TECHNICAL REPORT NO. 306

Linear-Processor NC Algorithms for Planar Directed Graphs II: Directed Spanning Trees

by

Ming-Yang Kao and Gregory E. Shannon

March 1990

COMPUTER SCIENCE DEPARTMENT
INDIANA UNIVERSITY
Bloomington, Indiana 47405-4101

LINEAR-PROCESSOR NC ALGORITHMS FOR PLANAR DIRECTED GRAPHS II: DIRECTED SPANNING TREES*

MING-YANG KAO† AND GREGORY E. SHANNON‡

Abstract. It has long been a fundamental open problem whether polylog time and *linear* processors are sufficient to find the strongly connected components of a directed graph and compute directed spanning trees for these components. This paper provides the first nontrivial partial solution to the tree problem: for a planar directed graph with n vertices, if the graph is strongly connected, then a directed spanning tree rooted at a specified vertex can be built in $O(\log^2 n)$ time using $O(n)$ processors. The algorithm is deterministic and runs on a parallel random access machine that allows concurrent reads and concurrent writes in its shared memory. The result complements an algorithm by Kao that computes the strongly connected components of a planar directed graph in $O(\log^3 n)$ time and $O(n)$ processors.

1. Introduction. The problems of finding directed spanning trees and strongly connected components frequently appear in more complex problems involving directed graphs. In sequential computation, both problems have linear-time algorithms [1]. In parallel computation using a parallel random access machine, the best algorithms for these two problems are based on matrix multiplication and require $O(\log^2 n)$ time and $O(n^{2.376})$ processors for an n-vertex directed graph [11], [8], [6]. The work of an algorithm can be estimated by the product of its time and processor complexities. Thus, there is a substantial gap between the work done by the above sequential and parallel algorithms. In light of the gap, it has long been a fundamental open problem whether polylog time and *linear* processors are sufficient for computing the strongly connected components and their directed spanning trees. This paper offers the first nontrivial partial solution to the tree problem: for a planar digraph with n vertices, if the graph is strongly connected, then a directed spanning tree rooted at a specified vertex can be built in $O(\log^2 n)$ time and $O(n)$ processors. The algorithm is deterministic and runs on a parallel random access machine that allows concurrent reads and concurrent writes in its shared memory. The result complements an algorithm by Kao that computes the strongly connected components of a planar digraph in $O(\log^3 n)$ time and $O(n)$ processors [10].

The directed spanning tree algorithm and the strongly connected component algorithm share some of their techniques and are based on new insights into the structure of planar digraphs. An insight used in the tree algorithm is as follows. If a strongly connected embedded planar digraph has a maximum degree of at most three and has exactly one *positive face* (a face whose boundary forms a clockwise directed cycle), then a directed spanning tree can be constructed by deleting from the graph an arbitrary edge of the positive face and the *first* edge of each *positive segment* (a maximal clockwise directed path on the boundary of a noncycle face). (See Fig. 1.) This edge cutting technique takes only logarithmic time and linear processors. The efficiency of the technique relies on the fact that edges are chosen and deleted locally. The correctness of the technique derives from a fundamental order property: if an embedded

---

* An extended abstract of this work appeared as one of the two parts in *Local Reorientation, Global Order, and Planar Topology* in the Proceedings of the 21st Annual ACM Symposium on Theory of Computing, Seattle, Washington, May 15-17, 1989, pages 286-296.

† Department of Computer Science, Duke University, Durham, North Carolina 27706. This work was done while the author was at the Department of Computer Science, Indiana University, Bloomington, Indiana 47405.

‡ Department of Computer Science, Indiana University, Bloomington, Indiana 47405.

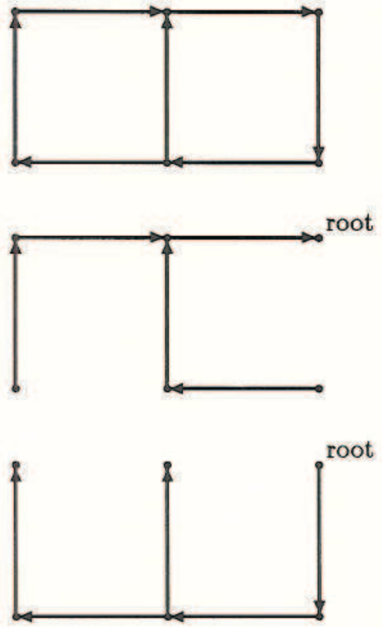

FIG. 1. *Cutting the first and last edges.*

planar digraph is strongly connected, then its dual digraph is acyclic. The acyclicity of the dual graph induces a partial order on the faces of the primal graph. This partial order in turn induces an ear decomposition for the primal graph with the positive face being the first ear and the positive segments being the remaining ears. To build a directed spanning tree for the primal graph, it suffices to remove an arbitrary edge of the first ear and the first edge of each remaining ear.

This simple edge cutting technique can do more. A directed tree is called *convergent* (or *divergent*) if the edges in the tree point from a vertex to its parent (respectively, children). A *CD-pair* of spanning trees is a convergent spanning tree and a divergent spanning tree *rooted at the same vertex*. The spanning tree built above is convergent. Symmetrically, a divergent spanning tree can be built by deleting from the input graph an arbitrary edge of the positive face and the *last* edge of each positive segment. (See Fig. 1.) These two spanning trees form a CD-pair of spanning trees if the two edges deleted from the positive face are chosen appropriately. Given a CD-pair of spanning trees, it is straightforward to *reroot* the pair to compute another CD-pair of spanning trees rooted at a specified vertex. The rerooting can be easily done in logarithmic time using linear processors. Therefore, the edge cutting technique in effect computes a spanning tree oriented in a specified direction and rooted at a specified vertex in logarithmic time and linear processors.

However, the edge cutting technique does not work if the input graph has two or more positive faces or has a maximum degree greater than three. If the graph has exactly one positive face but the maximum degree of the graph is four or greater, then the edge cutting technique sometimes produces a subgraph containing directed cycles. (See Fig. 2.) On the other hand, if the maximum degree of the graph is at most three but the graph has two or more positive faces, then the edge cutting technique always produces a CD-pair of spanning *forest* by deleting an appropriate pair of edges on each positive face and the first and last edges of each positive segment. (A CD-pair

