An Efficient Parallel Algorithm for Planarity

PHILIP N. KLEIN*

Laboratory for Computer Science, MIT, Cambridge, Massachusetts

AND

JOHN H. REIF†

Computer Science Department, Duke University, Durham, North Carolina

We describe a parallel algorithm for testing a graph for planarity, and for finding an embedding of a planar graph. For a graph on n vertices, the algorithm runs in $O(\log^2 n)$ steps on n processors of a parallel RAM. The previous best parallel algorithm for planarity testing also ran in $O(\log^2 n)$ time (J. Ja'Ja' and J. Simon, *J Comput.* **11**, No. 2 (1982), 313–328), but used a reduction to solving linear systems, and hence required $\Omega(M(n)/\log^2 n)$ processors, where $M(n)$ is the sequential time for $n \times n$ matrix multiplication, whereas our processor bounds are within a polylog factor of optimal. The most significant aspect of our parallel algorithms is the use of a sophisticated data structure for representing sets of embeddings, the *PQ*-tree of K. Booth and G. Lueker, *J. Comput. System Sci.* **13**, No. 3 (1976), 335–379). Previously no parallel algorithms for *PQ*-trees were known. We have efficient parallel algorithms for manipulating *PQ*-trees, which we use in our planarity algorithm.

1. INTRODUCTION

The study of planar graphs dates back to Euler. A drawing of a graph on a plane in which no edges cross is called a *planar embedding*. A graph for which such an ambedding exists is called a *planar* graph. The search for an efficient algorithm to decide planarity and find a planar embedding culminated in Hopcroft and Tarjan's linear-time algorithm [8].

Continuing in this tradition, we have developed an efficient *parallel* algorithm for this problem. Our new algorithm finds a planar embedding for an n-node graph (or reports that none exists) in $O(\log^2 n)$ time using only n processors of an exclusive-write, concurrent-read P-RAM [7].[1] Thus it achieves near-optimal speedup.

* Research supported by an ONR Graduate Fellowship.

† Research supported by Office of Naval Research Contract N00014-80-C-0647 and National Science Foundation Grant DCR-85-03251.

An extended abstract of this paper appeared in the "Proceedings, 27th Annual IEEE Symposium on Foundations of Computer Science, 1986, pp. 465–477.

[1] We assume this model through this paper.

Preprint of paper appearing in Journal of Computer and System Sciences, Vol. 37, No. 2, October 1988, pp. 190-246.

In contrast, the previous best parallel algorithm for testing planarity, due to Ja'Ja' and Simon [9], reduced the problem to solving linear systems, and hence required at least $M(n)$ total operations (time $\times$ number of processors), where $M(n)$ is the number of operations required to multiply two $n \times n$ matrices. Ja'Ja' and Simon's algorithm was important because it showed that planarity could be decided quickly in parallel. However, such a large processor bound makes their algorithm infeasible. Moreover, their algorithm found a planar embedding for triconnected graphs but not for arbitrary graphs. In [17], Miller and Reif showed how embeddings found by Ja'Ja' and Simon's algorithm could be combined to find an embedding for an arbitrary graph. However, the processor bound for Miller and Reif's algorithm was no better than that of Ja'Ja' and Simon's.

The inspiration for our parallel algorithm is an efficient *sequential* algorithm resulting from the combined work of Lempel, Even, and Cederbaum [13], Even and Tarjan [6], and Booth and Lueker [3]. One essential ingredient we use from the work of Lempel, Even, and Cederbaum is that in building an embedding for a graph, premature commitment to a particular embedding of a subgraph should be avoided. Instead, we use a data structure called a *PQ*-tree, due to Booth and Lueker, to represent *all* embeddings of each subgraph. We introduce some new operations for the parallel manipulation of *PQ*-trees and use the parallel tree contraction technique of [17] to help implement these operations.

Our parallel algorithm differs significantly from the sequential algorithm that inspired it. The sequential algorithm extended an embedding node by node. In contrast, we use a divide-and-conquer strategy, computing embeddings for subgraphs and combining them to form embeddings of larger subgraphs. To handle the numerous complications that arise in carrying out this approach, we are forced to generalize the approach of Lempel, Even, and Cederbaum.

Our parallel planarity algorithm is rare among parallel algorithms in that it uses a sophisticated data structure. We have parallelized the *PQ-tree* data structure, due to Booth and Lueker [3], giving efficient parallel algorithms for manipulating *PQ*-trees. No parallel algorithms for *PQ*-trees existed previously. We define three operations on *PQ*-trees, *multiple-disjoint-reduction*, *join*, and *intersection*, and give linear-processor parallel algorithms for these operations. We use *PQ*-trees for representing sets of graph embeddings.

However, *PQ*-trees are generally useful for representing large sets of orderings subject to adjacency constraints. Booth and Lueker use *PQ*-trees in efficient sequential algorithms for recognizing sparse (0, 1)-matrices with the consecutive one's property, and in recognizing and testing isomorphism of interval graphs. Using our parallel algorithms for *PQ*-trees, one can recognize $n \times n$ (0, 1)-matrices with the consecutive one's property in $O(\log^3 n)$ time using n^2 processors.

In Section 2, we discuss the *PQ*-tree data structure. In Subsection 2.1, we give definitions of *PQ*-trees and the new operations on them: *multiple disjoint reduction*, *intersection*, and *join*. We show how these operations may be implemented in parallel in Subsections 2.2, 2.3, and 2.5. In Subsection 2.4, we prove some lemmas concerning the use of *PQ*-trees for representing sets of cycles. In Section 3, we

discuss the problem of planarity. In Subsection 3.1, we give some definitions and results concerning embeddings of graphs. In Subsection 3.2, we describe our parallel planarity algorithm.

2. Parallel *PQ*-Tree Algorithms

2.1. *PQ-Tree Definitions*

In our planarity algorithm, we will need to represent large sets of sequences of sets of edges. These sets are too large to represent explicitly, so we make use of an efficient data structure, the *PQ*-tree, due to Booth and Lueker [3]. In this section, we define the *PQ*-tree and some operations on it, and show how these operations may be carried out effciently in parallel. We freely adapt the terminology of [3] to suit our needs.

A *PQ*-tree over the ground set S is a tree with two kinds of internal nodes, *P*-nodes and *Q*-nodes. Every internal node has at least two children, so the number of internal nodes is no more than the number of leaves. The children of each internal node are ordered. The leaves are just the elements of S. For example, in Fig. 1 is depicted a *PQ*-tree T over the ground set $\{a, b, c, d, e, f\}$. Here, as henceforth, *Q*-nodes are depicted by rectangles and *P*-nodes are depicted by circles. Throughout this section, n will denote the cardinality of S.

For concreteness, we will assume that a *PQ*-tree is represented by a pointer structure as follows: each node has a pointer to its parent, its left sibling, its right sibling, its leftmost child, and its rightmost child (using null pointer where necessary). This representation permits constant time insertion and deletion of consecutive sequences of children by a single processor, and also $O(\log n)$ time tree contraction [17] on an exclusive-write *P*-RAM.

A *PQ*-tree is used to represent certain classes of linear orderings of its ground set S. Let T be a *PQ*-tree over S. We will denote by $L(T)$ the set of linear orders represented by T, and say that T *generates* $L(T)$. One element of $L(T)$ is obtained by reading off the leaves left to right in the order in which they appear in T. This is called the *frontier* of T, and written $\mathrm{fr}(T)$. (The frontier of the tree in Fig. 1 is *bafdce*.) The other elements are those linear orders obtained in the same way from

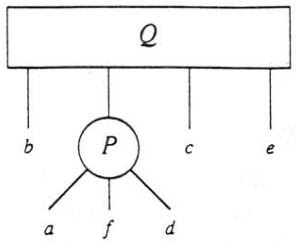

Fig. 1. A *PQ*-tree over the ground set $\{a, b, c, d, e, f\}$.

all trees T' *equivalent* to T. We say T' is *equivalent* to T if T' can be transformed into T by a sequence of transformations. The permissible equivalence transformations are:

(1) the order of children of a Q-node may be reversed (we say the Q-node is *flipped*), and

(2) the children of a P-node may be arbitrarily reordered.

It is useful to think of PQ-tree T as a representative of all the trees equivalent to it. We write $T' \cong T$ if T' is equivalent to T.

We shall occasionally speak of "flipping" all the nodes of a tree For this purpose, "flipping" a P-node means reversing the order of its children, which is certainly a permissable transformation. For a leaf, "flipping" has no effect.

Consider once again the PQ-tree T in Fig.1. There are 12 orderings in $L(T)$, including *bafdce*, *bdafce*, *ecafdb*, *ecfadb*. The first is just the frontier of T. The second ordering, *bdafce*, is obtained by reordering the P-node's children; the third ordering, *ecafdb*, is obtained by reversing the order of the Q-node's children; the fourth ordering, *ecfadb*, is obtained by both reversing the order of the Q-node's children and reordering the P-node's children.

Since there is no way a PQ-tree over a non-empty ground set can represent the empty set of orderings, we use a special *null* tree, denoted by T_{null}, to represent this empty set.

