Establishing Order in Planar Subdivisions*

David G. Kirkpatrick

Department of Computer Science, University of British Columbia, Vancouver,
British Columbia, Canada V6T 1W5

Abstract. A planar subdivision is the partition of the plane induced by an embedded planar graph. A representation of such a subdivision is *ordered* if, for each vertex v of the associated graph G , the (say) clockwise sequence of edges in the embedding of G incident with v appears explicitly.

The worst-case complexity of establishing order in a planar subdivision, i.e., converting an unordered representation into an ordered one, is shown to be $\Theta(n + \log \lambda(G))$, where n is the size (number of vertices) of the underlying graph G and $\lambda(G)$ is (essentially) the number of topologically distinct embeddings of G in the plane.

1. Introduction

Let $G = (V, E)$ denote an arbitrary planar graph¹ with vertex set $V = \{1, 2, \dots, n\}$ and edge set E . A (*planar*) *embedding* of G is a drawing of G in the plane in which each vertex is represented as a point and each edge is represented as a simple curve joining its endpoints in such a way that no pair of edges intersect except (possibly) at their end points. An embedding G of G induces a partition of the plane into regions or faces bounded by the edges of G . We call this the *planar subdivision* associated with G . If each of the edges of G is a straight line segment (respectively, a sequence of straight line segments) then G is said to be a *straight-edge* (resp. *linear*) embedding and the associated subdivision is a *straight-edge* (resp. *linear*) subdivision. It is well known that all planar graphs have straight-edge embeddings; in fact every planar embedding has a topologically

* This work was supported by the National Science and Engineering Research Council of Canada under Grant A3583. A preliminary version of this paper appeared in the *Proceedings of the Third Annual ACM Symposium on Computational Geometry*.

¹ Unless otherwise stated it is assumed graphs do not have loops or multiple edges.

equivalent straight-edge embedding [8]. Hereafter we will restrict our attention to linear embeddings.

Since planar graphs and their associated subdivisions arise naturally in the description of polyhedra and planar maps and, either directly or indirectly, in the formulation or solution of numerous problems in computational geometry [19], it is important to establish standard representations for embedded planar graphs. Among the standard representations that have been proposed—and employed in solid modeling and the formulation of fundamental geometric algorithms—are the “winged edge” representation [2], the doubly-connected-edge-list (DCEL) representation [17], [19], and the quad-edge data structure [6], [9]. Each of these is a variant of the same basic structure in which the edges are given in *sorted order* about each of the vertices. They have the desirable properties that:

- (i) Other natural representations are easily shown to be linear-time equivalent (i.e., transformations between representations can be carried out in time linear in the size of the representation) [9], [17].
- (ii) The representations extend to more general settings, for example, the modeling of general polyhedral surfaces [2] and embeddings of undirected graphs in arbitrary two-dimensional manifolds [9].
- (iii) Algorithms which take as their input an embedded planar graph can be described in a natural and efficient manner in terms of these representations. Examples include algorithms for regularizing, triangulating, and dualizing embedded planar graphs and preprocessing planar subdivisions for fast point location [6], [9], [13], [15], [19], [20].

In certain circumstances the construction of one of these standard subdivision representations may require more than linear time. As a consequence, this initialization may dominate the complexity of the associated algorithm. Such a situation can arise, for example, when the input is a straight-edge subdivision presented as a list of vertices, together with their location in the plane, and an *unordered* list of vertex pairs describing the edge set. More generally, each (not necessarily straight) edge of a subdivision may have its embedding described only by the angles at which it meets its two endpoints. Formally, an *endpoint embedding* of the graph G is an assignment of locations in the plane to each vertex of V and an assignment of angles to each end of each edge of E describing its angle of incidence with the corresponding endpoint. We wish to determine the cost, in general, of converting an endpoint embedding of a given graph G into one of the standard (ordered) representations. Clearly, it suffices to compute, for each vertex v of G , a list, denoted $\pi[v]$, of all of the edges incident with v , in clockwise order about v . We call this the problem of *establishing order in a planar subdivision*.

Note that we assume that the given endpoint embedding has a *planar realization*, that is there exists a planar embedding that is consistent with the given endpoint embedding. In effect we are asking what the cost is of recovering the (implicit) topological structure of a planar subdivision given only the combinatorial structure of the underlying graph and the basic geometrical structure of its embedding.

Fig. 1. The graphs (a) $K_{1,n-1}$ and (b) W_{n-1} .

Determining whether or not an arbitrary endpoint embedding has a planar realization is a separate though related question.