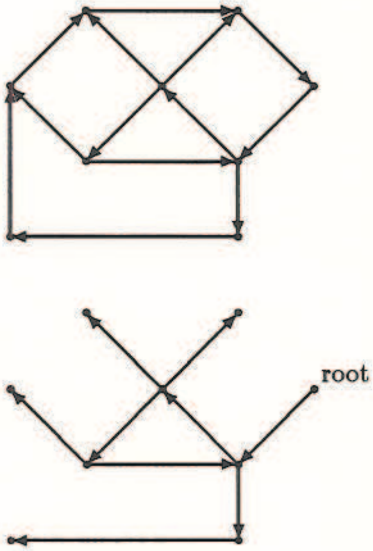

FIG. 2. *Large-degree vertices.*

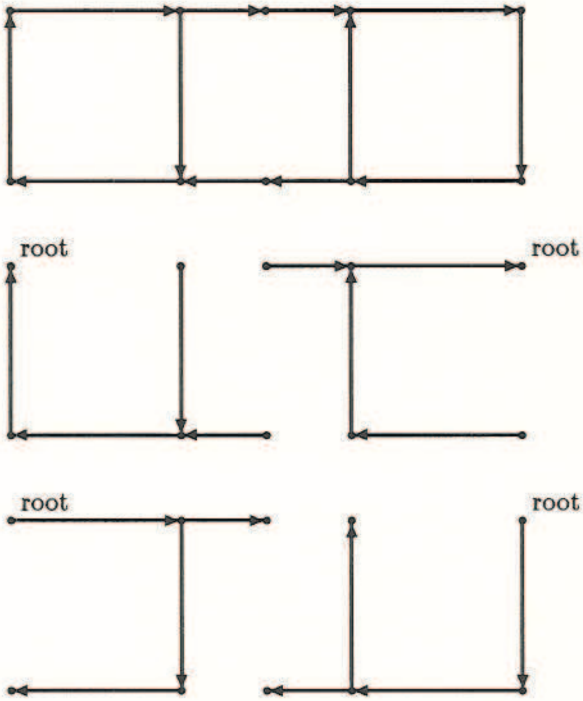
root
root
root
root

FIG. 3. *Multiple positive faces.*

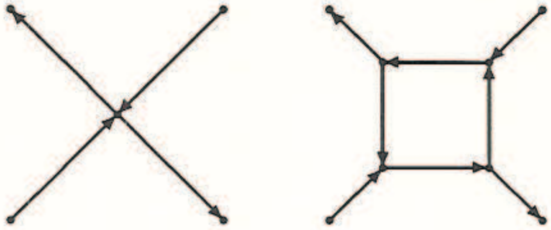

FIG. 4. *Vertex expansion.*

of spanning *forests* is a convergent spanning forest and a divergent spanning forest rooted at the same *vertices.*) (See Fig. 3.) To resolve these two difficulties, the spanning tree algorithm employs several novel ideas to reduce a strongly connected embedded planar digraph into one that has exactly one positive face and a maximum degree at most three.

Reducing the number of positive faces is coupled with merging the CD-pair of spanning forests produced by the edge cutting technique. It is in general very costly to merge a forest into a tree. To pinpoint the problem, let T_1 and T_2 be two disjoint divergent trees. Let $d = u_1 \rightarrow u_2$ be a directed edge from T_1 to T_2. To attach T_2 to T_1 through d, there are two cases. If u_2 is the root of T_2, then a larger divergent tree is readily formed by d, T_1, and T_2. However, if u_2 is not the root of T_2, then merging T_2 to T_1 through d requires rerooting T_2 to u_2. Rerooting is in general very expensive. To facilitate rerooting, the spanning tree algorithm uses the following ideas. A key property of the CD-pair of spanning forests produced by the edge cutting technique is that each positive face is covered by a unique convergent tree and a unique divergent tree. These two trees are rooted at the same vertex in that positive face. They may cover different vertices but they share a well-structured neighborhood of that positive face. Using the shared root and the opposite orientations of these two trees, it is easy to reroot them at any vertex in the neighborhood and thus it is easy to merge the forests in the neighborhood. Furthermore, from its good structure, the neighborhood can be contracted to reduce by half the number of positive faces. Therefore, logarithmic iterations are sufficient to recursively reduce the number of positive faces down to one and recursively merge the forests into a CD-pair of spanning trees. Of course, vertex contraction may produce large-degree vertices and the two major difficulties of the edge cutting technique have be resolved in concert.

To remove large-degree vertices, the basic idea is to replace each vertex of degree greater than three with an appropriate cycle face. (See Fig. 4.) This operation is called *vertex expansion.* There are three problems associated with vertex expansion. The first problem is about undoing vertex expansion. Let F be a graph with large-degree vertices. Let F' be the graph constructed from F by expanding the large-degree vertices into cycle faces. To build a spanning tree for F, the spanning tree algorithm first builds a spanning tree T' for F'. Then, the problem is that T' contains several duplicates for each large-degree vertex of F. The spanning tree algorithm uses a duplicate removal technique to turn T' into a spanning tree of F without duplicates. The second problem is concerned with the graph size. To reduce the number of positive faces, well-structured neighborhoods of positive faces are contracted. Vertex contraction in general creates large-degree vertices. Therefore, it is necessary to expand these new large-degree vertices into cycle faces. Because vertex contraction is repeated several times to recursively reduce the number of positive faces, it is necessary to repeatedly expand large-degree vertices. Now, because vertex expansion increases the size of a graph by a constant factor, there is a danger that repeated vertex expansion might increase the size of the input graph to more than linear and thus make a linear-processor complexity unattainable. By their good structures, the contracted neighborhoods of positive faces are large enough to offset the effect of vertex expansion on the graph size. The third problem is to decide whether large-degree vertices should be replaced by positive faces or negative faces. The replacement by a negative face never increases the number of positive faces, and the replacement by a positive face sometimes destroys more than one positive face and thus makes progress on reducing the number of positive faces. The spanning tree algorithm actually uses different replacement strategies to suit the needs of different stages.

The above discussion has highlighted some of the major difficulties and key ideas in the linear-processor NC algorithm for computing planar directed spanning trees. The following sections proceed to detail the algorithm. Section 2 provides basics of planar digraphs. Section 3 elaborates on the edge cutting technique. Section 4 describes the tree rerooting technique and the duplicate removal technique. Section 5 gives the spanning tree algorithm. Section 6 briefly discusses open problems.

2. Basics for planar digraphs.

A *planar* digraph is one that can be drawn on a plane such that the edges in the drawing intersect only at common ends [9], [4]. A drawing of a planar digraph can be specified by the clockwise cyclic order of edges incident with each vertex. Such a specification is called a *combinatorial embedding* and is useful for algorithmic purposes. For an n-vertex planar digraph, a combinatorial embedding can be found in $O(\log^2 n)$ time using $O(n)$ processors [12]. Recently, the complexity has been reduced to optimal $O(\log n)$ time and $O(n/\log n)$ processors [16]. Because these complexity bounds are comparable to the desired complexity for computing planar directed spanning trees, the tree algorithm assumes without loss of generality that an embedding of its input graph is already given. However, be aware that these embedding complexity bounds are for graphs without multiple edges and loop edges. In the spanning tree algorithm, although these two types of edges can simply be removed without affecting the existence of a directed spanning tree, the edges require special attention. The reason is that multiple edges and loop edges play subtle roles in tracing and analyzing boundary portions of a face that remain unchanged under vertex contraction. These edges are treated as follows. At the beginning, without loss of generality, the input to the directed spanning tree algorithm is a planar digraph without multiple edges and loop edges. Then, the tree algorithm uses vertex contraction to manipulate the input graph. Vertex contraction may create multiple edges and loop edges. Multiple edges must be kept to trace invariant boundary portions of a face; loop edges are useful in the analysis of these portions and are deleted to simplify the contracted graph.

The remaining discussion is divided into two parts. Subsection 2.1 defines basic concepts for planar digraphs and describes the planar primal-dual orientation structure highlighted in the Introduction. Subsection 2.2 defines vertex expansion and vertex contraction, and describes useful boundary portions of a face that remain unchanged under vertex contraction.

2.1. Basic definitions and primal-dual orientation.

Throughout this subsection, let F be a connected embedded planar digraph with more than one vertex. For simplicity, further assume that F is bridge-connected. A *bridge-connected* graph is a connected graph such that the removal of any single edge cannot disconnect the graph. To justify this focus, notice that if F is strongly connected, then F is bridge-connected.