DEFINITION 2.1. If v is any node of some PQ-tree T, the subtree rooted at v is itself a PQ-tree, whose ground set is the set $\text{leaves}_T(v)$ of leaves below v. The *frontier* of v (in T) is just the frontier of the subtree of T rooted at v, and is written $\text{fr}_T(v)$. We write $\text{leaves}(v)$ for $\text{leaves}_T(v)$ and $\text{fr}(v)$ for $\text{fr}_T(v)$ when the choice of T is clear.

Note. Throughout this section, we use the terms *descendent* and *ancestor* to refer to non-proper descendents and ancestors, unless otherwise specified. That is, v is considered its own descendent and its own ancestor. We use the term *proper descendent* to refer to a descendent other than v. An *endpoint* of a linear ordering is either a leftmost (first) or a rightmost (last) element. The parent of v is denoted by $p(v)$.

DEFINITION 2.2. Let A be a subset of the ground set S. We say a linear ordering $\lambda = s_1 \cdots s_n$ of S *satisfies* the set A if all the elements of A are consecutive in λ; i.e., for some i and j, $s_i s_{i+1} \cdots s_j$ are all the elements of A. For a PQ-tree T, let $\Psi(T, A) = \{\lambda \in L(T)\colon \lambda \text{ satisfies } A\}$.

For example, if T is the PQ-tree in Fig.1, then $\Psi(T, \{a, c, f\}) = \{bdafce, bdface, ecfadb, ecafdb\}$. Booth and Lueker prove that given any T and $A \subseteq S$, there is a PQ-tree $\hat{T}$ such that $L(\hat{T}) = \Psi(T, A)$, called the *reduction* of T with respect to A. In fact, they give an algorithm REDUCE(T, A) which modifies T to get $\hat{T}$. Their

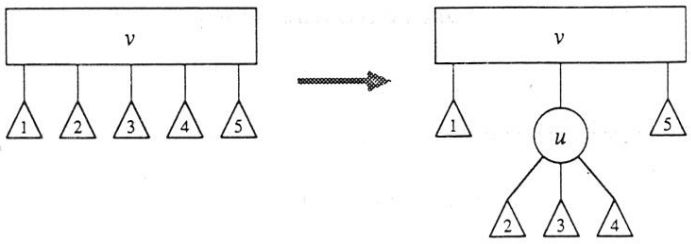

FIG. 2. Inserting the node u between v and children 2, 3, 4.

algorithm works in time proportional to the cardinality of A. Note that if *no* ordering generated by T satisfies A, the reduction $\hat{T}$ is just the null tree.

For applications to parallel algorithms, it is useful to be able to reduce a PQ-tree with respect to many disjoint sets $A_1, ..., A_k$ simultaneously. We let $\Psi(T, \{A_1, ..., A_k\}) = \{\lambda \in L(T): \lambda \text{ satisfies each } A_i \ (i = 1, ..., k)\}$.

In Subsection 2.2, we give a parallel algorithm for "multiple" reduction, MREDUCE$(T, \{A_1, ..., A_k\}$ which modifies T to obtain a PQ-tree $\hat{T}$ such that $L(\hat{T}) = \Psi(T, \{A_1, ..., A_k\})$ if the A_i's are disjoint. (Any ordering automatically satisfies a singleton set.)

THEOREM 2.1. *MREDUCE can be computed in* $O(\log n)$ *time using n processors.*

Next we make some observations and introduce some terminology useful to the algorithms in this section.

DEFINITION 2.3. Suppose that a node v of a PQ-tree has children $v_1 \cdots v_s$ in order. To "insert" a node u between v and a consecutive subsequence $v_p \cdots v_q$ of its children is to make u the pth child of v, and let the children of u be $v_p \cdots v_q$ in order. Note that the operation of insertion does not change the frontier of any node of the PQ-tree.

In Fig. 2, we show a P-node being inserted between a Q-node v and its children 2, 3, 4. (In this figure and others to come, we use a triangle to represent a subtree; if the triangle is numbered, we use the number to refer to the root of the subtree.)

If each ordering $\lambda \in L(T)$ satisfies A, the part of the tree "pertinent" to the set A is "contiguous," in a sense described below.[2]

DEFINITION 2.4. Let $\text{lca}_T(A)$ denote the least common ancestor of the leaves belonging to A. Suppose that $v = \text{lca}_T(A)$ has children $v_1 \cdots v_s$ in order. We say A is *contiguous* in T if

[2] In [3], Booth and Lueker referred to this part of the tree as the *pertinent* subtree of a PQ-tree.

- v is a Q-node, and for some consecutive subsequence $v_p \cdots v_q$ of the children of v, $A = \bigcup_{p \leq i \leq q} \text{leaves}(v_i)$, or
- v is a P-node or a leaf, and $A = \text{leaves}(v)$.

For example, the sets $\{a, b, d, f\}$ and $\{a, d, f\}$ are both continguous in the PQ-tree of Fig. 1. Note that if A is contiguous in T, then A is contiguous in any tree T equivalent to T. In [14a] is proved essentially the following:

LEMMA 2.1. *Suppose that A is a non-empty subset of the ground set of T. Then A is contiguous in T iff each ordering $\lambda \in L(T)$ satisfies A.*

Proof. ($\Rightarrow$) Suppose A is contiguous in T. Clearly $\lambda_v = \text{fr}_T(v)$ satisfies A, because $\lambda_v = \lambda_1 \cdots \lambda_s$, where $\lambda_i = \text{fr}_T(v_i)$, and $\lambda_p \cdots \lambda_q$ consists exactly of all the elements of A. But λ_v is a consecutive subsequence of $\text{fr}(T)$, so $\text{fr}(T)$ also satisfies A. Using the fact that A is contiguous in every $T' \cong T$, we conclude that every $\lambda \in L(T)$ satisfies A.

($\Leftarrow$) Assume every $\lambda \in L(T)$ satisfies A. If only one child v_i of $v = \text{lca}_T(A)$ satisfies the condition

$$\text{leaves}(v_i) \cap A \neq \varnothing \tag{1}$$

then $\text{lca}(A)$ is a descendent of v_i, contrary to choice of v. Thus v has more than one child v_i satisfying condition (1). Let the leftmost such child of v be v_p and the rightmost v_q. Suppose that for some child v_i $(p \leqslant i \leqslant q)$, we did not have leaves $(v_i) \subset A$. Then either the frontier of T does not satisfy the subset A, or the frontier of the tree obtained from T by flipping v_i does not satisfy A (or both). Hence we have $\text{leaves}(v_i) \subset A$ for each $p \leqslant i \leqslant q$. But then $A = \bigcup_p^q \text{leaves}(v_i)$ because v was chosen to be an ancestor of every leaf in A. If v is a Q-node, this completes the proof that A is contiguous in T.

Suppose that v is a P-node, and v had some child v_i such that $\text{leaves}(v_i) \not\subset A$. By reordering the children of v so that v_i is between v_p and v_q, we obtain a tree whose frontier does not satisfy A. Thus for every child v_i, we have $\text{leaves}(v_i) \subset A$; i.e., $\text{leaves}(v) = A$. ∎

COROLLARY 2.1. *Suppose the following non-empty sets are contiguous in T: A, B, C, $A \cup B$, $B \cup C$. Then $A \cup B \cup C$ is contiguous in T.*

We next define a new operation on PQ-trees, not considered in [3]. A PQ-tree $\hat{T}$ is the *intersection* of two PQ-trees T and T' over the same ground set if $L(\hat{T}) = L(T) \cap L(T')$. In Subsection 2.3, we describe an algorithm INTERSECT(T, T') for computing the intersection of two PQ-trees using disjoint reduction as a subroutine.

THEOREM 2.2. *INTERSECT can be computed in $O(\log^2 n)$ time using n processors.*

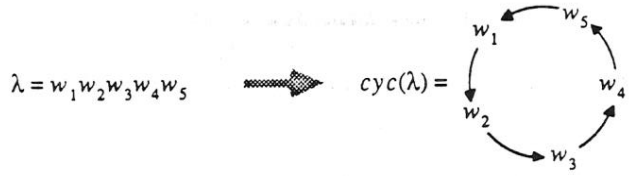

FIG. 3. Obtaining a cycle cyc(λ) from a linear ordering λ.

Note that, like reduction, intersection can "fail," i.e., the result may be the null tree.

Remark. As an illustration of the usefulness of INTERSECT for parallel algorithms, we can use it to obtain a parallel algorithm for reducing a *PQ*-tree with respect to a sequence $A_1, ..., A_k$ of subsets that are not necessarily disjoint. The algorithm works in $O(\log^2 n \cdot \log k)$ time using $O(kn)$ processors.‡ Make k copies $T_1, ..., T_k$ of the *PQ*-tree T, and in parallel reduce each T_i with respect to A_i. Pair up the T_i's and intersect the pairs, obtaining $\lceil k/2 \rceil$ intersected trees. Iterate, pairing up the intersected trees and intersecting the pairs, until there remains only one tree. Each iteration reduces the number of trees by a constant factor, so there are $O(\log k)$ iterations, each of which takes $O(\log^2 n)$ time (to do the intersection).

