17] D. Peuquet and D. Marble. Arc/info: an example of a contemporary geographic information system. In *Introductory Readings in Geographic Information Systems*, pages 90–99. Taylor & Francis, 1990.

[18] D. Peuquet and D. Marble. Technical description of the dime system. In *Introductory Readings in Geographic Information Systems*, pages 100–111. Taylor & Francis, 1990.

[19] F. P. Preparata and M. I. Shamos. *Computational geometry: an introduction.* Texts and Monographs in Computer Science. Springer-Verlag, New York, 1985.

APPENDIX

A-1 Proof of Lemma 1

Proof. Let $e = (u, v)$. Suppose by way of contradiction that $e_0 \prec e$, and $\mathbf{ismin}(e) = \mathrm{T}$. Hence $e = \mathbf{entry}(\mathbf{qface}(e))$, $e^- = \mathbf{entry}(\mathbf{qface}(e^-))$ and $e \prec e^-$. If $e = e_0^-$, then $e_0^- \prec e_0$, a contradiction with the minimality of e_0. Hence we may assume $e \neq e_0^-$.

If $\mathbf{left}(e) = v$, then obviously we must have $e^- \prec e$, a contradiction. Hence $\mathbf{left}(e) = u$. Let $\mathbf{pred}(e) = (w, u) =: d$. If $u \notin V_p$, then by the Left-Neighbor Rule, there must exist its neighbor x so that $\mathbf{xcor}(x) < \mathbf{xcor}(u)$. Hence either $\mathbf{xcor}(w) < \mathbf{xcor}(u)$, then $d \prec e$, a contradiction, or $\mathbf{xcor}(w) = \mathbf{xcor}(u)$, but then $\mathbf{left}(d) = w$ and since $\mathbf{ycor}(w) < \mathbf{ycor}(u)$, we have $d \prec e$, a contradiction, or finally $\mathbf{xcor}(w) > \mathbf{xcor}(u)$, but then since $\measuredangle \check{u}uw < \measuredangle \check{u}uv$, we again have $d \prec e$, a contradiction. Hence, $u \in V_p$. Since $e = \mathbf{entry}(\mathbf{qface}(e))$, $\measuredangle \check{u}uv < \measuredangle \check{u}uw$, see Figure 5.

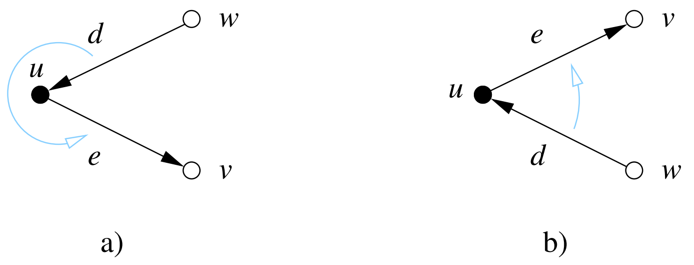

FIGURE 5. The edge $d = \mathbf{pred}(e)$ on the quasi-face $\mathbf{qface}(e)$. The configuration depicted in a) is possible, however the configuration depicted in b) is not possible since $d \prec e$. Grey arrows indicate how the counter-clockwise rule is applied to determine $\mathbf{succ}(d) = e$.

By a similar argument, we can prove that $\mathbf{left}(e_0) = u_0$, $u_0 \in V_p$, and $\measuredangle \check{u}_0 u_0 v_0 < \measuredangle \check{u}_0 u_0 w_0$, where $(w_0, u_0) = \mathbf{pred}(e_0)$.

Since e_0 is the minimum edge and since edges are straight lines and do not cross vertices, the vertex u_0 is on the outer-face of the underlying planar subgraph P. If $\mathbf{xcor}(u) = \mathbf{xcor}(u_0)$, then again since edges are straight lines and do not cross vertices, the vertex u is on the outer-face of the underlying planar subgraph P. Since the outer-face of P does not contain any vertex in V_c or any edge of $G - P$, $e_0 \in \mathbf{qface}(e)$. However this contradicts the assumption that $\mathbf{entry}(\mathbf{qface}(e)) = e$. Hence $\mathbf{xcor}(u_0) < \mathbf{xcor}(u)$.