First, consider the naive algorithm which explicitly sorts (by angle) all of the edges incident with each vertex of G . This takes $\Theta(\sum_{v \in V} (\deg_G v) \log(\deg_G v))$ where $\deg_G v$ denotes the degree (number of incident edges) in G of vertex v . For graphs of bounded degree this is $\Theta(n)$, but in general it can be as bad as $\Theta(n \log n)$. That $\Theta(n \log n)$ time might be required for certain graphs is easy to see by considering, for example, the graph $K_{1,n-1}$ (see Fig. 1(a)), the star with $n-1$ rays. It is straightforward to reduce the problem of sorting to the problem of establishing order in a subdivision with underlying graph $K_{1,n-1}$. Hence, an $\Omega(n \log n)$ lower bound follows in rather general computational settings [10], [14], [18].

The example of $K_{1,n-1}$ might lead us to conjecture that the naive algorithm is optimal. However, a moment's reflection reveals that linear algorithms exist even for certain cases when the underlying graph has vertices of high degree. It is well known, for example, that three-connected planar graphs have (essentially) a unique planar embedding [7]. Thus, linear-time planarity algorithms [12], [16] which promise only to demonstrate *an* embedding will, in fact, construct *the* embedding, in this case. In other words, the topological structure is implied by the combinatorial structure alone. Thus, for example, order can be established in linear time in a subdivision whose underlying graph is W_{n-1} , the wheel with $n-1$ spokes (see Fig. 1(b)), despite its superficial similarity to $K_{1,n-1}$.

We show that it is the number of distinct embeddings of the underlying graph, rather than the magnitude of the vertex degrees, that contributes to the complexity of establishing order in planar subdivisions. Specifically, if G is a connected planar graph then denote by $\lambda(G)$ the number of topologically distinct planar embeddings of G . If G has planar connected components $G_1, \dots, G_t$, then define $\lambda(G) = \prod_{1 \leq i \leq t} \lambda(G_i)$. We show that the cost of establishing order in planar subdivisions with underlying graph G is $\Theta(n + \log(\lambda(G)))$. Section 2 presents and analyses an algorithm for establishing order in a natural class of directed subdivisions to which our undirected case is easily reduced. Section 3 considers

lower bounds on the complexity of establishing order in subdivisions. An $\Omega(n + \log(\lambda(G)))$ time lower bound on quite general models of computation follows by straightforward information theoretic arguments. Section 4 takes up the planar realizability question. It is shown that this problem is $O(n)$ time reducible to the problem of establishing order. A complementary $\Omega(n + \log(\lambda(G)))$ lower bound that holds for arbitrary fixed-order algebraic decision trees is also outlined.

2. Establishing Order by Repeated Vertex Elimination and Face Contraction

2.1. First Overview

Our algorithm for establishing order in unordered subdivisions is based on three simple observations:

- (i) establishing order at low degree vertices is cheap;
- (ii) the average degree of a planar graph is low; and
- (iii) knowing the order at certain vertices and the combinatorial structure of the underlying graph, constrains the order at other vertices.

It is the detection, representation, and utilization of the constraints mentioned in observation (iii) that makes the algorithm nontrivial.

Our algorithm establishes order in a subdivision incrementally. Vertices are selected (in a carefully chosen fashion), ordered, and then removed. In order to avoid the cost of the naive algorithm, it is necessary to embed within the reduced subdivision information concerning the order established at the removed vertex that constrains the order of the remaining vertices. This is achieved, in part, by the idea of *face shrinking*. On the removal of a vertex v each face incident on v is shrunk; pairs of edges (u, v) , (v, w) that bound the same face are replaced by a single edge (u, w) . The total number of faces remains unchanged and hence the average degree of vertices becomes unbounded. However, since shrinking preserves planarity, the average *effective degree* of vertices (the number of *distinct* adjacent vertices) remains constant. Furthermore, as a result of continued shrinking, faces eventually reduce to simple loops. A loop on vertex v carries information about the ordering of the edges incident on v imposed by the orderings of other vertices on the original (unshrunk) face. This information is used to reduce the cost of establishing order at v . The critical step in the analysis is to show that the remaining cost is attributable to the reduction in the number of embeddings that stems from fixing a particular (realizable) order at vertex v .

2.2. Normal Digraphs and Their Embeddings

In order to make precise the details of our algorithm it turns out to be convenient to consider the problem of establishing order at the vertices of more general combinatorial objects that we call normal subdivisions. A directed graph consists

of a multiset A of ordered pairs of vertices—called *arcs*. The arc (u, v) is said to have *foot* u and *head* v . A digraph H is said to be *normal* if it has an associated bijection $\text{succ}_H: A \rightarrow A$ with the property that the head of a coincides with the foot of $\text{succ}_H(a)$. This ensures, among other things, that for normal digraphs the number of arcs emerging from a given vertex v is the same as the number of arcs entering v . We denote number of arcs emerging from v by $\deg_H v$. The bijection succ_H , and its inverse pred_H , serves to partition the arcs of A into disjoint cycles.