We can use this algorithm to determine whether a (0, 1)-matrix has the *consecutive one's property*: after some reordering of rows, the 1's in each column are consecutive. For an $n \times m$ matrix M, let the ground set S be the set of rows of M. For $i = 1, ..., m$, let $A_i =$ the set of rows in which there is a 1 in the ith column. Let T be the trivial *PQ*-tree generating all orderings over S. Then the reduction of T with respect to $A_1, ..., A_m$ generates those orderings of the rows making the 1's in each column consecutive. This is how Booth and Lueker recognized (0, 1)-matrices with the consecutive one's property sequentially in time $O(n+m+f)$, where $f = \sum_1^m |A_i|$ is the number of 1's in M. Our parallel algorithm takes time $O(\log^2 n \cdot \log m)$ using nm processors. Note that this application is not required for our planarity algorithm.

In our planarity algorithm, we will use *PQ*-trees to represent sets of *cycles*, rather than sets of linear orderings. We next discuss this representation.

DEFINITION 2.5. With each linear ordering λ we associate the cycle cyc(λ) obtained from λ by letting the first element of λ follow the last.[3] For example, Fig. 3 illustrates how a cycle cyc(λ) = $(w_1 \cdots w_5)$ is obtained from the linear ordering $\lambda = w_1 w_2 w_3 w_4 w_5$. The frontier fr($T$) of a *PQ*-tree T represents a cycle cyc(fr(T)); for readability, we define cycfr(T) = cyc(fr(T)). Then the *PQ*-tree T represents the set of cycles $CYC(T) = \text{cyc}(L(T)) = \{\text{cycfr}(T'): T' \cong T\}$.

‡ *Note added in proof.* The first author has recently discovered an improved algorithm for this problem, one that works in $O(\log n \cdot (\log(n+t))$ time using t processors, where $t = \sum_1^k |A_i|$. The algorithm will be described in the first author's Ph.D. dissertation.

[3] Throughout this paper we use the term *cycle* to refer to an *oriented* cycle, analogous to a directed cycle in a directed graph.

Note that our representation of a cycle by a linear ordering $\lambda = w_1 \cdots w_n$ involves considerable redundancy, as the same cycle is also represented by $w_i \cdots w_n w_1 \cdots w_{i-1}$, for $i = 2, ..., n$. We make use of this redundancy in obtaining the following lemma.

LEMMA 2.2. *Let T be a PQ-tree whose ground set is the disjoint union of nonempty sets A, B, C. Suppose that each of these sets is contiguous in T. Then T may be modified to be a PQ-tree T' in which $A \cup B$ and $B \cup C$ are contiguous, and such that* $\mathrm{CYC}(T') = \mathrm{CYC}(T)$.

See Fig. 4 for an example of this modification. The modification of Lemma 2.2 can be carried out easily (i.e., in $O(\log n)$ time using n processors, where n is the size of the ground set). We call this modification ROTATE(A, B, C); it is used in the planarity algorithm. The proof of Lemma 2.2 appears in Subsection 2.4.

DEFINITION 2.6. In analogy to our terminology for linear orderings, we say a cycle σ of S satisfies a subset $A \subset S$ if the elements of A form a consecutive subsequence of σ.

An analog of Lemma 2.1 does *not* hold for PQ-trees used to represent sets of cycles: even if every ordering in $\mathrm{CYC}(T)$ satisfies a set D, it does not follow that D is contiguous in T. For example, consider the PQ-tree T' of Lemma 2.2, as depicted in Fig. 4. While every ordering in $\mathrm{CYC}(T)$ satisfies $A \cup C$, in fact T' is *not* $(A \cup C)$-contiguous. However, with the addition of an extra condition, an analog of Lemma 2.1 holds.

LEMMA 2.3. *Suppose A is a proper subset of the ground set of T, and is contiguous in T. If $A_1 \subseteq A$, then for each $\lambda \in L(T)$, λ satisfies A_1 iff* $\mathrm{cyc}(\lambda)$ *satisfies A_1.*

We can use Lemmas 2.2 and 2.3 in conjunction to "reduce" a PQ-tree used to represent a set of cycles. Suppose we have a PQ-tree T representing a set of cycles, and we want to obtain a PQ-tree $\hat{T}$ such that $\mathrm{CYC}(\hat{T}) = \{\sigma \in \mathrm{CYC}(T)$: σ satisfies

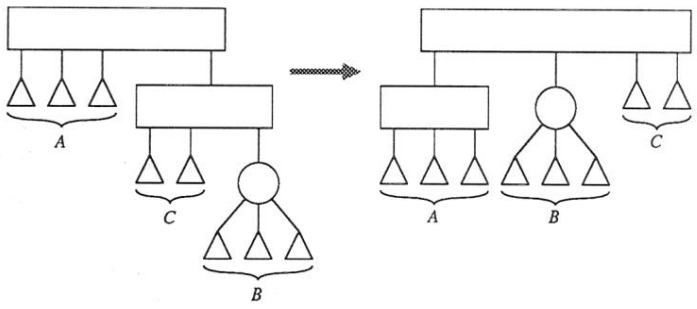

FIG. 4. Modifying a tree T in which A, B, and C are contiguous to get a tree T' in which $A \cup B$ and $B \cup C$ are contiguous.

$A_1, ..., A_k\}$; i.e., we want to reduce T with respect to disjoint sets $A_1, ..., A_k$. Sometimes (not always), we can apply Lemma 2.2 to modify T so that all the A_i's are contained in a set A that is contiguous in T. Then by Lemma 2.3, we need only let $\hat{T}$ be the PQ-tree computed by MREDUCE.

The proof of Lemma 2.3 appears in Subsection 2.4.

DEFINITION 2.7. For a cycle σ of S that satisfies a non-empty proper subset $A \subset S$, let $\sigma \mid A$ denote the consecutive subsequence of σ consisting of elements of A. Note that $\sigma \mid A$ is a *linear ordering*.

For example, if σ is the cyclic ordering $\text{cyc}(w_1 w_2 w_3 w_4 w_5)$ depicted in Fig. 3, then $\sigma \mid \{w_1, w_2, w_5\} = w_5 w_1 w_2$. This notation extends elementwise to sets and ordered pairs; e.g., $\langle \sigma, \sigma' \rangle \mid A$ is the same as $\langle \sigma \mid A, \sigma' \mid A \rangle$. Note that if the cycle σ of S satisfies A, it also satisfies $S - A$, so $\sigma \mid (S - A)$ is well-defined.

We next define another new operation on PQ-trees that corresponds to combining embeddings of subgraphs.[4] For a linear ordering λ, let λ^R denote the reverse of λ.

In the following, let S_0 and S_1 be ground sets whose intersection E is a non-empty proper subset of S_0 and of S_1, let σ_0 be a cycle of S_0, and let σ_1 be a cycle of S_1.

DEFINITION 2.8. If σ_0 and σ_1 satisfy E and $\sigma_0 \mid E = (\sigma_1 \mid E)^R$, we let σ_0 *join* σ_1 denote the cycle of $S_0 \cup S_1$ obtained from σ_0 by substituting $\sigma_1 \mid (S_1 - E)$ for $\sigma_0 \mid E$.

Figure 5 illustrates how the join operation works. As we shall see in Section 3.2, the join operation corresponds to the operation of contracting some edges between two nodes of an embedded graph, identifying the nodes. The operator *join* is left-associative.

DEFINITION 2.9. Let T_0 and T_1 be PQ-trees over the ground sets S_0 and S_1, and suppose $E = S_0 \cap S_1$ is contiguous in T_0 and in T_1, and $S_1 - E$ is also contiguous in T_1. We say T_+ is the *join* of T_0 with T_1 if

$$\text{CYC}(T_+) = \{\sigma_0 \text{ join } \sigma_1 : \sigma_0 \in \text{CYC}(T_0), \sigma_1 \in \text{CYC}(T_1), \text{ and } \sigma_0 \mid E_1 = (\sigma_1 \mid E_1)^R\}. \tag{2}$$

Note that $\text{CYC}(T_+)$ may be empty, even if $\text{CYC}(T_0)$ and $\text{CYC}(T_1)$ are non-empty, because of the last clause in the definition. The PQ-tree join corresponds roughly to combining embeddings of a pair of nodes.

In Subsection 2.5, we show how to compute the join T_+, using PQ-tree intersection as a subroutine. More specifically, we give two procedures, one for computing the "provisional" join, and the other for verifying the join. The *provisional-join* procedure constructs a PQ-tree T_+ such that T_+ satisfies (2) *unless* the right-hand side of (2) is empty (in which case T_{null} is the join of T_0 with T_1). The *verification*

[4] In their application of PQ-trees to planarity ([3]), Booth and Lueker carried out a tree-splicing operation that can be viewed as a special case of our new operation. They only needed a rudimentary version because in their algorithm an embedding was only extended one node at a time.