DEFINITION 2.1 (HOLES, FACES, ORIENTATIONS, AND CLUSTERS). Let B be a connected subgraph of F that has more than one vertex. If the vertices and edges of B are removed from the embedding plane of F, then the plane is divided into disconnected regions. Exactly one of the regions is infinite; the others are all finite. The infinite region is called the ***external hole*** of B. Among the finite regions, some may contain edges from F; these regions are called the ***internal holes*** of B. Similarly, if the edges and vertices of F are deleted from the embedding plane of F, then the plane is divided into disconnected regions. Each region is called a ***face*** of F. The ***boundary*** of a face f is the edges and vertices surrounding f. Because F is connected, the boundary of f is connected. Furthermore, because F is bridge-connected, the boundary of f can be arranged into a unique edge-simple undirected cycle. This arrangement is made by an observer staying inside f and walking along the boundary of f. The ***orientation of a boundary edge with respect to f*** is also determined by the observer. A boundary edge is ***positive*** (or ***negative***) with respect to f if it points in the clockwise (respectively, counterclockwise) direction on the boundary cycle of f. This definition of edge orientation is in fact based on the corresponding spherical embedding of F. The face f is called a ***cycle*** face if its boundary cycle is a directed cycle. The face f is called a ***positive*** (or ***negative***) face if f is a cycle face and every boundary edge is positive (respectively, negative) with respect to f. Two positive faces h_1 and h_2 are ***linked*** if there is a sequence of positive faces $f_1, \cdots, f_s$ such that $h_1 = f_1$, $h_2 = f_s$, and for $1 \leq i \leq s-1$, f_i and f_{i+1} share at least one vertex. A ***cluster of positive faces*** is a maximal set of positive faces that are linked. For brevity, a cluster of positive faces is called a ***positive cluster***. Negative clusters can be defined in a symmetric way. The planar directed spanning tree algorithm is described mainly in terms of positive boundary paths, positive faces, and positive clusters. Symmetrically, the algorithm can also be described using only the negative orientation.

DEFINITION 2.2 (DUAL PLANAR DIGRAPHS). The dual of F, denoted by $\tilde{F}$, is the embedded planar digraph constructed as follows. For each face f in F, there is a vertex $\tilde{f}$ in $\tilde{F}$; $\tilde{f}$ is called the dual of f. For each edge d in F, there is an edge $\tilde{d}$ in $\tilde{F}$; $\tilde{d}$ is called the dual of d. The edge $\tilde{d}$ is determined as follows. Let f_1 and f_2 be the two faces in F that share d as a boundary edge. If d is positive (or negative) on f_1, then $\tilde{d}$ is a directed edge from $\tilde{f}_1$ to $\tilde{f}_2$ (respectively, from $\tilde{f}_2$ to $\tilde{f}_1$). Intuitively, $\tilde{F}$ is obtained by placing a vertex in each face of F and turning each edge of F counterclockwise by 90 degrees. Observe that the dual of the dual of F is the same as F with reversed edge directions; consequently, each vertex in F also corresponds to a face in $\tilde{F}$. Finally, observe that a source (or sink) in $\tilde{F}$ corresponds to a positive (respectively, negative) cycle face in F.

THEOREM 2.3 (PRIMAL-DUAL ORIENTATION). *F is strongly connected if and only if $\tilde{F}$ is acyclic. Consequently, if F is strongly connected, then F contains at least one positive face and one negative face.*

Proof. A detailed proof is given in the planar strongly connected component paper [10]. Briefly, this theorem is based on the following folklore observation: for an edge $d \in F$, if the two ends of d are in the same strongly connected component of F, then the two ends of $\tilde{d}$ are in different strongly connected components of $\tilde{F}$. □

2.2. Vertex expansion and vertex contraction. Vertex expansion and vertex contraction are two useful operations in the planar directed spanning tree algorithm. This subsection gives the precise definitions of these two operations and describes useful boundary portions of a face that remain unchanged under vertex contraction.

DEFINITION 2.4 (VERTEX EXPANSION). The planar directed spanning tree algorithm is sensitive to vertices with degree greater than three. For this reason, a vertex of degree four or more is called a *large-degree* vertex. A useful operation for handling large-degree vertices is to replace them each by a positive face or a negative face as shown in Fig. 4. This operation is called *vertex expansion.* In another important application, vertex expansion is used to destroy cycle faces and may apply to vertices of degree three or less. For technical uniformness, if the graph in question consists of a single vertex, then this vertex is expanded into a cycle face with two edges. Finally, vertex expansion for degree two or more is formally defined as follows. Let v be a vertex with degree s. Let $d_1, \cdots, d_s$ be the edges incident with v in the clockwise order. To expand v into a negative face, there are three steps: (1) create s copies of v, namely, $w_1, \cdots, w_s$, (2) replace the v-end of each d_i by w_i, and (3) link the w_i's into a counterclockwise directed cycle with s edges, $w_1 \leftarrow w_2, \cdots, w_{s-1} \leftarrow w_s, w_s \leftarrow w_1$. Expanding v into a positive face can be done by reversing the direction of the edges connecting the w_i's to form a clockwise directed cycle.

DEFINITION 2.5 (VERTEX CONTRACTION). Throughout this paper, vertex contraction always satisfies the following two specifications: (1) to preserve planarity, vertex contraction contracts only connected vertex subsets, and (2) to trace invariant boundary portions of a face, vertex contraction always keeps multiple edges and deletes loop edges as mentioned earlier. The embedding induced by vertex contraction is as follows. Let F be a strongly connected embedded planar digraph. Let B be a connected vertex subset of F. Let F' be the graph constructed from F by contracting B. The embedding for F' is specified as follows. For a vertex $v \notin B$, v remains a distinct vertex in F' and the cyclic order of the edges incident with v is the same in F and F'. For the vertices in B, all these vertices become a single vertex B' in F'. The cyclic order of the edges incident with B' in F' are determined by whether an edge is around the external hole or an internal hole of B in F. The edges around each hole of B stay together around B'. For the edges around the external hole of B, their order around B' is the same as their order around the external hole of B. For the edges around an internal hole of B, their order around B' is opposite to their order around that internal hole. The order difference between the external and internal holes is for maintaining consistency with the edge order around uncontracted vertices. The embedding of F' as defined above is not necessarily unique. In this paper, any induced embedding that satisfies the above definition is sufficient for the planar directed spanning tree algorithm.

The next lemma describes useful boundary portions of a face that remain unchanged under vertex contraction. In general, it is difficult to trace a face under vertex contraction because adjacent faces may be combined and thus lose their individual identities. In contrast, it is easy to trace an edge because although the end vertices of an uncontracted edge may change, the link stays the same and thus keeps its unique identity. For this reason, the next lemma is stated implicitly in terms of uncontracted edges. Be aware that if multiple edges are not kept, then the lemma is false.

LEMMA 2.6. *Let f' be a face in F'. Let P' be an internally vertex-simple path on the boundary of f' such that the two end vertices of P' are new vertices contracted from F but the internal vertices of P' are original vertices in F. Then P' originally is a path on the boundary of a face f in F, and each edge of P' has the same orientation with respect to f and f'.*

Proof. Vertex contraction can be done by performing a sequence of contracting a nonloop edge and deleting a loop edge. The proof of the lemma is by straightforward induction and is based on direct observations of the effects of deleting a loop edge and contracting a nonloop edge. □

3. Edge cutting.

This section elaborates on the edge cutting technique highlighted in the Introduction. The elaboration is divided into four parts. Subsection 3.1 brings out the mechanism behind the deceivingly simple technique. Subsection 3.2 uses the technique to compute a CD-pair of spanning *trees* for a positive cluster. Subsection 3.3 computes a well-structured CD-pair of spanning *forests* for an embedded planar digraph that is strongly connected and has no large-degree vertices. Subsection 3.4 takes advantage of the structure of the forests and computes a useful neighborhood around each positive face. The neighborhood is called the *biconnected territory* of a positive face, and the contraction of the neighborhood is a key step in reducing the number of positive faces.