The notion of an embedding (or endpoint embedding) and its associated subdivision extends naturally to planar digraphs. For normal digraphs H we add the constraint that, for each arc $(u, v) \in A$, the arc $\text{succ}_H(u, v)$ must be embedded so as to appear as the (clockwise) next arc, following arc (u, v) , incident with vertex v . As before, we denote by $\lambda(H)$ the product, over all connected components C of H , of the number of topologically distinct planar embeddings of C .

The class of normal digraphs generalizes the class of undirected graphs in the following sense. Let $G = (V, E)$ be any undirected graph. We associate with G the normal digraph $\hat{G} = (V, A)$ in which $A = \{(v, w) | vw \in E\}$ and $\text{succ}_{\hat{G}}(v, w) = (w, v)$, for all $(v, w) \in A$. Note that for each $v \in V$, $\deg_G v = \deg_{\hat{G}} v$.

It should be clear that;

Proposition 1. *Embeddings (resp. endpoint embeddings) of $\hat{G}$ correspond in a one-to-one fashion with embeddings (resp. endpoint embeddings) of G .*

Obviously, digraphs of the form $\hat{G}$ form a very special case of normal digraphs. The real motivation for this notion stems from the role played by normal digraphs in representing the effects of (ordered) vertex deletion.

Let $H = (V, A)$ be a normal digraph, let v be any vertex of H , and let π_v be a cyclic permutation of the arcs of H emerging from v . We denote by $\pi_v - \text{next}(a)$ the next arc, following a , emerging from vertex v according to π_v . Consider the normal digraph $H' = (V - \{v\}, A')$ in which arc a belongs to A' if and only if both the head and foot of a are different from v and either:

- (i) $a \in A$; or
- (ii) there exist $t + 1 \geq 2$ arcs $a_0, a_1, \dots, a_t \in A$ such that:
 1. the foot of a_0 is the foot of a ;
 2. the head of a_i is the head of a ;
 3. the head of a_i is v for $0 \leq i < t$;
 4. the foot of a_j is v for $1 \leq j \leq t$; and
 5. $\text{succ}_H(a_i) = \pi_v - \text{next}(a_{i+1})$ for $0 \leq i < t$.

In case (i) we say that $a = \text{replacement}(a)$. In case (ii) the arc a is said to *bypass* arcs $a_0, a_1, \dots, a_t$ and $a = \text{replacement}(a_i)$ for $0 \leq i \leq t$. Finally, the bijection $\text{succ}_{H'}$ is defined so that if $a = \text{succ}_H(a')$ then $\text{succ}_{H'}(\text{replacement}(a)) = \text{replacement}(a')$. We denote the digraph H' by $H \downarrow \pi_v$.

The whole point of defining the normal digraph $H \downarrow \pi_v$ is the following:

Proposition 2. *The set of topologically distinct planar embeddings of $H \downarrow \pi_v$ corresponds one-to-one with the set of topologically distinct planar embeddings*

of H in which the arcs emerging from v appear (clockwise) in the order specified by π_v .

Proof. The constraints imposed by $\text{succ}_{H'}$, where $H' = H \downarrow \pi_v$, ensure that every planar embedding of $H \downarrow \pi_v$ can be extended to a planar embedding of H in which the arcs emerging from v appear (clockwise) in the order specified by π_v . $\square$

It is Proposition 2 that justifies our assertion that normal digraphs are capable of expressing the constraints associated with partial embeddings of a given graph.

In order to understand the reduction in the number of embeddings of H when we fix the order of edges incident with vertex v , that is the relationship between $\lambda(H)$ and $\lambda(H \downarrow \pi_v)$, it is necessary to characterize the components of H incident with v . Let w be any vertex of H (not necessarily distinct from v). We denote by $\mu_H(v, w)$ the maximum s such that there exist s normal digraphs (called *normal components* of H) $H_1 = (V_1, A_1), \dots, H_s = (V_s, A_s)$, where:

- (i) $\{A_1, \dots, A_s\}$ is a partition of A ;
- (ii) succ_{H_i} is just succ_H restricted to A_i , and
- (iii) $V_i \cap V_j = \{v, w\}$ for all $1 \leq i < j \leq s$.

If $\mu_H(v, w) > 1$, $\{v, w\}$ is said to form a *normal separator* of H . Normal separators are analogous to cutpoints and cutpairs in undirected graphs. It should be intuitively clear that if $\mu_H(v, w)$ is large for some w then, when embedding H , there is a good deal of freedom as to the order of edges incident with v . It turns out that this freedom can be expressed in terms of the values $\mu_H(v, w)$, for $w \in V$, and $\deg_H v$.

Lemma 1.

$$\lambda(H) \geq \lambda(H \downarrow \pi_v) \left(\left\lceil \frac{\deg_H v}{2} \right\rceil \right)^{\lfloor \mu_H(v, v)/2 \rfloor} \prod_{w \neq v} (\mu_H(v, w) - 1)!$$

Proof. Consider any embedding of $H \downarrow \pi_v$. By Proposition 2 this corresponds to an embedding $\mathbf{H}$ of H in which the edges emerging from v appear (clockwise) in the order given by π_v .