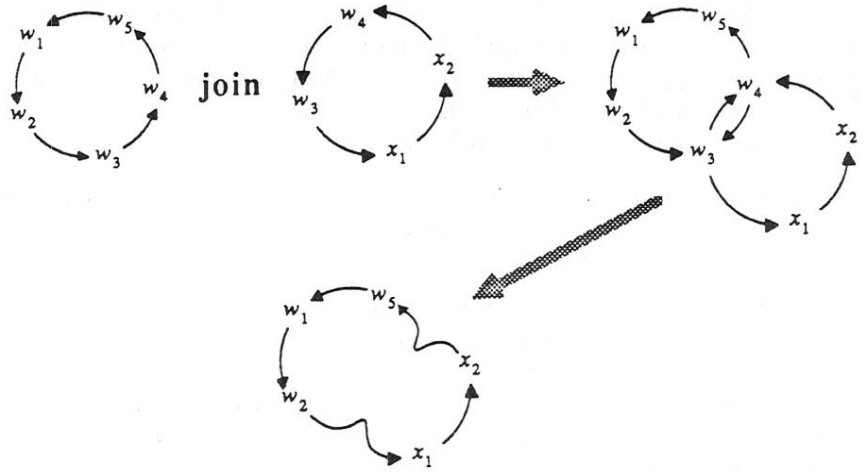

FIG. 5. Computing cyc($w_1 \cdots w_5$) join cyc($x_1 x_2 w_4 w_3$). First the cycle are matched up along the shared ground elements, then the shored elements are deleted and the remaining linear orders are spliced together.

procedure uses *PQ*-intersection to determine if the right-hand side of (2) is empty. In the planarity algorithm, we use the provisional join so as to quickly proceed to the next stage of the algorithm, and verify all joins simultaneously once the final stage is complete.

In fact, we can solve a slightly more general problem. It turns out that the algorithm for computing the join of T_0 with T_1 only needs access to the part of the tree T_0 that is "pertinent" to E in the sense of Lemma 2.1 (and in the sense of [3]). Hence it is possible to join T_0 with many other trees simultaneously, as long as these other trees have disjoint ground sets.

Let T_0 be a *PQ*-tree over the ground set S_0 as before, and let $T_1, ..., T_k$ be trees over disjoint ground sets $S_1, ..., S_k$, where $E_j = S_j \cap S_0$ is non-empty for $j = 1, ..., k$. As before, assume E_j is contiguous in T_0 and in T_j, and $S_j - E_j$ is contiguous in T_j, for all $j = 1, ..., k$. In this case, the join T_+ of T_0 with $T_1, ..., T_k$ is defined to be the tree obtained by first taking the join of T_0 with T_1, then taking the join of the result with T_2, and so on.

THEOREM 2.3. *The provisional join T_+ of T_0 with $T_1, ..., T_k$ can be computed in $O(\log n)$ time using n processors, where n is the total number of ground elements. The join can be verified in $O(\log^2 m)$ time using m processors, where m is the number $\sum_{j=1}^{k} |E_j|$ of common elements.*

Remark 2.1. The elements of CYC(T_+) have the form σ_0 join σ_1 join $\cdots$ join σ_k, where σ_j is a cycle of S_j ($\forall j$). An example appears in Fig. 6. Since $S_1, ..., S_k$ are all disjoint, the cycle σ_0 join $\cdots$ join σ_k can be obtained directly from σ_0 by substituting $\sigma_j \mid (S_j - E_j)$ for $\sigma_0 \mid E_j$. $j = 1, ..., k$. Thus this cycle does not depend on the order of

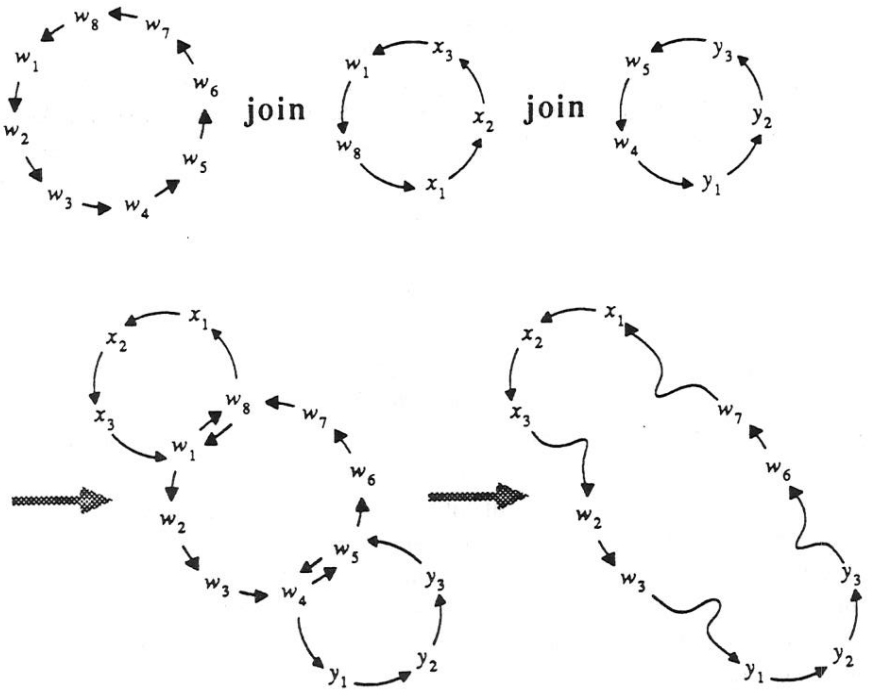

FIG. 6. Computing $\text{cyc}(w_1 \cdots w_8)$ join $\text{cyc}(w_1 w_8 x_1 x_2 w_3)$ join $\text{cyc}(w_5 \geqslant w_4 y_1 y_2 y_3)$.

$\sigma_1, ..., \sigma_k$; e.g., σ_0 join σ_k join $\cdots$ join σ_1 denotes the same cycle. It follows that the order of $T_1, ..., T_k$ is irrelevant to the final tree T_+ obtained.

Remark 2.2. In the algorithms of this section, we assume n-processor, $O(\log n)$ time algorithms for two problems:

- sorting n numbers, each consisting of $O(\log n)$ bits (called "small integer sorting"), and
- finding the lowest common ancestors of n pairs of nodes in an n-node tree.

Simple algorithms for these two problems appear in the Masters' thesis of the first author [10]. More sophisticated algorithms have appeared in the literature. For example, the second author has a randomized algorithm solving the first problem using only $n/\log n$ processors. Deterministic algorithms for the first problem follow from parallel comparison sorting algorithms; see [1, 12, 4]. The second problem may be solved using techniques of [19].

2.2. *Reduction*

In this subsection, we describe the algorithm MREDUCE and prove its correctness, proving Theorem 2.1.

For notational convenience, we summarize the disjoint sets $A_1, ..., A_k$ in a "coloring" $\delta(v)$ of the bound elements $v \in S$, where the elements of each A_i receive a single color. We say an ordering λ *satisfies* a color if all the elements of that color appearing in λ occur consecutively. (Not all need appear.) We let $\Psi(T, \delta) = \{\lambda: \lambda \in L(T), \lambda \text{ satisfies every color defined by } \delta\}$. Note that since any ordering automatically satisfies a singleton set, we can assume ever element of S receives a color—each element not appearing in any A_i receives its own unique color.

There are essentially four conditions that necessitate changes to the structure of a PQ-tree during reduction.

1. Condition: some (but not all) of the children of a P-node have descendents with the same color. In this case, a new P-node will be inserted between these selected children and their parent P-node, in order to ensure that these children remain consecutive under equivalence transformations.

2. Condition: some (but not all) of the c-colored ground elements are descendents of a node. In this case, it must be ensured (by changes to the tree) that these c-colored elements occur as an initial or final subsequence of an ordering generated by the subtree rooted at the node. Otherwise, if they are allowed to be internal, there is no hope that they will meet up with the other c-colored elements to form a consecutive sequence.

3. Condition: a P-node has children $u, v, w, ..., x$ (in some order) such that u and v have descendents with a common color, v and w have descendents with another common color, and so on. In this case, a Q-node must be inserted between this "chain" of children and their parent in order to ensure that the common colors are satisfied by any ordering generated.

4. Condition: a Q-node's leftmost child has descendents of some color c different from that of its rightmost child's descendents, and the Q-node's left sibling has descendents of the same color c. In this case, the Q-node must be prevented from flipping relative to its parent, in order to ensure that its c-colored descendents meet up with the c-colored descendents of its left sibling.

The algorithm consists of the following phases:

Pre-processing Phase. The coloring δ of the ground elements is extended to a "coloring" Δ of all the nodes of the PQ-tree T.

Phase A. P-nodes are processed and new nodes are inserted between each P-node and its children. The resulting PQ-tree is denoted by T_A.

Phase B. Q-nodes are processed: their children are assigned labels and then some are flipped in accordance with the labelling. The resulting PQ-tree is denoted by T_B.