3.1. The mechanism of edge cutting.

The mechanism behind the edge cutting technique is best exposed via a generalization of the standard ear decomposition. Let F be a digraph. An *ear* of F is an internally vertex-simple directed path, that is, the internal vertices appear only once in the path. A *decomposition* of F is a partition of its edges into ears. An *ordered* decomposition is a decomposition whose ears are arranged into a partial order. A *head* is an ear that is a minimal element in the partial order. A *multihead ear decomposition* of F is an ordered decomposition with the following properties. (1) For each head, its end vertices are the same. (2) Two distinct heads may not intersect at all. (3) For each nonhead ear, its end vertices must be in some smaller ears but its internal vertices may not be in any smaller or incomparable ears.

The next lemma describes the relationship between the notion of multihead ear decomposition and the technique of edge cutting. In the lemma, given a multihead ear decomposition of F, let F_c and F_d be the graphs constructed from F by deleting, respectively, the first edge and the last edge in each ear.

LEMMA 3.1. *F_c and F_d form a* CD*-pair of spanning forests for F. For each head, its vertices are in exactly one tree in each forest; conversely, in each forest, each tree contains the vertices of exactly one head. Each tree is rooted at the end vertex of its corresponding head.*

Proof. The proof is by straightforward induction on the partial order of the ears. The idea is to add into an initially empty graph the ears one by one starting with the heads. The forest shape is enforced by the internal simplicity of ears, the second property, and the second half of the third property. The number of trees in a forest is controlled by the first property, and the second property, and the first half of the third property. □

There are two immediate observations concerning the usage of the lemma. One is that to compute a CD-pair of spanning trees, it suffices to create a decomposition with only one head. The other is that given a multihead ear decomposition, a CD-pair of spanning forests can be computed in constant time using linear processors. Therefore, the complexity of computing a CD-pair of spanning forests is determined by that of computing a multihead ear decomposition.

3.2. Computing spanning trees for positive clusters.

Let F be an embedded planar digraph that may have large-degree vertices. Let Π be a positive cluster of F. A CD-pair of spanning trees for Π can be computed in three stages as follows.

The first stage computes an auxiliary graph for Π. Notice that because F may have large-degree vertices, Π may contain several positive faces and each face may not be vertex-simple. Also notice that a positive face can be decomposed into vertex-simple directed cycles by computing the biconnected components of that face. Let Γ be the set of all such cycles from Π. Let U be the set of vertices shared by cycles in Γ. The auxiliary graph Π' is an *undirected* graph such that (1) the vertex set of Π' consists of the vertices in U and the cycles in Γ with each cycle treated as a single vertex, and (2) for $C \in \Gamma$ and $u \in U$, there is an undirected edge in Π' between C and u if and only u is a vertex in C.

The second stage imposes a total order on the vertex-simple cycles in Π. Observe that because Π is connected, Π' is also connected and thus has a spanning tree. The desired total order on Π is constructed by computing an arbitrary *undirected* spanning tree T of Π' and a preorder numbering of T. This preorder numbering naturally induces a total oder on the cycles in Γ.

The third stage uses the total order to construct for Π an ear decomposition with only one head. The construction is based on the notion of breakpoints and segments. For the smallest-numbered cycle in Γ, the *breakpoint* of the cycle is an arbitrary vertex in the cycle; for each other cycle in Γ, the *breakpoints* of that cycle are the vertices shared by that cycle and smaller-numbered cycles. The breakpoints are defined with respect to a face, and a vertex may be a breakpoint for a face but not for another face. A *segment* is the directed subpath of a cycle between two adjacent breakpoints of the cycle. The segments are the desired ears. The desired partial order $\prec$ of the segments is naturally induced by the preorder numbers of the cycles in Γ. Let L_1 and L_2 be two segments. Let C_1 and C_2 be the cycles that produce L_1 and L_2, respectively. Then $L_1 \prec L_2$ if and only if the preorder number of C_1 is less than that of C_2.

LEMMA 3.2. *The segments and $\prec$ form an ear decomposition with only one head.*

Proof. First of all, because a cycle in Γ is vertex-simple, a segment is internally vertex-simple and thus is an ear. Furthermore, in Γ, except the smallest-numbered cycle, every cycle shares at least one vertex with some smaller-numbered cycle. This vertex sharing property is inherited in essence from the property that in a preorder numbering of a tree, except the smallest-numbered vertex, every vertex is adjacent to at least one smaller-numbered vertex. From the vertex sharing property, every cycle in Γ has at least one breakpoint and thus is partioned into segments. Consequently, the segments form a decomposition of Π. Because the preorder numbering of T induces an order on the cycles in Γ, the order $\prec$ on the segments is a partial order and the decomposition is an ordered one. Because the order on the cycles in Γ is a total order, from the definition of the breakpoints, the segments and $\prec$ satisfy the three properties of a multihead ear decomposition. Finally, because the order on Γ is a total order and the smallest-numbered cycle in Γ produces only one segment, the decomposition has only one head, which is the segment produced by the smallest-numbered cycle. ☐

THEOREM 3.3. *Given an embedded planar digraph, a* CD*-pair of spanning trees for each positive cluster can be constructed in logarithmic time using linear processors in the size of the graph.*

Proof. Lemmas 3.2 and 3.1 can be used to compute a CD-pair of spanning trees for each positive cluster. As for the complexity, the key fact is that from the construction rule for the edges in Π', the size of Π' is linear in the size of Π. Therefore, the total complexity is logarithmic time and linear processors using well-known fundamental algorithms [18], [17], [3], [2], [5], [13], [14]. ☐

3.3. Computing directed spanning forests. Let F be an embedded planar digraph that is strongly connected, has more than one vertex, and contains *no* large-degree vertices. From Lemma 3.1, to compute a CD-pair of spanning forests for F, it suffices to compute a multihead ear decomposition for F. The construction of the desired decomposition is based on the notions of breakpoints and positive segments. For a noncycle face, the *breakpoints* of the face are the local sources and sinks on the boundary of the face; for a positive face, the *breakpoint* of the face is an arbitrary vertex on the face; a negative face has no breakpoints. Notice that the breakpoints are defined with respect to a face and that a vertex may be a breakpoint for a face but not for another face. Also observe that the boundary subpath of a face between two consecutive breakpoints is a directed path. Such a path is called a *segment*. A segment of a face is called *positive* if the edges of the segment are positive with respect to that face. The positive segments are the desired ears. As for a suitable partial order on the positive segments, because F is strongly connected and has more than one vertex, from Theorem 2.3, the dual graph of F is acyclic and thus can be regarded as a partial order $\prec_f$ on the faces of F with the positive faces being the minimal elements. The partial order $\prec_f$ on the faces naturally induces a partial order $\prec_s$ on the positive segments. Let f_1 and f_2 be two faces. Let L_1 and L_2 be two positive segments of f_1 and f_2, respectively. Then $L_1 \prec_s L_2$ if and only if $f_1 \prec_f f_2$.

LEMMA 3.4. *The positive segments and $\prec_s$ form a multi-head ear decomposition. Each positive face is a head and the positive faces are the only heads.*

Proof. Because F is strongly connected, the boundary of a face forms an edge-simple cycle. Moreover, because F has a maximum degree of at most three, the boundary cycle is vertex-simple. Thus, a segment is internally vertex-simple. Because F is strongly connected, an edge in F is contained in exactly two faces and is positive on either face. Therefore, the positive segments together cover all the edges and form a decomposition of F. Because $\prec_f$ is a partial order, $\prec_s$ is also a partial order and the decomposition is an ordered one. Because the positive faces are the minimal elements in $\prec_f$, their segments are the heads. Because a positive face has only one breakpoint, these heads satisfy the first property of a multihead ear decomposition. Because F has a maximum degree of at most three, no two positive faces share a vertex, the heads satisfy the second property of a multihead ear decomposition.