Suppose that $\mu_H(v, v) = k \geq 2$. If we remove from $\mathbf{H}$ the $\lfloor k/2 \rfloor$ normal components of H associated with v with the smallest number of edges incident with v , then each of these can be independently reembedded in any of the at least $\lceil (\deg_H v)/2 \rceil$ remaining faces incident with v . This leads to a total of at least $(\lceil \deg_H v/2 \rceil)^{\lfloor k/2 \rfloor}$ distinct embeddings of H which differ only in the order in which normal components associated with v are embedded around v .

Now consider an arbitrary pair $\{v, w\}$, $w \neq v$, and assume that $\mu_H(v, w) = j \geq 2$. The j (embedded) normal components separated by the pair $\{v, w\}$ can be permuted (around v) in $(j-1)!$ ways, each of which gives rise to a distinct embedding of H .

Since all of these modifications to $\mathbf{H}$ can be performed independently, and are uniquely reversible given $\pi(v)$, it follows that there correspond at least

$$\left(\left\lceil \frac{\deg_H v}{2} \right\rceil \right)^{\lfloor \mu_H(v,v)/2 \rfloor} \prod_{w \neq v} (\mu_H(v, w) - 1)!$$

distinct embeddings of H for every embedding of $H \downarrow \pi(v)$. $\square$

2.3. Second Overview

Our algorithm takes as input the normal digraph $\hat{G}$ associated with the planar graph G , together with an endpoint embedding of its arcs. By Proposition 1 it suffices to establish order at the vertices of $\hat{G}$. The algorithm chooses vertices of $\hat{G}$ in an appropriate sequence $v_1, v_2, \dots, v_n$ and constructs a sequence of normal digraphs $H_0, H_1, \dots, H_n$ where $H_0 = \hat{G}$ and $H_i = H_{i-1} \downarrow \pi_{v_i}$. At step i the algorithm determines π_{v_i} , the clockwise order of edges emerging from v_i imposed by the endpoint embedding, given the structure of H_{i-1} . It constructs H_i from H_{i-1} and π_{v_i} . The choice of the elimination sequence is critical to the efficiency of the algorithm. As we shall demonstrate, it is possible to determine with only $O(n)$ overhead a sequence with the property that the cost of determining π_{v_i} given H_{i-1} is $O(\log(\lambda(H_{i-1})) - \log(\lambda(H_i)))$. Thus the total cost is $O(n + \log(\lambda(H_0)))$ (i.e., $O(n + \log(\lambda(G)))$).

In order to keep the overhead linear, we divide the vertex eliminations into phases, each of which involves an independent set of vertices. The number of vertices remaining after each phase is bounded by a geometrically decreasing sequence.

2.4. Data Structures and Pseudocode

The input normal digraph $\hat{G}$ and its endpoint embedding is represented as a collection of arcs A . Each arc $\bar{a} \in A$ is assumed to have the following associated pointers already initialized:

$\bar{a}.foot$	the foot of $\bar{a}$,
$\bar{a}.head$	the head of $\bar{a}$,
$\bar{a}.angle$	the angle at which $\bar{a}$ emerges from $\bar{a}.foot$,
$\bar{a}.sym$	the arc with foot $\bar{a}.head$ and head $\bar{a}.foot$,
$\bar{a}.body$	the most recent arc (if any) to bypass arc $\bar{a}$ (initially null).

The objective of the algorithm is to construct, for each $v \in V$, a list $\pi[v]$ of all of the arcs with foot v , in clockwise order about v .

The algorithm constructs new arcs as it shrinks the faces of $\hat{G}$. Each new arc a has associated with it the pointers:

$a.foot$	the foot of a ,
$a.leg$	the first arc on the path in $\hat{G}$ bypassed by a ,

a.neck the last arc on the path in $\hat{G}$ bypassed by *a*,
a.head the head of *a*.

We abbreviate $((a.neck).sym).body$ by *a.succ* and $((a.leg).sym).body$ by *a.pred*.

Finally, for each *Active* (i.e., undeleted) vertex *v* the algorithm maintains two lists *L[v]* and *New[v]*. *L[v]* is a list of arcs *a* with *a.foot* = *v*. Following the initialization step and until *v* is chosen for removal *L[v]* remains partitioned according to the *a.head* values of its elements. The number of distinct such values is recorded as *effective_degree(v)*. The list *New[v]* records the new arcs *a* with *a.foot* = *v*, constructed during the current phase. While *v* is active $|L[v]| + |New[v]| = \deg_G v$.