Since P is connected, there must exist a path C in P joining u_0 to u. Either C contains neither e nor d, or C contains e but not d, or Ccontains d but not e; see Figure 6. Note that since $d = \mathbf{pred}(e)$ then in the first case the neighbor of u on C must lie in between v and w as depicted in Figure 6 a).

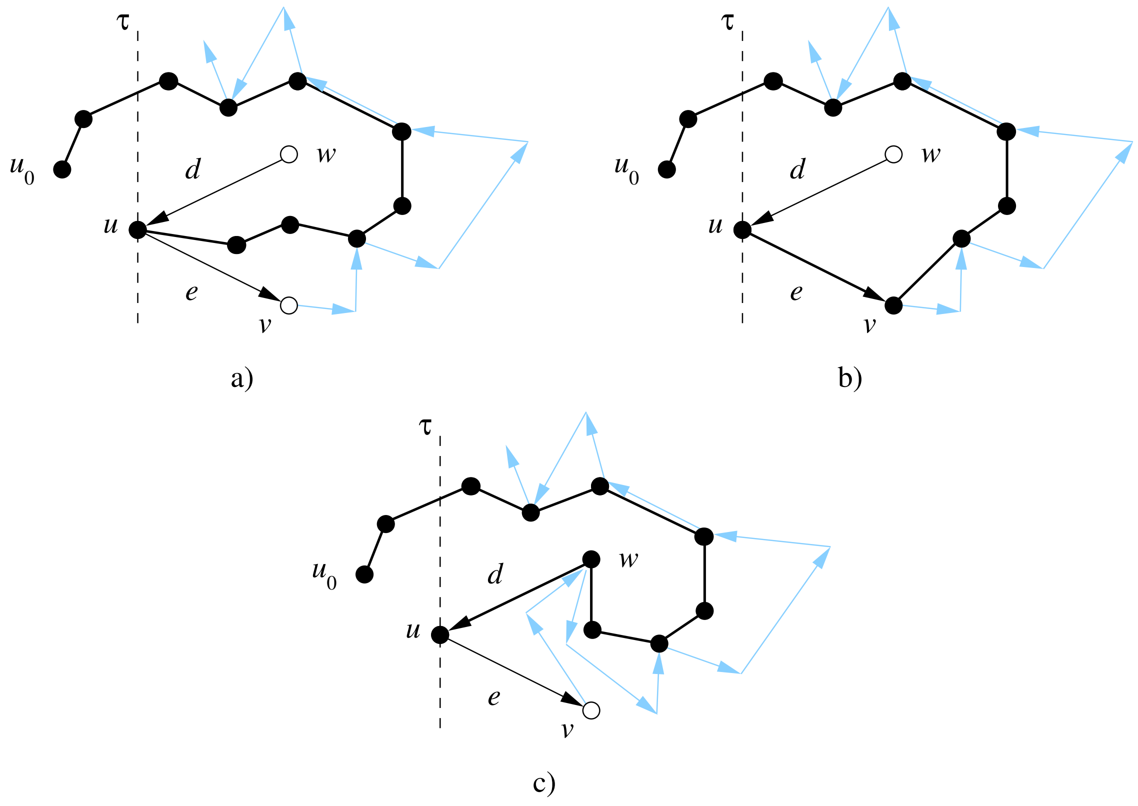

FIGURE 6. Three cases of a mutual incidence of the path C (bold edges) and the edges e and d. The gray arrows represent a part of **qface**(e).

Since C crosses the vertical line τ drawn through the vertex u and since every vertex $x \in$ **qface**(e) has **xcor**$(x) \geq$ **xcor**(u) and since no edge of G crosses edges of P, the edge $d \notin$ **qface**(e), a contradiction since $d =$ **pred**(e), see Figure 6 a) - c).