To prove third property, let L_1 be a nonhead ear, and let L_2 be a distinct segment that intersects L_1 at a vertex v. Because F has no large-degree vertices, v cannot be an internal vertex for both L_1 and L_2. Therefore, there are two cases: (1) v is an end vertex of L_1, and (2) v is an internal vertex of L_1 and thus is an end vertex of L_2. These two cases are actually symmetric. The first case proves the first half of the third property and the second case proves the second half of the third property. The following proof only discusses the first case. By definition, v is a source or a sink on the face f_1 that induces L_1. By symmetry, assume without loss of generality that v is a source on f_1. From the strong connectivity of F, the degree of v is at least three. From the small degree of F, the degree of v is exactly three. Let w_1, w_2, and w_3 be the three vertices adjacent with v in the clockwise order. Assume that w_1 and w_2 are on the boundary of f_1 and w_3 is not on that boundary. Then because v is a source, the three edges between v and the w_i's are the following: $d_1 = v \rightarrow w_1$, $d_2 = v \rightarrow w_2$, and $d_3 = w_3 \leftarrow v$. Because the face f_1 contains d_1 and d_2, the segment L_1 must be the one on f_1 that contains d_1 as a positive edge. Let f_2 be the face that contains d_2 and d_3 as positive edges. Then the segment L_2 must be the one on f_2 that contains d_2 and d_3 as positive edges. Now notice that $f_2 \prec_f f_1$. Thus, $L_2 \prec_s L_1$. ☐

In the next theorem, let F_c and F_d be the graphs constructed from F by deleting, respectively, the first edge and the last edge of each positive segment.

THEOREM 3.5. *F_c and F_d can be computed in logarithmic time using linear processors in the size of F. Furthermore, F_c and F_d form a* CD*-pair of spanning forests for F. For each positive face, the vertices of the face is contained in exactly one tree in each forest; conversely, in each forest, each tree contains the vertices of exactly one positive face. Each tree is rooted at the breakpoint of its corresponding positive face.*

Proof. The key fact for the complexity is that the partial orders $\prec_f$ and $\prec_s$ are implicit and require no computation at all. The other statements follow Lemmas 3.1 and 3.4. □

THEOREM 3.6. *Given a strongly connected embedded planar digraph with exactly one positive face and no large-degree vertices, a* CD*-pair of spanning trees can be computed in logarithmic time using linear processors in the size of the graph.*

Proof. By Theorem 3.5, the proof follows the assumption that the graph has exactly one positive face and thus there is only one tree in each of F_c and F_d. □

3.4. Contracting the biconnected territories. The notion of biconnected territory is based on F_c and F_d as follows. Let F be a strongly connected embedded planar digraph without large-degree vertices. Let f be a positive face in F. Let $\mathcal{C}_c(f)$ and $\mathcal{C}_d(f)$ be the trees in F_c and F_d respectively that contain f. Let $\mathcal{S}(f)$ denote the set of vertices shared by $\mathcal{C}_c(f)$ and $\mathcal{C}_d(f)$, and consider $\mathcal{S}(f)$ as an induced subgraph of F. Because f is in $\mathcal{S}(f)$ and is biconnected in F, f is in a biconnected component of $\mathcal{S}(f)$. This biconnected component is denoted by $\mathcal{BT}(f)$ and is called the *positive biconnected territory* of f. For brevity, $\mathcal{BT}(f)$ is called a *biconnected territory* of F.

THEOREM 3.7. *Given a strongly connected embedded planar digraph that has no large-degree vertices, the biconnected territories can be computed in logarithmic time using linear processors in the size of the graph.*

Proof. First of all, F_c and F_d can be easily computed in logarithmic time and linear processors. The $\mathcal{C}_c(f)$'s and $\mathcal{C}_d(f)$'s can be computed in logarithmic time and linear processors, using a well-known connectivity algorithm [17]. More importantly, because from Theorem 3.5, distinct positive faces have disjoint $\mathcal{C}_c(f)$'s and $\mathcal{C}_d(f)$'s, the $\mathcal{S}(f)$'s can be computed in logarithmic time and linear processors in a straightforward manner. Consequently, the biconnected territories can be computed in logarithmic time using linear processors. □

Contracting the biconnected territories is a key step in reducing the number of positive faces. Let F' be the graph constructed from F by contracting each biconnected territory of F into a single vertex. Because $\mathcal{BT}(f)$ is connected, $\mathcal{BT}(f)$ can be contracted without destroying the planarity of F. Therefore F' is planar. Originally, because F has no large-degree vertices, positive faces are disjoint. The contraction of the territories may create large-degree vertices in F' and thus positive faces in F' may cluster together. The next two propositions discuss the relationship between the number of positive clusters in F' and the number of positive faces in F.

In the next lemma, let f be a noncycle face in F. Let P be a directed path on the boundary of f such that P is positive on f and has at least two edges. Notice that the existence of at least two edges ensures that P has at least one internal vertex.

LEMMA 3.8. *If the end vertices of P are in biconnected territories of F but the internal vertices of P are not, then the end vertices of P must be in different biconnected territories.*

Proof. Let v and w be the end vertices of P such that P goes from v to w. Let f_v

and f_w be the positive faces with $v \in \mathcal{BT}(f_v)$ and $w \in \mathcal{BT}(f_w)$. The goal is to show that $\mathcal{BT}(f_v) \neq \mathcal{BT}(f_w)$. To prove by contradiction, assume that $\mathcal{BT}(f_v) = \mathcal{BT}(f_w)$. From Theorem 3.5, this implies $f_v = f_w$. Now, it suffices to show that $f_v = f_w$ implies $P \subseteq \mathcal{BT}(f_w)$, contradicting the assumption that the internal vertices of P are not in any biconnected territory. To show $P \subseteq \mathcal{BT}(f_w)$, notice that because f is a non-cycle face and P is a positive boundary path of f, the path P is a subpath of a positive segment Q of f. Let $P = p_1, \cdots, p_s$ with $v = p_1$ and $w = p_s$. Let $Q = q_1, \cdots, q_t$. After the first edge of Q is cut, the subpath $Q' = q_2, \cdots, q_t$ remains a path in F_c. Because P is a subpath of Q, the path $P_c = p_2, \cdots, p_s$ is a subpath of Q'. Thus, the vertices $p_2, \cdots, p_s$ are connected to $p_s = w$ in F_c. Because $w \in \mathcal{BT}(f_w) \subseteq \mathcal{C}_c(f_w)$, the vertices $p_2, \cdots, p_s$ are in $\mathcal{C}_c(f_w)$. On the other hand, because $p_1 = v \in \mathcal{BT}(f_v) = \mathcal{BT}(f_w) \subseteq \mathcal{C}_c(f_w)$, p_1 is also in $\mathcal{C}_c(f_w)$. In sum, P is in $\mathcal{C}_c(f_w)$. By symmetry, P is also in $\mathcal{C}_d(f_v)$. Because $f_v = f_w$, $\mathcal{C}_d(f_v) = \mathcal{C}_d(f_w)$ and P is in $\mathcal{S}(f_w)$. Finally, from the proof assumption, P is connected to $\mathcal{BT}(f_w)$ at both v and w. Because F has a small degree, the union of P and $\mathcal{BT}(f_w)$ is biconnected in $\mathcal{S}(f_w)$. Consequently, the path P lies in $\mathcal{BT}(f_w)$. □

THEOREM 3.9. *The number of positive clusters in F' is less than half the number of positive faces in F.*