In pseudocode our algorithm is as follows:

procedure *establish_order*($\hat{G}$)

1. (* initialize *)
 - for each** $v \in V$ **do**
 - $L[v] \leftarrow New[v] \leftarrow \varnothing$
 - for each** arc $\bar{a} \in A$ **do**
 - create a new arc *a*
 - $a.foot \leftarrow \bar{a}.foot$
 - $a.head \leftarrow \bar{a}.head$
 - $a.leg \leftarrow a.neck \leftarrow \bar{a}$
 - $\bar{a}.body \leftarrow a$
 - add *a* to *L[a.foot]*
 - for each** $v \in V$ **do**
 - partition the arcs *a* on *L[v]* on *a.head* values
 - compute *effective_degree(v)*
 - Active* $\leftarrow V$
2. **while** *Active* $\neq \varnothing$ **do**
 - (* perform another phase of vertex deletions *)
 - 2.1. **for each** $v \in Active$ **do**
 - if** *effective_degree(v)* < 12 and no neighbor of *v* has been chosen
 then choose *v*
 - 2.2. **for each** chosen vertex *v* **do**
 - (* construct $\pi[v]$ *)
 - 2.2.1. (* sort the elements *a* of *L[v]* clockwise about *v* (using $(a.leg).angle$) *)
 - 2.2.2. $\pi[v] \leftarrow \varnothing$
 - for each** arc *a* of *L[v]* in order **do**
 - append *a.leg* to $\pi[v]$
 - (* shrink faces incident with *v* *)
 - 2.2.3. **for each** arc *a* of *L[v]* in order **do**
 - $a' \leftarrow \text{successor of } a \text{ on } L[v]$
 - $b \leftarrow a'.pred$
 - if** $b \neq a$ **then**
 - create a new arc *c*


```

    c.head ← a.head
    c.neck ← a.neck
    c.leg ← b.leg
    c.foot ← b.foot
    (c.neck).body ← (c.leg).body ← c
    if b.foot = v
        then replace b by c on L[v]
        else remove b from L[b.foot]
            add c to New[c.foot]
2.2.4. remove v from Active
2.3. for each w ∈ Active do
    partition New[w] on a.head values
    merge New[w] into L[w]
    New[w] ← ∅
    recompute effective_degree (w)

```

2.5. Correctness and Complexity Analysis

The procedure described above leaves the critical arc sorting step (2.2.1) unspecified. Hence our preliminary analysis of both correctness and complexity is subject to assumptions about this step. In Section 2.6 we present and analyse a (correct) procedure for arc sorting.

Let $v_1, v_2, \dots, v_n$ denote the vertices of $\hat{G}$ indexed in order of their deletion. Let H_i denote the digraph formed by the set of arcs $A_i = \{L[v_j] \cup \text{New}[v_j] \mid j > i\}$ following the removal of vertex v_i (step 2.2.4), and let $\text{succ}_{H_i}(a) = a.\text{succ}$, for each $a \in A_i$. We denote by $\text{sort}(v_i, H_{i-1})$ the cost of the i th execution of step 2.2.1. Subject to the disclaimer above, the correctness of our procedure for establishing order is an immediate consequence of the following lemma.

Lemma 2. *For each v_i , $1 \leq i \leq n$, $\pi[v_i]$ gives a list of the edges of $\hat{G}$ with source v_i , in clockwise order about v_i . Furthermore, $H_i = H_{i-1} \downarrow \pi_{v_i}$, for $1 \leq i \leq n$, where $H_0 = \hat{G}$, and π_{v_i} is the cyclic permutation associated with the list $\pi[v_i]$.*

Proof. It is straightforward to confirm that the i th execution of the face shrinking step 2.2.3 implements the definition of $H_{i-1} \downarrow \pi_{v_i}$. Furthermore, it preserves the following invariant:

$$\text{for each } v \in \text{Active}, \quad \{a.\text{leg} \mid a \in L[v]\} = \{\bar{a} \in A \mid \bar{a}.\text{foot} = v\}.$$

Thus steps 2.2.1 and 2.2.2 construct $\pi[v]$ correctly. □

With regard to complexity, first note that step 1 has cost $O(n)$. Here (and later in the analysis of step 2.3) we note that the required arc partitioning applies to a field (*a.head*) whose values are restricted to $\{1, 2, \dots, n\}$. Hence, using a familiar 2-phase bucket sort (see [1]) all of the lists $L[v]$ can be partitioned in

$O(n)$ time in total. Consider the cost of step 2. Let $Active_i$ denote the vertices *Active* at the beginning of the i th repetition of loop 2. Let $Chosen_i$ denote the set of vertices chosen (and subsequently deleted) in the i th repetition of loop 2. Then, for the i th repetition of loop 2, we have the following costs:

step 2.1. $O(|Active_i|)$,

step 2.2. $O\left(\sum_{v \in Chosen_i} |L[v]| + \sum_{j=n-|Active_i|+1}^{n-|Active_i|+|Chosen_i|} sort(v_j, H_{j-1})\right)$,

step 2.3. $O\left(\sum_{w \in Active_{i+1}} (|New[w]| + effective_degree(w))\right)$.