Phase C. Certain sets of equivalence transformations are disallowed, by

changing some Q-nodes into special nodes called R-nodes, to be defined later. The resulting tree is T_C.

Post-processing Phase. Each of the R-nodes of the previous phase is eliminated and its children made children of its parent.

Note that phases A and B may "fail," in which case the reduced tree is just T_{null}.

The structure of the proof is as follows: We first prove that $L(T) \supseteq L(T_A) \supseteq \Psi(T, \delta)$. It follows that $\Psi(T_A, \delta) = \Psi(T, \delta)$. Since $T_B \cong T_A$, $L(T_B) = L(T_A)$. Finally, we prove that $L(T_C) = \Psi(T_B, \delta) = \Psi(T, \delta)$. Thus T_C generates exactly the desired orderings.

We first consider the pre-processing stage. This stage consists of extending the coloring δ of the leaves to obtain a coloring Δ of the entire tree. The following terminology will be used throughout the proof.

DEFINITION 2.10. For an internal node v of T, say a color is *complete* at v if all the leaves with that color are descendents of v. Say a color is *incomplete* at v if some, but not all, of the leaves of that color are descendents of v. Say that a color *covers* v if all the leaves below v are of that color, and that v is *uncovered* if no color covers v.

In general, the coloring of the ground elements imposes constraints on the ordering of the children of each internal node u. However, if a color is complete at a child u of v, that color does not constrain the ordering of v's children at all. The constraints arise because of colors incomplete at children of v. Therefore, the first step in extending the coloring is to compute for each internal node v the set $\text{INC}(v)$ of colors incomplete at v.

Note that for any $T' \cong T$ with a node v, the frontier of the subtree of T' rooted at v, denoted $\text{fr}_{T'}(v)$, is a consecutive subsequence of the frontier $\text{fr}(T')$ of T' (see Definition 2.1). If in addition $\text{fr}(T')$ satisfies the color $c \in \text{INC}(v)$, then the c-colored ground elements form a consecutive subsequence τ of $\text{fr}(T')$. Since $c \in \text{INC}(c)$, $\text{fr}(v)$ and τ overlap, but $\text{fr}(v)$ does not contain τ. These considerations yield the following lemmas:

LEMMA 2.4. *If* $\text{fr}(T')$ *satisfies a color* c *incomplete at* v *then at least one endpoint of* $\text{fr}_{T'}(v)$ *is colored* c.

LEMMA 2.5. *If* $\Psi(T, \delta) = \varnothing$ *then each node has at most two uncovered children at which the color* c *is incomplete.*

Proof of Lemma 2.5. If the c-colored elements form a consecutive subsequence τ of $\text{fr}(T')$ (where $T' \cong T$), then for each uncovered child u of v at which c is incomplete, $\text{fr}(u)$ contains an endpoint of τ. ∎

If $\Psi(T, \delta) \neq \varnothing$, then it follows from Lemma 2.4 $|\text{INC}(v)| \leqslant 2$ for all nodes v. We can therefore obtain, using tree contraction:

LEMMA 2.6. *If* $\Psi(T, \delta) \neq \varnothing$, INC$(\cdot)$ *can be computed for all nodes* v *simultaneously in* $O(\log n)$ *time using processors.*

Proof. By Lemma 2.4, we may assume that INC(v) never contains more than two elements. Consider the sequence of leaves of T, read left to right. For each color c, consider the first and the last node in this sequence that have color c. Their lowest common ancestor, which we will call LCA(c), is the lowest node in the tree at which the color c is complete. It is easy to compute lowest common ancestors for all colors simultaneously within the stated bounds (see Remark 2.2).

If v is a leaf colored c, INC(v) is either $\{c\}$, or, if v is the only node colored c, then INC(v) is empty. If v is an internal node, then

$$\text{INC}(v) = \left(\bigcup_{u \text{ is child of } v} \text{INC}(u) \right) - \{c\colon \text{LCA}(c) = v\}.$$

For an internal node v, INC(v) can be computed by determining the colors incomplete at the children of v, and checking each such color c to see if LCA(c) is v. By Lemma 2.4, each child can be assumed to contribute at most two colors. Moreover, the colors contributed by all children can be checked against LCA$(\cdot)$ simultaneously. Thus we have defined the problem of computing INC$(\cdot)$ as an expression evaluation problem, which can then be solved using the technique of parallel tree contraction. ∎

It at any node, the number of incomplete colors turns out to exceed two, the processor at that node should set a flag signifying failure. After the computation completes, it can be determined in $O(\log n)$ time whether any processor has set a failure flag. If so, the result of the reduction is T_{null}.

If the above computation succeeds, we can proceed with extending the coloring. The new coloring Δ will assign each node v a pair of colors

$$\Delta(v) = \langle c_1, c_2 \rangle$$

according to the following cases:

- If two colors are incomplete at v, then c_1 and c_2 are these colors.
- If only one color c is incomplete at v but c does *not* cover v, then $c_1 = c$ and c_2 is a new color c_v, unique to v.
- If one color c is incomplete at v and covers v, then $c_1 = c_2 = c$.
- If no colors are incomplete at v, $c_1 = c_2 = c_v$, the new color unique to v.

If the two colors c_1 and c_2 assigned to v are distinct (i.e., in the first two cases), we say that v is *orientable*. Note that no colors are incomplete at the root, so the root is not orientable.

DEFINITION 2.11. For a color c, let

$$h_v(c) = \begin{cases} c & \text{if } c \in \text{INC}(v) \\ c_v & \text{if } c \notin \text{INC}(v), \end{cases}$$

where c_v is the new color we associated with v in defining $\Delta(\cdot)$.

For a PQ-tree T', if w_1 and w_k are the leftmost and rightmost elements, respectively, of $\text{fr}_{T'}(v)$, let $l_{T'}[v] = h_v(\delta(w_1))$ and $r_{T'}[v] = h_v(\delta(w_k))$. Let $lr_{T'}[v] = \langle l_{T'}[v], r_{T'}[v] \rangle$. We leave out the subscript T' when the choice of PQ-tree is clear. We write $\langle a, b \rangle \sim \langle a', b' \rangle$ if $\{a, b\} = \{a', b'\}$. We use juxtaposition to stand for concatenation of tuples.

The following corollary follows from Lemma 2.4 and the definition of $\Delta(u)$.

COROLLARY 2.2. *If* $\text{fr}(T')$ *satisfies every color, then for every node* u,

$$lr_{T'}[u] \sim \Delta(u). \tag{3}$$

We now give a lemma that gives a local characterization of color satisfaction.

LEMMA 2.7. *Assume* (3) *holds for every node* u *of* T'. *For each node* u, *the following two conditions are equivalent*:

(a) *The frontier of* u *in* T' *satisfies every color.*

(b) *For each (not necessarily proper) descendent* v *of* u, *if* $v_1 \cdots v_s$ *are the children of* v *in order, then every color in* $lr[v_1]\, lr[v_2] \cdots lr[v_s]$ *occurs consecutively.*

Proof. By induction on height of u; trivial for $u = a$ leaf.

(a) $\Rightarrow$ (b) Let w_i and x_i be the leftmost and rightmost elements, respectively, of the frontier of chld v_i. Since $w_1 x_1 \cdots w_s x_s$ is a subsequence of the frontier of u, if the frontier of u satisfies every color, then certainly so does $w_1 x_1 \cdots w_s x_s$. Hence every color occurring in $\delta(w_1)\, \delta(x_1) \cdots \delta(w_s)\, \delta(x_s)$ occurs consecutively. Suppose $c = (w_i) \notin \text{INC}(v_i)$. Then c must be *complete* at v_i, so c is not the color of any element of the frontier of any other child v_j of v. It follows that every color occurring in $\delta(w_1)\, \sigma(x_1) \cdots c_{v_i} \delta(x_i) \cdots \delta(w_s)\, \delta(x_s)$ occurs consecutively, where we have simply replaced $\delta(w_i)$ with c_{v_i}. Continuing in this way, we obtain condition (b).

(b) $\Rightarrow$ (a) Suppose condition (b) holds, and let the frontier of u be $w_1 \cdots w_t$. Suppose for a contradiction that condition (a) does hot hold. A counterexample to condition (a) would be a consecutive subsequence $w_i \cdots w_j \cdots w_k$ of the frontier of u such that $\delta(w_i) = \delta(w_k) \neq \delta(w_j)$. Choose such a counterexample of smallest length. If $w_i \cdots w_k$ is contained within the frontier of some child of u, that frontier does not satisfy $\delta(w_i)$, so apply the inductive hypothesis to obtain a contradiction. Thus w_i and w_k must belong to the frontiers of distinct children $u_{i'}$ and $u_{k'}$ of u $(i' < k')$. Hence the color $c = \delta(w_i)$ is incomplete at $u_{i'}$ and $u_{k'}$, so, by (3), $c \in lr[u_{i'}]$ and $c \in lr[u_{k'}]$. By condition (b), since $i' < k'$, $r[u_{i'}] = l[u_{k'}] = c$.