Proof. It suffices to show that each positive face in F' must contain at least two contracted biconnected territories of F. To prove this claim by contradiction, assume that there exists a positive face f' in F' that contains at most one contracted biconnected territory of F. There are two cases based on whether f' contains one or zero contracted biconnected territory of F. Case (1): f' contains zero contracted biconnected territory of F. Then, f' must be a positive face in F. This contradicts the fact that every positive face of F is contracted in F'. Case (2): f' contains exactly one vertex v' that is a contracted biconnected territory of F. Let f'' be a vertex-simple directed subcycle on the boundary f' such that f'' contains v'. From Lemma 2.6, the boundary path P of f'' from v' to v' is originally a positive boundary path of a noncycle face f in F such that (1) the two ends of P are in biconnected territories of F, and (2) the internal vertices of P are not in any biconnected territories of F. Because F' has no loop edges, f'' has at least two edges and thus P has at least two edges. Now from Lemma 3.8, the two ends of P must be in different biconnected territories of F. This contradicts the proof assumption that f' contains exactly one contracted biconnected territory of F. □

4. Tree rerooting and duplicate removal.

As mentioned in the Introduction, tree rerooting and duplicate removal are useful techniques for computing planar directed spanning trees. When used together, duplicate removal provides crucial flexibility to tree rerooting so that forests can be merged efficiently. The details of the two techniques and their combination are given in §4.1, 4.2 and 4.3, respectively

4.1. Tree rerooting.

The next lemma describes the generic technique for tree rerooting. In the lemma, let F be a digraph. Let T_c and T_d be, respectively, a convergent tree and a divergent tree in F that are rooted at the same vertex r. Let U_s and U_u be the sets of vertices, respectively, shared by T_c and T_d, and contained in the union of T_c and T_d. Let n_c and n_d be the numbers of vertices in T_c and T_d, respectively.

LEMMA 4.1 (TREE REROOTING). *Given T_c, T_d, and a vertex $r' \in U_s$, it takes only $O(\log(n_c + n_d))$ time and $O(n_c + n_d)$ processors to compute a convergent tree T'_c and a divergent tree T'_d of F such that T'_c and T'_d are rooted at r', share at least U_s, and contain at most U_u.*

Proof. To create T'_c, the idea is to add into T_c the tree path P_d in T_d from r to r'. The path P_d exists because $r' \in U_s$. In $T_c \cup P_d$, there is a directed path from every vertex to r via T_c then to r' via P_d. The subgraph $T_c \cup P_d$ is not yet a convergent tree rooted at r' because the addition of P_d may cause some vertices to have two outgoing edges. This can be easily remedied by deleting appropriate edges. After the remedy, it is straightforward to verify the remaining statements in the lemma for T'_c. The tree T'_d is constructed and analyzed in a symmetrical way. □

COROLLARY 4.2. *Let F be a strongly connected digraph. Given a* CD*-pair of spanning trees for F, another* CD*-pair of spanning trees for F rooted at a specified vertex can be computed in logarithmic time using linear processors in the size of F.*

Proof. This is a straightforward corollary of Lemma 4.1. □

4.2. Duplicate removal. The next lemma discusses the generic technique for removing duplicates from a tree. In the lemma, let F be a digraph with n vertices. Let $U_1, \cdots, U_s$ be disjoint vertex subsets of F. Let F' be the version of F with the vertices in each U_i identified as a single vertex.

LEMMA 4.3 (DUPLICATE REMOVAL). *Given the U_i's and a directed spanning tree T for F, it takes $O(\log n)$ time and $O(n)$ processors to construct a directed spanning tree T' for F' such that T and T' have the same orientation and the same root.*

Proof. The basic idea is to choose an appropriate representative for U_i. This done in three steps as follows. First, let the representative be the highest vertex u_i of U_i in T; if there are two vertices or more at the same highest level, choose one of them arbitrarily. Second, delete from T all the edges between the vertices in $U_i - \{u_i\}$ and their parents. Third, contract U_i into u_i. The resulting graph is the desired tree T'. It is straightforward to verify that T' is indeed a directed spanning tree for F' with the same orientation and the same root as those of T. As for the complexity, the key fact is that the representative vertices can be found in $O(\log n)$ time and $O(n)$ processors using well-known tree contraction algorithms [14]. □

COROLLARY 4.4. *Let F be a strongly connected embedded planar digraph. Let F' be constructed from F by replacing some vertices each with a positive face or a negative face. Then given a* CD*-pair of spanning trees for F', a* CD*-pair of spanning trees for F can be constructed in logarithmic time using linear processors in the size of F.*

Proof. This is a straightforward corollary of Lemma 4.3 with F and F' in the reversed roles. The key observation is that for each expanded vertex of F, there is a U_i consisting of the vertices of the corresponding cycle face in F'. As for the complexity, the key fact is that the size of F' is at most five times the size of F. □

4.3. Combining tree rerooting and duplicate removal. The next lemma combines the tree rerooting technique and the duplicate removal technique. In the lemma, let F be a strongly connected digraph. Let $U_1, \cdots, U_s$ be disjoint vertex subsets of F. Let $S_{i,c}$ and $S_{i,d}$ be a CD-pair of trees in F that each cover U_i. The trees $S_{i,c}$ and $S_{i,d}$ may contain vertices not in U_i; this condition provides necessary flexibility for Theorem 4.6 given below. Let F' be the graph constructed from F by contracting each U_i into a single vertex. Let T'_c and T'_d be a CD-pair of spanning trees for F'.

LEMMA 4.5 (TREE REROOTING AND DUPLICATE REMOVAL). *Let n be the number of vertices in F. Let α be the sum of the numbers of vertices in the $S_{i,c}$'s and $S_{i,d}$'s. Given T'_c, T'_d, the $S_{i,c}$'s, and the $S_{i,d}$'s, it takes $O(\log(\alpha + n))$ time using $O(\alpha + n)$ processors to compute a* CD*-pair T_c and T_d of spanning trees for F.*

Proof. By symmetry, it suffices to construct T_c in two stages as follows. The first stage applies Lemma 4.1 to every $S_{i,c}$ and $S_{i,d}$. Let u'_i be the vertex in F' that is contracted from U_i. The supervertex u'_i in T'_c is expanded back to an appropriate tree covering U_i in two steps: (1) choose a vertex u_i at which T'_c enters U_i, and (2) apply Lemma 4.1 to $S_{i,c}$ and $S_{i,d}$ to find a convergent tree $R_{i,c}$ in F that is rooted at u_i and covers U_i. Let T_c be the subgraph of F resulting from the first stage. Observe that T_c is very close a convergent spanning tree for F. The only remaining problem is that T_c may have duplicates because $S_{i,c}$ and $S_{i,d}$ may contain vertices not in U_i. The second stage simply uses Lemma 4.3 to remove the duplicates from T_c. As for the complexity, the tree rerooting stage clearly runs in $O(\log\alpha)$ time using $O(\alpha)$ processors. The duplicate removal stage relies on the fact that the number of extra copies of vertices concerning each U_i cannot exceed the total number of vertices in $S_{i,c}$ and $S_{i,d}$. Thus, before the duplicates are removed, the total number of vertices in T_c is bounded by $\alpha + n$. Consequently, the total complexity for the duplicate removal stage is at most $O(\log(\alpha + n))$ time using $O(\alpha + n)$ processors. In sum, the total complexity for computing T_c is $O(\log(\alpha + n))$ time using $O(\alpha + n)$ processors. □

THEOREM 4.6. *Let F be a strongly connected embedded planar digraph without large-degree vertices. Let F' be constructed from F by contracting each biconnected territory into a single vertex. Then given a* CD*-pair of spanning trees for F', a* CD*-pair of spanning trees for F can be constructed in logarithmic time using linear processors in the size of F.*

Proof. This theorem combines Lemma 4.5 and Theorem 3.5. The complexity derives from the fact that the CD-pair of spanning forests computed for F in Theorem 3.5 has a total size less than twice the size of F. □