Next, we must show that **ismin**$(e_0) = \mathrm{T}$. Since e_0 is the minimum edge in the order $\preceq$, we have $e_0 =$ **entry**(**qface**(e_0)) and $e_0 \prec e_0^-$. It remains to show that $e_0^- =$ **entry**(**qface**(e_0^-)). This will follow instantly if we show that there is no edge e so that $e_0 \prec e \prec e_0^-$. Suppose by way of contradiction e exists. Since the first three coordinates in **key**(e_0) are the same as the first three coordinates in **key**(e_0^-), the same must be true for the first three coordinates in **key**(e) and, say, **key**(e_0). However if for any two edges e' and e'' the first three coordinates in **key**(e') equal to the first three coordinates in **key**(e''), then $e' =$ **rev**(e''). Hence $e = e_0^-$, a contradiction. We conclude that $e_0^- =$ **entry**(**qface**(e_0^-)), and hence **ismin**$(e_0) = \mathrm{T}$. This proves the lemma. □

A-2 Proof of Lemma 2

Proof. Suppose by way of contradiction that for some quasi-face f with **entry**$(f) = e \succ e_0$ we have **entry**(**parent**(f)) $\succeq e$. If $e^- \prec e$, then since **entry**(**qface**(e^-)) $\preceq e^-$, we would have $\mathbf{entry}(\mathbf{parent}(f)) \prec e$, a contradiction with our assumption. Hence $e \prec e^-$. By our assumption, for every edge $e' \in \mathbf{qface}(e^-)$, $e \preceq e'$. We show that $e^- \preceq e'$. Suppose by way of contradiction that for some $e' \in \mathbf{qface}(e^-)$, $e \preceq e' \prec e^-$. Since the first three coordinates in $\mathbf{key}(e)$ equal to the first three coordinates in $\mathbf{key}(e^-)$, the same must be true for the first three coordinates in $\mathbf{key}(e')$ and $\mathbf{key}(e)$. It follows from this that $e' = \mathbf{rev}(e) = e^-$, a contradiction. Hence we have proved that $e^- = \mathbf{entry}(\mathbf{qface}(e^-)$. In summary, we have $e \prec e^-$, $e = \mathbf{entry}(\mathbf{qface}(e))$, and $e^- = \mathbf{entry}(\mathbf{qface}(e^-))$. Thus by definition, $\mathbf{ismin}(e) = \mathrm{T}$, and by Lemma 1, $e = e_0$, a contradiction. □

A-3 Proof of Theorem 2

Proof. The algorithm receives a pointer to an edge e of G as an input. Then the algorithm starts to search the $\mathbf{qface}(e^-)$ for its entry edge. When the entry edge is found it will switch to the parent of $\mathbf{qface}(e^-)$, and repeats the search for the entry edge in that quasi-face. Lemma 2 guarantees that either e_0 is found or the new entry edge is smaller (in the total order $\preceq$) than the last entry edge found. Since there is only finitely many edges in G, eventually the algorithm must proceed to line 6 and e will contain a pointer to the edge e_0.

At line 8, the algorithm starts the traversal of G. A proof that the algorithm will report each quasi-face of G exactly once is based of the fact that $G(F)$ is a rooted tree with the root e_0 (Theorem 1) and that our algorithm performs a depth-first search traversal on it. The proof of this is simple and is the same as the corresponding proof in [9], hence omitted.

Our technique of reporting vertices and (undirected) edges of G is essentially the same as the one in [6], hence we again leave out the details of proof. The proof requires a point p that is different from any vertex of G and does not lie on any edge of G. However, the fact that e_0 is the minimum edge in G and the fact that the outer face of P does not contain any vertex in V_c or any edge of $G - P$ guarantee that the point p computed at line 7 will have this property.

The running time analysis of Algorithm 1 are almost the same as in [6] so we only sketch them. If we ignore the cost of the tests in lines 3, 17, and 21, then the preprocessing phase (finding the edge e_0) takes at most $O(|E|)$ time, since each quasi-face is processed at most once and each edge is in at most two different quasi-faces. After that, the main phase (traversal) takes at most $O(|E|)$ time, since each quasi-face is traversed exactly once and each edge is in at most two different quasi-faces. Hence the running time of the algorithm would be $O(|E|)$. Since Algorithm 1 is similar enough to the planar subdivision traversal algorithm from [6], we can use their technique to implement lines 3, 17, and 21, so that the total running time of Algorithm 1 will be $O(\sum_{f\in F} |f| \log |f|)$ which in worst case is $O(|E| \log |E|)$. □