Let $u_{j'}$ be the child of u to whose frontier w_j belongs, so $i' \leqslant j' \leqslant k'$. Since $i' < k'$,

either $i' < j'$ or $j' < k'$. In former case, condition (b) implies that $l[u_{j'}] = c$. It follows that if w_t is the leftmost element of the frontier of $u_{j'}$, then $w_t \cdots w_j$ is a smaller counterexample to condition (a) than $w_i \cdots w_k$, contradicting the choice of $w_i \cdots w_k$. The case $j' < k'$ is analogous. ∎

The following corollary follows from Eq. (3) and condition (b).

COROLLARY 2.3. *Suppose the frontier of $T' \cong T$ satisfies every color. Then for each node v and each color c, the children of v at which c is incomplete form a consecutive subsequence $v_j \cdots v_k$, where $v_{j+1} \cdots v_{k-1}$ are all covered by c.*

Proof. By Corollary 2.2 and Lemma 2.7, condition (b) holds for T'. Let v be a node of T' with children $v_1 \cdots v_s$, and let c be a color. By (3), any child v_i at which c is incomplete has $c \in lr[v_i]$. By condition (b), the children v_i with color $c \in lr[v_i]$ form a consecutive subsequence $v_j \cdots v_k$. Moreover, a child v_i such that $l[v_i] \neq r[v_i]$ must be an endpoint, i.e., either v_j or v_k. For suppose $j < i < k$ and, say, $l[v_i] = c$ but $r[v_i] \neq c$. Then $r[v_i]$ lies between two occurrences of the color c in $lr[v_1] \cdots lr[v_s]$, contradicting condition (b). We see that, for $j < i < k$, v_i must have $l[v_i] = r[v_i] = c$. But then v_i must be covered by c, or else the frontier of v_i would fail to satisfy the color c. ∎

Phase A is shown below. Figure 7 illustrates Phase A being applied to a single P-node.

PHASE A. For each P-node v:

A1 Reorder the children of v so that for each color c, all children covered by c are consecutive.

A2 For each color c, if there are at least two children covered by c (and at least one child not covered by c), insert a new P-node w_c between these c-covered children and v.

A3 At most two colors are incomplete at each child v_i. For each color c, find the set A_c of children at which c is incomplete. There is at most one child covered by c. If there are more than two uncovered children in A_c, set a flag signifying FAILURE. Otherwise, form the degree-2 graph G_v whose nodes are the children of v, where there is an edge between v_i and v_j if

- there is some color c common to $\mathrm{INC}(v_i)$ and $\mathrm{INC}(v_j)$, and
- either one node, say v_j, is covered by c, or there are no children covered by c.

A4 Using known pointer-jumping techniques, identify the connected components of G_v, and verify that each is a simple path—not a cycle (otherwise, FAILURE). Call these paths *color chains.*

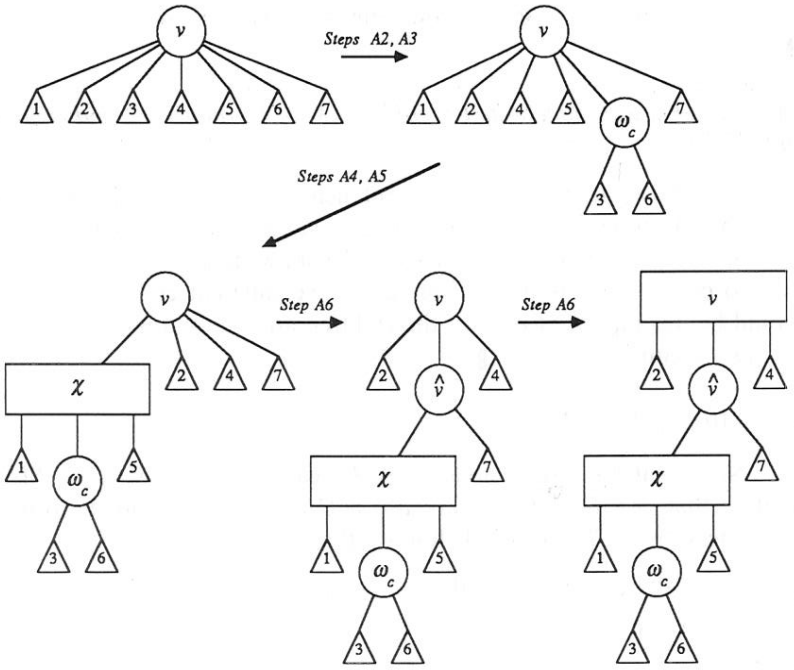

FIG. 7. Applying phase A to a P-node v, with children labelled 1 through 7. For this example, the colors are *blue*, *red*, and *tan*. The coloring of nodes is as follows: $\Delta(v) = \langle \text{tan, blue} \rangle$, $\Delta(1) = \langle \text{red}, c_1 \rangle$, $\Delta(2) = \langle \text{tan}, c_2 \rangle$, $\Delta(3) = \langle \text{red, red} \rangle$, $\Delta(4) = \langle \text{blue}, c_4 \rangle$, $\Delta(5) = \langle \text{red}, c_5 \rangle$, $\Delta(6) = \langle \text{red, red} \rangle$, $\Delta(7) = \langle c_7, c_7 \rangle$.

A5 For each color chain χ containing at least two nodes,

> Choose one of the two orientations of χ arbitrarily.
> Reorder the children of v so that the nodes of χ are consecutive,
> and insert a new Q-node between these nodes and v.

A6 Consider the subset of v's current children consisting of nodes at which no color is incomplete. Reorder the children of v to make this subset consecutive, and, if it is a proper subset containing at least two children, insert a new P-node v between v and the subset, and rename v to be a Q-node.

Steps A1, A3, A5, and A6 can be implemented using small-integer sorting. Let T_A be the result of Phase A.

LEMMA 2.8. *No P-node of T_A is orientable.*

Proof. A node v that was a P-node in T is a Q-node in T_A, or else no color is incomplete at a child of v in T_A, in which case $\text{INC}(v) = \varnothing$, so v is not orientable. A node w_c created in step A2 is covered by c. For a P-node $\hat{v}$ created in step A6, if u

is a child of $\hat{v}$, then no color is incomplete at u. Hence no color is incomplete at $\hat{v}$. ▮

LEMMA 2.9. *Every P-node u in T_A satisfies* (3) *of Corollary* 2.2 *and condition* (b) *of Lemma* 2.7.

Proof. A node w_c is covered by c, and hence trivially satisfies both (3) and condition (b). A node $\hat{v}$ has no incomplete colors (as shown in the proof of Lemma 2.8), and hence trivially satisfies (3). To show that $\hat{v}$ satisfies condition (b), note that no colors appears in the frontier of two different children of $\hat{v}$ (else the color would be incomplete at each of the children and hence appear in two distinct color chains—a contradiction). ▮

LEMMA 2.10. $L(T_A) \subseteq L(T)$.

Proof. Note that for every node v of T, $\text{leaves}_{T_A}(v) = \text{leaves}_T(v)$. Moreover, every node v that is a Q-node in T is also a Q-node in T_A, and the order of v's children is the same in T and T_A. It follows that

- there is a PQ-tree $T' \cong T$ with $fr(T') = fr(T_A)$, and
- any equivalence transformation that may be applied to T_A may also be applied to T'.

This proves the lemma. ▮

LEMMA 2.11. $L(T_A) \supseteq \Psi(T, \delta)$.

Proof. If $\Psi(T, \delta) = \varnothing$, the lemma is trivial. Therefore, assume $\Psi(T, \delta) \neq \varnothing$. It is easy to see that if Phase A is applied to a tree T' equivalent to T, the result T'_A is always a tree equivalent to T_A. Indeed, when a new node is inserted between v and some of v's children, the choice of children depends only on what colors are incomplete at what nodes, and this is independent of the order of children. Thus each node of T'_A has the same children as in T_A. Moreover, since the construction of G_v also depends only on incomplete colors, the color chains are the same, and so the order of children of each new Q-node is the same in T'_A and T_A, up to a flip.

Let T' be any PQ-tree $\cong T$ whose frontier satisfies every color. We will prove that Phase A may be applied to T', yielding a PQ-tree T'_A, without altering the frontier, i.e., such that $\text{fr}(T'_A) = \text{fr}(T')$. By the remarks above, $T'_A \cong T_A$. This proves that $\Psi(T, \delta) \subseteq L(T_A)$.

The reader may find it helpful to refer to Fig. 7 as well as to the procedure for Phase A.

Let v be a P-node of T' with children $v_1 \cdots v_s$. Let c be any color incomplete at some child. Suppose at least one child of v is covered by c; the case in which there are no such children is similar. By Corollary 2.3, the children covered by c are all consecutive. Hence in step A2, when a P-node w_c is inserted between these covered children and v, this transformation can be carried out without altering the frontier.

The result of step A2 is a PQ-tree T_1 such that $\mathrm{fr}(T_1) = \mathrm{fr}(T')$ and each P-node that was also in T' has at most one child covered by a given color.