THEOREM 4.7. *Let F be a strongly connected embedded planar digraph. Let F' be constructed from F by contracting each positive cluster into a single vertex. Then given a* CD*-pair of spanning trees for F', a* CD*-pair of spanning trees for F can be constructed in logarithmic time using linear processors in the size of F.*

Proof. This theorem combines Lemma 4.5 and Theorem 3.3. The complexity derives from the fact that the CD-pairs of spanning trees for the positive clusters computed in Theorem 3.3 have a total size less than twice the size of F. □

5. Computing directed spanning trees. The main result of this paper is stated in the following theorem.

THEOREM 5.1. *Let G be a strongly connected embedded planar digraph. Let n be the size of G. Let β be the number of positive clusters in G. A* CD*-pair of spanning trees for G can be computed in $O(\lceil\log(\beta + 1)\rceil \log n)$ time using $O(n)$ processors. Consequently, from Corollary 4.4, a directed spanning tree with a specified orientation and a specified root can be computed in the same complexity. Because $\beta < n$, the time complexity is at most $O(\log^2 n)$. The algorithms are deterministic and run on a parallel random access machine that allows concurrent reads and concurrent writes in its shared memory.*

To prove the theorem, it suffices to describe how to build a CD-pair of spanning trees for G. The description assumes that G consists of more than a single vertex to avoid triviality. The theorem is first reduced to a restricted version in which the graph in question has no large-degree vertices. The reduction is through the following three steps. (1) Let G_1 be constructed from G by contracting each positive cluster into a single vertex. (2) Let G_2 be constructed from G_1 by expanding into a *positive* face each contracted positive cluster of G. (3) Let H be constructed from G_2 by expanding each large-degree vertex into a *negative* face. Notice that if G consists of a positive cluster, then G_1 consists of a single vertex. As mentioned in Definition 2.4, for technical uniformness, this single vertex is expanded into a positive face with two edges in G_2.

The following facts about H are straightforward. Because vertex expansion and vertex contraction preserve planarity and strong connectivity, H is a strongly connected embedded planar digraph. The third step ensures that H has no large-degree vertices. This degree property in turn ensures that no two positive faces in H share a vertex. Thus, the number of positive clusters in H is the same as the number of positive faces in H. As for the complexity, observe that $|G_1| < n, |G_2| < 5n, |H| < 5n$, where $|\cdot|$ denotes the size. Consequently, H can be built easily in $O(\log n)$ using $O(\log n)$ processors.

With the above basic facts established, the next two lemmas proceed to describe the key relationship between H and G.

LEMMA 5.2. *The number of positive faces in H is equal to the number of positive clusters in G.*

Proof. In the first step of the reduction, the contraction of each positive cluster into a single vertex destroys the positive faces in G but may create new positive faces in G_1. Because vertex contraction preserves the edge order around an uncontracted vertex, each new positive face must contain at least one contracted cluster. In the second step, the expansion of each contracted cluster into a positive face destroys all new positive faces. Therefore, the number of positive faces in G_2 is the same as the number of positive clusters in G. In the third step, the expansion of each large-degree vertex into a negative face cannot destroy or create positive faces. Thus, the number of positive faces of H is equal to that of G_2. In sum, the number of positive faces in H is equal to the number of positive clusters in G. ☐

LEMMA 5.3. *Given a* CD-*pair spanning trees for H, a* CD-*pair spanning trees for G can be constructed in $O(\log m)$ using $O(m)$ processors.*

Proof. The given CD-pair for H can be iteratively converted into a CD-pair of spanning trees first for G_2 via Lemma 4.3, then for G_1 via Lemma 4.3, and finally for G via Theorem 4.7. From the size relationship among G, G_1, G_2 and H, the total complexity is $O(\log m)$ using $O(m)$ processors. ☐

In view of the above discussion, to prove Theorem 5.1, it suffices to prove the theorem assuming that the graph in question has no large-degree vertices. This is formally stated in the next theorem.

THEOREM 5.4. *Let H be a strongly connected embedded planar digraph without large-degree vertices. Let m be the size of H. Let β be the number of positive faces in H. A* CD-*pair of spanning trees for H can be computed in $O(\lceil\log(\beta+1)\rceil \log m)$ time using $O(m)$ processors.*

At the top level, a CD-pair of spanning trees for H is built in three stages. The first stage uses vertex contraction and vertex expansion to recursively simplify H. The simplification operates in phases. Each simplification phase converts H into a new version H' that is still a strongly connected embedded planar digraph without large-degree vertices. More importantly, H' has the following two properties. (1) The number of positive faces of H' is at most half that of H; this face property states the progress made by a phase. (2) The number of edges and vertices of H' is at most that of H; this size property is essential for achieving a linear-processor complexity for the planar directed spanning tree algorithm. Because H has more than one vertex and is strongly connected, from Theorem 2.3, H has at least one positive cluster and the simplification can start. In $\lfloor\log\beta\rfloor$ simplification phases, H is converted to H'' such that (1) H'' has exactly one positive face and no large-degree vertices and (2) the number of edges and vertices of H'' is at most that of H. Once H'' is obtained, the second stage uses the edge cutting technique to compute a CD-pair Υ'' of spanning trees for H''. The third stage converts Υ'' back to a CD-pair of spanning trees for H. The construction employs the duplicate removal technique and the tree rerooting technique to undo the contractions and expansions used in the simplification phases. The construction also operates in phases and the construction phases correspond to the simplification phases in the reverse order. More precisely, a construction phase converts a CD-pair Υ' of spanning trees for H' back to a CD-pair Υ of spanning trees for H.

As for the complexity of the first stage, the size property of H' ensures that while H keeps changing in the stage, its size never exceeds the original size at any time. Thus, for this stage to achieve the complexity stated in Theorem 5.4, it suffices that each simplification phase takes only logarithmic time and linear processors in the size of the input graph to the phase. As for the second stage, from the face and size properties of H'' and by Theorem 3.6, this stage stage takes only logarithmic time and linear processors in the size of H. As for the third stage, this stage is symmetric to the first stage. Therefore, for this stage to achieve the complexity stated in Theorem 5.4, it suffices that a construction phase takes only logarithmic time and linear processors in the size of the input graph to the phase. In sum, if a simplification phase and a construction phase each take only logarithmic time and linear processors in the size of their input graphs, then the total complexity for computing a CD-pair of spanning trees for H is $O(\lceil\log(\beta+1)\rceil \log m)$ using $O(m)$ processors.

To finish the proof of Theorem 5.4, it suffices to detail a simplification phase and a construction phase as follows.

A simplification phase consists of the following four steps. (1) Let H_1 be constructed from H by contracting each biconnected territory into a single vertex. (2) Let H_2 be constructed from H_1 by contracting each positive cluster into a single vertex. (3) Let H_3 be constructed from H_2 by expanding into a *positive* face each contracted positive cluster of H_2. (4) Let H' be constructed from H_3 by expanding each large-degree vertex into a *negative* face. Notice that the last three steps are exactly the same as the steps in the reduction from G to H. Also, H is a strongly connected embedded planar digraph and has no large-degree vertices. The following lemmas show that H' satisfies the face and size properties and can be computed in logarithmic time using linear processors in the size of H.

LEMMA 5.5. *The number of positive faces of H' is at most half that of H.*

Proof. By Lemma 5.2, the number of positive faces in H' is equal to the number of positive clusters in H_1. By Theorem 3.9, the number of positive clusters in H_1 is at most half the number of positive faces in H. In sum, the number of positive faces of H' is at most half that of H. □

LEMMA 5.6. $|H_2| \leq |H_1| \leq |H|$ *and* $|H_2| \leq |H_3| \leq |H'| \leq |H|$.