Lemma 3. *Procedure establish_order has time complexity $O(n + \sum_{i=1}^n sort(v_i, H_{i-1}))$.*

Proof. It is easy to see that:

- (i) $\sum_{w \in Active_{i+1}} |New[w]| \leq \sum_{v \in Chosen_i} |L[v]|$,
- (ii) $\sum_{w \in Active_{i+1}} effective_degree(w) \leq O(|Active_i|)$, by the planarity of H_i , and
- (iii) $|L[v]| \leq \deg_G w$ for all $v \in V$.

Since the sequence $|Active_i|$, $i = 1, 2, \dots$, decreases geometrically,² it follows from the analysis above that the total cost of step 2 (and hence the entire procedure) is $O(n + \sum_{i=1}^n sort(v_i, H_{i-1}))$. $\square$

It remains to focus our attention on the arc sorting step 2.2.1.

2.6. The Arc Sorting Step

The pseudocode expansion of step 2.2.1 is as follows:

- 2.2.1. (* sort the elements a of $L[v]$ clockwise about v (using $(a.leg).angle$) *)
 - 2.2.1.1. partition $L[v]$ into $effective_degree(v)$ lists
 - $L_0, L_1, \dots, L_t$ of arcs a with distinct $a.head$ values
 - (L_0 consists of arcs a with $a.head = a.foot$)
 - 2.2.1.2. **for** each list L_i with $i > 0$ **do**
 - sort the elements a of L_i clockwise about v (using $(a.leg).angle$)
 - 2.2.1.3. **for** each list L_i with $i > 0$ **do**
 - for** each element a of L_i **do**
 - $\hat{a} \leftarrow a.pred$

² Here we use the fact that any planar graph (with no multiple edges) has average degree less than 6 and hence has an independent set of size at least $|V|/24$ consisting of vertices of degree at most 11 (see [13]).

- ```

while $\hat{a}.foot = v$ do
 remove $\hat{a}$ from L_0
 insert $\hat{a}$ immediately before a on L_i
 $a \leftarrow \hat{a}$
 $\hat{a} \leftarrow a.pred$
2.2.1.4. sort the remaining elements of L_0 clockwise
 about v (using $(a.leg).angle$)
2.2.1.5. merge the sorted lists $L_0, L_1, \dots$ into one list $L[v]$
 sorted clockwise about v .

```

Consider the  $i$ th execution of step 2.2.1, when  $v = v_i$ . The correctness of the arc sorting procedure above depends on only one nontrivial observation, namely that the list insertions performed in step 2.2.1.3 preserve the ordering of elements on the lists  $L_1, L_2, \dots, L_t$ . But any assumption to the contrary would mean that two or more arcs of  $H_{i-1}$  are forced to cross, contradicting (via Proposition 2) the (assumed) planar realizability of the original endpoint embedding.

Since the arc sorting procedure is only used when  $effective\_degree(v) < 12$ , it follows that steps 2.2.1.1, 2.2.1.3, and 2.2.1.5 have a total cost which is  $O(|L[v_i]|)$ . It remains to analyse steps 2.2.1.2 and 2.2.1.4. Let  $w_j$  denote the vertex such that  $a.head = w_j$  for all  $a \in L_j$ ,  $1 \leq j \leq t$ . Since the pair  $\{v, w_j\}$  share at least  $|L_j|$  faces of  $H_{i-1}$  it follows that  $\mu_{H_{i-1}}(v, w_j) \geq |L_j|$ ,  $1 \leq j \leq t$ . Step 2.2.1.2 has cost bounded by  $c \sum_{j>0} |L_j| \log(|L_j|)$ , for some constant  $c$ , which is  $O(\sum_{w \neq v_i} \mu_{H_{i-1}}(v_i, w) \log(\mu_{H_{i-1}}(v_i, w)))$ .

Consider the elements remaining on  $L_0$  at the start of step 2.2.1.4. The list  $L_0$  is partitioned into some number  $k$  circular lists by following the pointers  $a.pred$ . The cost of step 2.2.1.4 is  $O(|L[v_i]|)$  plus the cost of merging these  $k$  lists, on a total of  $\deg_{H_{i-1}} v_i$  elements, into one sorted list. Since the elements in each list appear in (clockwise) sorted order about  $v_i$  and since the elements of distinct lists cannot interleave—otherwise the arcs of  $H_{i-1}$  would be forced to cross, leading to the same contradiction as above—this merging can be accomplished incrementally in  $O((k-1) \log(\deg_{H_{i-1}} v_i))$  time. Each of the  $k$  circular lists can be seen to bound faces of  $H_{i-1}$  which isolate a normal component of  $H_{i-1}$  separated by  $\{v_i\}$ . Hence,  $\mu_{H_{i-1}}(v_i, v_i) \geq k$  and the cost of step 2.2.1.4 is  $O(|L[v_i]| + (\mu_{H_{i-1}}(v_i, v_i) - 1) \log(\deg_G v))$ . Since  $|L[v_i]| = \deg_G v_i$  and

$$\begin{aligned}
 & (\mu_{H_{i-1}}(v_i, v_i) - 1) \log(\deg_{H_{i-1}} v_i) + \sum_{w \neq v_i} \mu_{H_{i-1}}(v_i, w) \log(\mu_{H_{i-1}}(v_i, w)) \\
 & \leq O(\log(\lambda(H_{i-1})) - \log(\lambda(H_{i-1} \downarrow \pi_{v_i}))),
 \end{aligned}$$

by Lemma 1, we have established the following:

**Lemma 4.** *The cost of the  $i$ th execution of step 2.2.1,  $sort(v_i, H_{i-1})$ , is  $O(\deg_G v_i + \log(\lambda(H_{i-1})) - \log(\lambda(H_i)))$ .*

Combining Lemma 4 with Lemma 3 yields the following:

**Theorem 1.** *Procedure `establish_order` has time complexity  $O(n + \log(\lambda(\hat{G})))$ .*

**Corollary.** *Establishing order in an arbitrary planar graph  $G$  with  $n$  vertices can be done in  $O(n + \log(\lambda(G)))$  time.*

### 3. Lower Bounds on Establishing Order

The familiar  $\Omega(n \log n)$  information theoretic lower bound for sorting (see [14]) is known to hold on rather powerful models of computation [4], [10], [18]. If the underlying planar graph  $G$  is fixed, it is natural to view the problem of establishing order in a subdivision as a constrained sorting problem. Since any algorithm for establishing order must identify one of the  $\lambda(G)$  possible planar embeddings, the natural information theoretic lower bound for this problem is  $\Omega(\log(\lambda(G)))$ . Specifically, if our model of computation is a comparison tree, each of whose leaves must report a unique embedding of  $G$ , then the worst-case complexity of establishing order is  $\Omega(\log(\lambda(G)))$ . More generally, the following is a direct consequence of Theorem 1 of [10].

**Theorem 2.** *Any RAM with primitive operations comparison, addition, subtraction, multiplication, and division that establishes order in subdivisions with underlying graph  $G$ , where the endpoint embedding is specified by arbitrary rational angles, must use at least  $\Omega(\log(\lambda(G)))$  steps in the worst case.*

Though they address the sorting problem specifically, the techniques of [4] and [18] appear to apply in extending this information theoretic bound to even more powerful RAM models. Since  $\Omega(n)$  is a lower bound on even the input/output activity of any algorithm that establishes order, the upper bound of Section 2 is seen to be optimal to within constant factors. Note that we draw this conclusion despite the fact that our algorithm makes use of indirect addressing, which is not permitted in the model assumed in Theorem 2 above. This is explained by the fact that our algorithm uses indirect addressing only for the sake of *uniformity*—the same algorithm works for all input graphs  $G$ . If  $G$  is fixed the algorithm could—at significant cost in clarity—be formulated as an appropriate RAM program, or even a linear comparison tree program with the same worst-case complexity.

### 4. Planar Realizability of Endpoint Embeddings

The algorithm of Section 2 assumes that the given endpoint embedding of  $G$  has a planar realization. As indicated in the introduction checking the planar realizability of a given endpoint embedding is a separate issue. Note, this topological question is to be distinguished from the geometric problem of checking whether

a given embedding is indeed planar (i.e., free of forbidden intersections). We know of no general method for checking the planarity of even straight-edge embeddings that has complexity  $o(n \log n)$ .

Interestingly, the planar realizability question has a straightforward (linear) reduction to the problem of establishing order. We can assume that the underlying graph  $G$  is planar (since this can be checked in linear time). Suppose we attempt to establish order in  $G$  using the given endpoint embedding. If the procedure *establish\_order* fails for any reason then the endpoint embedding must not have a planar realization. If it succeeds then, whether or not the endpoint embedding has a planar realization, it is possible, by a simple linear-time traversal of the edge lists, to identify the “faces” of the associated subdivision. (In the event that the embedding has no realization these “faces” overlap.) If we augment  $G$  by adding a new vertex  $v_f$  for each “face”  $f$  and an edge  $v_f w$  for each vertex  $w$  incident on  $f$ , the resulting graph  $G^*$  is 3-connected.  $G^*$  is planar if and only if the original endpoint embedding has a planar realization. We conclude that:

**Theorem 3.** *Verifying the planar realizability of a given endpoint embedding of a given graph  $G$  can be done in  $O(n + \log(\lambda(G)))$  time.*

**Corollary.** *If  $G$  is fixed then deciding the planar realizability of a given endpoint embedding of  $G$  can be done in  $O(n + \log(\lambda(G)))$  time.*