By Lemma 2.5, step A3 does not fail. Let $\{v_i, v_j\}$ be an edge of the graph G_v constructed in step A3 of Phase A. By construction of G_v, there is some color c incomplete at both v_i and v_j. If c covers one of these nodes, say v_j, then c does not cover the other, v_i. By Corollary 2.3, v_i must be adjacent to v_j in the ordering of v's children in T_1. If c covers neither node, then c covers none of v's children. Again v_i and v_j must be adjacent by Corollary 2.3.

We have verified that two adjacent nodes of G_v are adjacent children of v in T_1. It follows that each connected component of G_v is a simple path whose nodes, in order, are a consecutive subsequence of children of v. (Note that this implies that step A4 of Phase A does not fail.) Hence in step A5, when a new Q-node is inserted between v and this subsequence of children, the insertion can be carried out while preserving the frontier. The result of step A5 is a PQ-tree T_2 such that for each P-node v and each color incomplete at a child of v, v has only one child at which the color is incomplete.

Now if some color c is incomplete at v, it had better be incomplete at a leftmost or rightmost child of v in T_2 by Corollary 2.2. Each color incomplete at v occurs in a unique child of v. Therefore the children of v at which no color is incomplete form a consecutive subsequence. Hence step A6, where a new P-node $\hat{v}$ is inserted between v and these children, may be done without altering the frontier. We thus obtain T'_A such that $\mathrm{fr}(T'_A) = \mathrm{fr}(T')$. This completes the proof. ∎

Having processed the P-nodes in Phase A, we focus entirely on Q-nodes in Phase B. We now give an informal description of Phase B. The goal of Phase B is to rearrange the PQ-tree so that each node v satisfies (3) of Corollary 2.2 and condition (b) of Lemma 2.7. Each child v_i of v has a pair $\Delta(v_i) = \langle c_1, c_2 \rangle$ of colors. Assume (in accordance with Corollary 2.2) that $lr[v_i] \sim \langle c_1, c_2 \rangle$. Thus if v_i is *orientable* (i.e., $c_1 \neq c_2$), then v_i has two distinct "orientations," $lr[v_i] = \langle c_1, c_2 \rangle$ and $lr[v_i] = \langle c_2, c_1 \rangle$. Note that by flipping every node in the subtree rooted at v_i, we go from one orientation of v_i to the other.

In step B3, each Q-node v "tries" to choose orientations $\overline{LR}[v_i]$ for its children $v_1 \cdots v_s$ such that $\overline{LR}[v_1] \cdots \overline{LR}[v_s]$ satisfies every color (cf. condition (b) of Lemma 2.7). If such orientations can be chosen, v stores in $LR[v]$ the resulting orientation of v; i.e., such that if $lr[v_i] = \overline{LR}[v_i]$ for each child v_i, then $lr[v] = LR[v]$.

It is useful to think of $\overline{LR}[v_i] = \langle c_1, c_2 \rangle$ as a "request" that v_i adopt the orientation $\langle c_1, c_2 \rangle$, if the order of v's children remains unchanged, and adopt the orientation $\langle c_2, c_1 \rangle$ if v flips.

In steps B7 through B11, the requests are fulfilled. Each orientable node v compares the request it received ($\overline{LR}[v]$) to its "idea" of its current orientation ($LR[v]$) and sets $\mathrm{OPP}[v]$ to *true* if there is a discrepancy. Then each node computes the parity $\mathrm{REV}[v]$ of the number of discrepances along the path from itself to the root, and flips if the number of discrepancies is odd. This ensures that if there is

a discrepancy between $LR[v]$ and $\overline{LR}[v]$, either v or its parent flips. Thus in the resulting tree T_B, all discrepancies wil have been resolved. This is the meaning of Lemma 2.12. ("OPP" and "REV" are intended to be pneumonic for "opposite" and "reverse.")

It can be proved that $\mathrm{fr}(T_b)$ satisfies all colors. But we can say more. In fact, flips may be permitted which do not reintroduce discrepancies. Thus if an orientable node v flips, its parent must also flip, and vice versa. This is the meaning of Eq. (10).

We use the following notation in phase B and in the proof thereafter.

DEFINITION 2.12. If $\langle a, b\rangle \sim \langle a', b'\rangle$ and $a \neq b$, we write $\langle a, b\rangle \,\mathrm{swap}\langle a', b'\rangle$ for the predicate that is *true* if $\langle a, b\rangle = \langle b', a'\rangle$ and *false* if $\langle a, b\rangle = \langle a', b'\rangle$. We write $\oplus$ to denote "exclusive-or," and assign it lowest precedence.

Note that

$$\langle a, b\rangle \,\mathrm{swap}\langle a'', b''\rangle = \langle a, b\rangle \,\mathrm{swap}\langle a', b'\rangle \oplus \langle a', b'\rangle \,\mathrm{swap}\langle a'', b''\rangle \qquad (4)$$

if all three predicates are defined. Phase B is shown below. (The function h_v is defined in Definition 2.11.)

PHASE B.

B1 For each Q-node v:

B2 Let the children of v be $v_1 \cdots v_s$.

B3 Assign to each $\overline{LR}[v_i]$ either $\Delta(v_i)$ or $\Delta(v_i)^R$ such that every color in the sequence $\overline{LR}[v_1 \cdots \overline{LR}[v_s]$ occurs consecutively, and such that $h_v(\langle \bar{L}[v_1], \bar{R}[v_s]\rangle) \sim \Delta(v)$.

B4 If this is impossible, return FAILURE.

B5 Otherwise, set $LR[v] := h_v(\langle \bar{L}[v_1], \bar{R}[v_s]\rangle)$.

B6 For each node v:

B7 if v is orientable, then
set $\mathrm{OPP}[v] := LR[v]$ swap $\overline{LR}[v]$
otherwise, set $\mathrm{OPP}[v] :=$ *false*.

B8 For each node v:

B9 set $\mathrm{REV}[v] := \otimes_{u \text{ an ancestor of } v} \mathrm{OPP}[u]$.

B10 For every orientable node v:

B11 if $\mathrm{REV}[v]$ then flip v.

Let T_B be the result of Phase B. Note that by steps B3 and B5, if v is an orientable node of T_B, then each of $LR[v]$ and $\overline{LR}[v]$ is $\sim \Delta(v)$. The implementation of Phase B uses only elementary techniques. In particular, step B3 can be done using ideas from the proof of Corollary 2.3. Note also that step B9 of Phase B can be done using standard parallel pointer-jumping techniques.

DEFINITION 2.13. For a Q-node v (or an R-node, to be defined later) in PQ-trees T_1 and T_2, define $T_1 \operatorname{flip}_v T_2$ to be *true* if the order of children of v in T_1 is the reverse of the order in T_2, *false* if the order is the same, and undefined otherwise.

Note that

$$T_1 \operatorname{flip}_v T_3 = T_1 \operatorname{flip}_v T_2 \oplus T_2 \operatorname{flip}_v T_3 \tag{5}$$

if all three predicates are defined.

The following lemma captures the effect of the flips performed in step B11 of Phase B (see Definition 2.11.)

LEMMA 2.12. *If v is an orientable node of T_A,*

$$LR[v] \operatorname{swap} \overline{LR}[v] = T_A \operatorname{flip}_v T_B \oplus T_a \operatorname{flip}_{p(v)} T_B.$$

Proof. In step B7 of Phase *B*, we set

$$\mathrm{OPP}[v] := LR[v] \operatorname{swap} \overline{LR}[v].$$

In step B9, we have

$$\begin{aligned} \mathrm{REV}[v] &:= \bigoplus_{u \text{ an ancestor of } v} \mathrm{OPP}[u] \\ &= \mathrm{OPP}[v] \oplus \left(\bigoplus_{u \text{ an ancestor of } p(v)} \mathrm{OPP}[u] \right) \\ &= \mathrm{OPP}[v] \oplus \mathrm{REV}[p(v)] \end{aligned}$$

and by step B11, $T_A \operatorname{flip}_v T_B = \mathrm{REV}[v]$ and $T_A \operatorname{flip}_{p(v)} T_B = \mathrm{REV}[p(v)]$. The lemma follows. ∎

The following lemma states that the frontier of a tree satisfies every color iff every request is fulfilled in that tree.

LEMMA 2.13. *For $T'_B \cong T_B$, $\mathrm{fr}(T'_B)$ satisfies every color iff for every orientable node v,*

$$\overline{LR}[v] \operatorname{swap} lr_{T'_B}[v] = T'_B \operatorname{flip}_{p(v)} T_A. \tag{6}$$

Proof. Assuming that (6) holds for every orientable child v of u, and that $\overline{LR}[v]$ is chosen in accordance with step B3 of phase B, it follows that (3) and condition (b) hold for u, and hence that $\mathrm{fr}_{T'_B}(u)$ satisfies every color, by Lemma 2.7. On the other hand, a violation of (6) would mean a violation of either (3) or condition (b). ∎

Note that it follows from the proof that if Phase B fails, there is no ordering generated by T_A satisfying all colors.

The following lemma states that if no discrepancies are introduced below a node of a tree, that node's "idea" of its own orientation is correct.