Proof. Because vertex contraction cannot increase the size, $|H_2| \leq |H_1| \leq |H|$ is true. Because vertex expansion cannot decrease the size, $|H_2| \leq |H_3| \leq |H'|$ is also true. The remaining inequality $|H'| \leq |H|$ is shown as follows. A vertex in H_2 can be expanded at most once either in the third step or in the fourth step. Thus, H' is in effect expanded from H_2. On the other hand, H_2 is in effect contracted from H. Let u be a vertex of H_2 that is expanded in H'. Let U be the connected vertex subset of H that is contracted into u. The next step of proof is to establish that U is biconnected in H. There are two cases based on which step expands u. Case 1. If u is expanded in the third step, then because contracting the territories destroys all positive faces of H, the vertex u is contracted from a positive cluster of H_1 that contains contracted biconnected territories of H. From the small degree of H, the set U is biconnected in H. Case 2. If u is expanded in the fourth step, then because H has no large-degree vertex, U is a biconnected territory of H and thus is biconnected. The last step of proof is to use the biconnectivity to estimate the size change from H to H'. Let D_b and D_o be the sets of edges in H with, respectively, both ends and only one end in U. From the small degree of H and the biconnectivity of U, the inequalities $|D_o| \leq |D_b|$ and $|D_o| \leq |U|$ can be shown by examining the boundary of the subgraph induced by U. These two inequalities are now used to estimate the size change from H to H' concerning U. The contraction of U loses U and D_b, gains a super vertex, and keeps D_o. Thus, in the contraction stage the net loss is $|U| + |D_b| - 1$. The expansion of u keeps D_o, loses u, and gains $|D_o|$ new edges and $|D_o|$ new vertices. Thus, in the expansion stage, the net gain is $|D_o| + |D_o| - 1$. Thus, from in the conversion from H to H', the net gain concerning U is $|D_o| + |D_o| - |U| - |D_b|$, which is less than or equal to zeo. Consequently, $|H'| \leq |H|$. □

LEMMA 5.7. *H' can be constructed in $O(\log m)$ using $O(m)$ processors.*

Proof. In a simplification phase, the complexity for the first step follows Theorem 3.7. The complexity for the other three steps follows Lemma 5.3 and the inequality $|H_1| \leq |H|$. □

A construction phase iteratively converts Υ' into Υ in the following directions: $H' \rightarrow H_3$, $H_3 \rightarrow H_2$, $H_2 \rightarrow H_1$, $H_1 \rightarrow H$. The construction for the first three directions is as described in the tree construction from H back to G. The construction for $H_1 \rightarrow H$ follows Theorem 4.6. The next lemma gives the complexity of a construction phase and thus provides the last piece in the proof Theorem 5.4.

LEMMA 5.8. *The construction of Υ from Υ runs in $O(\log m)$ time and $O(m)$.*

Proof. In a construction phase, the complexity for the first three directions follows Lemma 5.3 and the inequality $|H_1| \leq |H|$. The complexity of the fourth direction follows Theorem 4.6. □

6. Open problems. This paper has shown that for a strongly connected embedded planar digraph with n vertices, a directed spanning tree rooted at a specified vertex can be computed in $O(\log^2 n)$ time using $O(n)$ processors. This result complements the linear-processor NC algorithm by Kao for computing the strongly connected components of a planar digraph [10]. There are several fundamental problems left open in this paper. Probably the most important one is to compute planar breadth-first search efficiently. Currently the best algorithm for this problem is by Pan and Reif [15]. Their algorithm computes the shortest paths in $O(\log^2 n)$ using $O(n^{1.5}/\log n)$ processors. The algorithm is based on matrix operations and uses an almost optimal randomized algorithm for planar separators by Gazit and Miller [7]. It would be of significant impact to reduce to linear the processor complexity of planar breadth-first search.

Acknowledgements. The authors thank Fang Wan for helping prepare the figures. The first author gratefully acknowledges Phil Klein, Gary Miller, and Vijaya Ramachandran for many helpful discussions and suggestions.

REFERENCES

[1] A. AHO, J. HOPCROFT, AND J. ULLMAN, *The Design and Analysis of Computer Algorithms*, Addison-Wesley, Reading, MA, 1974.

[2] R. J. ANDERSON AND G. L. MILLER, *Deterministic parallel list ranking*, in Proc. 3rd Aegean Workshop on Computing, Corfu, Greece, J. H. Reif, ed., Lecture Notes in Computer Science 319, Springer-Verlag, Berlin, New York, 1988.

[3] M. ATALLAH AND U. VISHKIN, *Finding euler tours in parallel*, J. Comput. System Sci., 29 (1984), pp. 330–337.

[4] J. BONDY AND U. MURTY, *Graph Theory with Applications*, North-Holland, 1976.

[5] R. COLE AND U. VISHKIN, *Optimal parallel algorithms for expression tree evaluation and list ranking*, in Proc. 3rd Aegean Workshop on Computing, Corfu, Greece, J. H. Reif, ed., Lecture Notes in Computer Science 319, Springer-Verlag, Berlin, New York, 1988.

[6] D. COPPERSMITH AND S. WINOGRAD, *Matrix multiplication via arithmetic progressions*, in Proc. 19th Annual ACM Symposium on Theory of Computing, Association for Computing Machinery, New York, 1987, pp. 1–6.

[7] H. GAZIT AND G. L. MILLER, *A parallel algorithm for finding a separator in planar graphs*, in Proc. 28th Annual IEEE Symposium on Foundations of Computer Science, 1987, pp. 238–248.

[8] ———, *An improved parallel algorithm that computes the* BFS *numbering of a directed graph*, Inform. Process. Lett., 28 (1988), pp. 61–65.

[9] F. HARARY, *Graph Theory*, Addison-Wesley, 1969.

[10] M. Y. KAO, *Linear-processor* NC *algorithms for planar directed graphs* I: *strongly connected components*. Submitted for publication, 1989.

[11] R. KARP AND V. RAMACHANDRAN, *A survey of parallel algorithms for shared-memory machines*, Tech. Rep. UCB/CSD 88/408, Computer Science Division, EECS, University of California at Berkeley, Mar. 1988. To appear in the *Handbook of Theoretical Computer Science* by North-Holland.

[12] P. N. KLEIN AND J. H. REIF, *An efficient parallel algorithm for planarity*, in Proc. 27th Annual IEEE Symposium on Foundations of Computer Science, 1986, pp. 465–477.

[13] S. R. KOSARAJU AND A. L. DELCHER, *Optimal parallel evaluation of tree-structured computations by raking*, in Proc. 3rd Aegean Workshop on Computing, Corfu, Greece, J. H. Reif, ed., Lecture Notes in Computer Science 319, Springer-Verlag, Berlin, New York, 1988.

[14] G. MILLER AND J. REIF, *Parallel tree contractions and its applications*, in Proc. 17th Annual ACM Symposium on Theory of Computing, 1985, pp. 478–489.

[15] V. PAN AND J. H. REIF, *Fast and efficient solution of path algebra problems*, Tech. Rep. 3, Computer Science Department, State University of New York at Albany, 1987.

[16] V. RAMACHANDRAN AND J. H. REIF, *An optimal parallel algorithm for graph planarity*, in Proc. 30th Annual IEEE Symposium on Foundations of Computer Science, 1989, pp. 282–287.

[17] Y. SHILOACH AND U. VISHKIN, *An* $O(\log n)$ *parallel connectivity algorithm*, Journal of Algorithms, 3 (1982), pp. 57–67.

[18] R. TARJAN AND U. VISHKIN, *An efficient parallel biconnectivity algorithm*, SIAM J. Comput., 14 (1985), pp. 862–874.