Note that a *necessary* condition for an endpoint embedding to have a planar realization is that all of the angles formed by edges at each vertex be distinct. In light of this the planar realizability question can be viewed as a variant of the familiar element uniqueness problem [5]. With this in mind it is not difficult to establish the (asymptotic) optimality of the upper bound given in the corollary above on fixed-order algebraic decision trees [3]. It is straightforward to encode a fixed endpoint embedding of  $G$  as a point in  $\mathbb{R}^{2|E|}$ . Consider the connected components of  $\mathbb{R}^{2|E|}$  that correspond to endpoint embeddings that have planar realizations. Since encodings of endpoint embeddings with topologically distinct planar realizations cannot belong to the same connected component—the argument is a slight variant of the corresponding result for the element uniqueness problem [3]—it follows from Theorem 8 of [3] that:

**Theorem 4.** *For any fixed planar graph  $G$ , the problem of deciding the planar realizability of a given endpoint embedding of  $G$  requires  $\Omega(n + \log(\lambda(G)))$  steps on any fixed-order algebraic decision tree.*

## 5. Conclusion

An alternative approach to establishing order in planar subdivisions might be to find the triconnected components (which can be done in  $O(n)$  time [11]) first and then to establish the order among these components using the available geometric information. What we have demonstrated is that we do not need to

use all of the machinery that seems necessary to identify triconnected components in arbitrary graphs, if we are given as part of the input a description of a planar embedding—a reasonable assumption in many geometric applications.

## Acknowledgments

I am indebted to Raimund Seidel and an anonymous referee for numerous helpful comments on this paper.

## References

1. A. V. Aho, J. E. Hopcroft, and J. D. Ullman, *The Design and Analysis of Computer Algorithms*, Addison-Wesley, Reading, MA, 1974.
2. B. G. Baumgart, A polyhedron representation for computer vision, *AFIPS Conference Proceedings*, Vol. 44 (1975 National Computer Conference, Anaheim, CA), AFIPS Press, Montvale, NJ, 1975, pp. 589–596.
3. M. Ben-Or, Lower bounds for algebraic computation trees, *Proc. 15th Annual ACM Symposium on Theory of Computing*, 1983, pp. 80–86.
4. E. Dittert and M. J. O'Donnell, Lower bounds for sorting with realistic instruction sets, *IEEE Trans. Comput.* **34**, 4 (1985), pp. 311–317.
5. D. P. Dobkin and R. Lipton, On the complexity of computations under varying sets of primitives, *J. Comput. System Sci.* **18** (1979), pp. 86–91.
6. H. Edelsbrunner, L. J. Guibas, and J. Stolfi, Optimal point location in a monotone subdivision, *SIAM J. Comput.* **15**, 2 (1986), pp. 317–340.
7. S. Even, *Graph Algorithms*, Computer Science Press, Potomac, MD, 1979.
8. I. Fary, On straight line representation of planar graphs, *Acta Sci. Math. (Szeged)* **11** (1948), pp. 229–233.
9. L. Guibas, L. Ramshaw, and J. Stolfi, A kinetic framework for computational geometry, *Proc. 24th Annual IEEE Symposium on Foundations of Computer Science*, 1983, pp. 100–111.
10. J. Hong, On lower bounds of time complexity of some algorithms, *Sci. Sinica* **22** (1979), pp. 890–900.
11. J. Hopcroft and R. Tarjan, Dividing a graph into triconnected components, *SIAM J. Comput.* **2**, 3 (1973), pp. 135–158.
12. J. Hopcroft and R. Tarjan, Efficient planarity testing, *J. ACM* **21**, 4 (1974), pp. 549–568.
13. D. G. Kirkpatrick, Optimal search in planar subdivisions, *SIAM J. Comput.* **12**, 1 (1983), pp. 28–35.
14. D. E. Knuth, *The Art of Computer Programming*, Vol. 3, Addison-Wesley, Reading, MA, 1973.
15. D. T. Lee and F. P. Preparata, Location of a point in a planar subdivision and its applications, *SIAM J. Comput.* **6**, 1 (1977), pp. 594–606.
16. A. Lempel, S. Even, and I. Cederbaum, An algorithm for planarity testing for graphs, in *Theory of Graphs* (International Symposium, Rome, July 1966), P. Rosenstiehl (ed.), Gordon and Breach, New York, 1967, pp. 215–232.
17. D. E. Muller and F. P. Preparata, Finding the intersection of two convex polyhedra, *Theoret. Comput. Sci.* **7**, 2 (1978), pp. 217–236.
18. W. J. Paul and J. Simon, Decision trees and random access machines, *Symp. über Logic and Algorithmik*, Zürich, 1980.
19. F. P. Preparata and M. I. Shamos, *Computational Geometry*, Springer-Verlag, New York, 1985.
20. R. E. Tarjan and C. J. Van Wyk, An  $O(n \log \log n)$ -time algorithm for triangulating simple polygons, unpublished manuscript, 1986.

Received May 4, 1987, and in revised form December 5, 1987.