LEMMA 2.14. *For any orientable node u, if $T'_B \operatorname{flip}_v T_B = T'_B \operatorname{flip}_{p(v)} T_B$ for each orientable proper descendent v of u, then $LR[u]$ swap $lr_{T'_B}[u] = T'_B \operatorname{flip}_u T_A$.*

Proof. By induction on height of u. Trivial for leaves, which are not orientable. Let u be an orientable node, and assume the lemma holds for the children $u_1 \cdots u_s$ of u. We consider the case $T'_B \operatorname{flip}_u T_A = \mathit{false}$. (The case in which $T'_B \operatorname{flip}_u T_A = \mathit{true}$ is analogous.) In this case, u_1 and u_s, respectively, are the leftmost and rightmost children of u in T'_B, so

$$lr_{T'_B}[u] = h_u(\langle l_{T'_B}[u_1], r_{T'_B}[u_s] \rangle)$$

In step B5, $LR[u]$ is assigned $h_u(\langle \bar{L}[u_1], \bar{R}[u_s] \rangle)$. We therefore must prove that $h_u(\bar{L}[u_1]) = h_u(l_{T'_B}[u_1])$ and $h_u(\bar{R}[u_s]) = h_u(r_{T'_B}[u_s])$. We show the former equality; the proof of the latter equality is analogous.

We know $LR[u_1] \sim lr[u_1]$; the equality is then trivial if u_1 is not orientable. Therefore, assume u_1 is orientable. Our premise implies that $T'_B \operatorname{flip}_{u_1} T_B \oplus T'_B \operatorname{flip}_u T_B = \mathit{false}$. We assumed $T_A \operatorname{flip}_u T'_B = \mathit{false}$, so

$$\begin{aligned} T'_B \operatorname{flip}_{u_1} T_B \oplus T_A \operatorname{flip}_u T_B &= (T'_B \operatorname{flip}_{u_1} T_B \oplus T'_B \operatorname{flip}_u T_B) \oplus (T_A \operatorname{flip}_u T'_B) \\ &= (\mathit{false}) \oplus (\mathit{false}) \qquad \text{by Eq. (5).} \end{aligned}$$

Then

$$\begin{aligned} T_A \operatorname{flip}_{u_1} T'_B &= T_A \operatorname{flip}_{u_1} T'_B \oplus (T'_B \operatorname{flip}_{u_1} T_B \oplus T_A \operatorname{flip}_u T_B) \\ &= T_A \operatorname{flip}_{u_1} T_B \oplus T_A \operatorname{flip}_u T_B \qquad (7) \\ &= LR[u_1] \text{ swap } \overline{LR}[u_1] \qquad \text{by Lemma 2.12.} \qquad (8) \end{aligned}$$

By the inductive hypothesis,

$$LR[u_1] \text{ swap } lr_{T'_B}[u_1] = T_A \operatorname{flip}_{u_1} T'_B. \qquad (9)$$

By combining (7) and (9), we obtain $\overline{LR}[u_1] = lr_{T'_B}[u_1]$ which implies the desired equality. ∎

Now we give a locally enforceable condition for a tree equivalent to T_B to have a frontier satisfying every color.

LEMMA 2.15. *For $T'_B \cong T_B$, the frontier of T'_B satisfies every color iff for every orientable node*

$$T'_B \operatorname{flip}_v T_B = T'_B \operatorname{flip}_{p(v)} T_B. \qquad (10)$$

Proof. By Lemma 2.13, the frontier of T'_B satisfies every color iff (6) holds for every orientable node v of T'_B. We use (4) and (5) to combine (6) with the conclusion of Lemma 2.12, obtaining

$$LR[v] \text{ swap } lr_{T'_B}[v] = T_A \text{ flip}_v T_B \oplus T_B \text{ flip}_{p(v)} T'_B. \tag{11}$$

Thus the frontier of T'_B satisfies every color iff (11) holds for every orientable node v. It remains to show that (11) holds for every orientable node v iff (10) holds for every orientable node v.

($\Leftarrow$) Assume that (10) holds for every orientable node v. Let v be an orientable node. By Lemma 2.14,

$$\begin{aligned} LR[v] \text{ swap } lr_{T'_B}[v] &= T'_B \text{ flip}_v T_A \\ &= T'_B \text{ flip}_v T_B \oplus T_B \text{ flip}_v T_A \qquad \text{by (5)} \\ &= T'_B \text{ flip}_{p(v)} T_B \oplus T_B \text{ flip}_v T_A \qquad \text{by assumption.} \end{aligned}$$

We have proved (11) holds for any orientable node.

($\Rightarrow$) Assume that (11) holds for every orientable node v. We prove that (10) holds for every orientable v, by induction on height of v. The basis is trivial because leaves are not orientable. Assume (10) holds for all orientable proper descendents of v. By Lemma 2.14,

$$LR[v] \text{ swap } lr_{T'_B}[v] = T'_B \text{ flip}_v T_A.$$

We use (11) to substitute for the left-hand side, yielding

$$T'_B \text{ flip}_v T_A = T_A \text{ flip}_v T_B \oplus T_B \text{ flip}_{p(v)} T'_B$$

which implies (10). ∎

In order to enforce (10), it is useful to invent a new kind of node, an *R-node*, which is like a Q-node, only more restrictive. A "legal" set of equivalence transformations on a PQ-tree with R-nodes is a set which flips each R-node if and only if it flips the parent of the R-node. In this sense, an R-node "follows the lead" of its parent node. (An R-node is not permitted to be the child of a P-node, only of a Q-node or another R-node.)

Another way to view an R-node is as a notational device for signifying that the R-node's children should be inserted into the sequence of its parent's children. An R-node may be eliminated and its children reattached to its parent without disturbing their order; the resulting tree generates exactly the same set of orderings. This is illustrated in Fig. 8, where we signify that a node is an R-node by using two lines to connect it to its parent. R-nodes are merely a notational and computational convenience—they do not enhance the expressibility of PQ-trees.

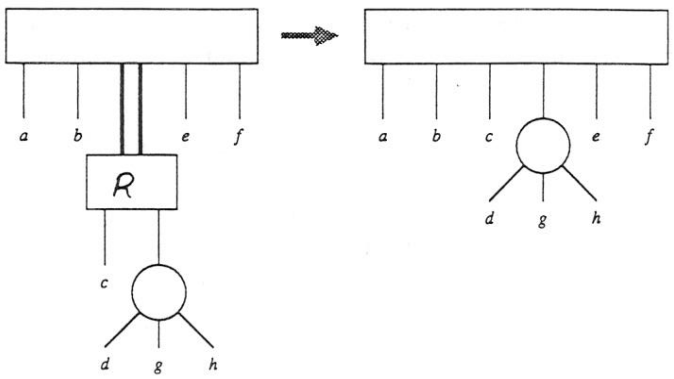

FIG. 8. Eliminating an R-node by reattaching its children to its parent without disturbing their order.

In order to ensure that (10) holds for every orientable node v, we merely carry out phase C:

PHASE C. For each orientable node v, rename v to be an R-node.

Let T_C be the result of phase C.

LEMMA 2.16. $L(T_C) = \Psi(T_B, \delta)$.

Proof. Suppose $T'_B \cong T_B$. Consider the set of equivalence transformations used to obtain T'_B from T_B. If this set of transformations may be legally applied to T_C, the result would be a PQ-tree T'_C equivalent to T_C and having the same frontier as T'_B, so $\text{fr}(T'_B) \in L(T_C)$. Otherwise, $\text{fr}(T'_B) \notin L(T_C)$. The set of transformations is applicable to T_C iff an R-node is flipped whenever its parent is flipped; i.e., $T'_B \text{flip}_v T_B = T'_B \text{flip}_{p(v)} T_B$ for each R-node v in T_C. The R-nodes of T_C are the orientable nodes, so we have

$$\text{fr}(T'_B) \in L(T_C)$$

iff for each orientable node v,

$$T'_B \text{flip}_v T_B = T'_B \text{flip}_{p(v)} T_B.$$

In view of Lemma 2.15, this completes the proof. ∎

It remains to obtain a PQ-tree $\hat{T}$ such that $L(\hat{T}) = L(T_C)$ and $\hat{T}$ has no R-nodes. We can accomplish this in $O(\log n)$ time using n processors as follows: First compute a preorder numbering of T_C (using the techniques in [19]). Now, by use of parallel pointer-jumping techniques, each child v of an R-node can determine its lowest Q-node ancestor. This will be v's new parent. Each Q-node can use the preorder numbering to sort all its new children to obtain the proper order for them. The resulting tree $\hat{T}$ generates the same orderings as T_C, but has no R-nodes.

This completes the description of the disjoint-reduction algorithm and the proof of Theorem 2.1.

2.3. *Intersection*

In this subsection, we give an algorithm for intersecting a pair of *PQ*-trees. For each node v of a *PQ*-tree T over a ground set S, we define a collection of sets, called the *constraint sets of* v (in T), as follows:

- One constraint set is leaves(v).
- If v has children $v_1, ..., v_s$ in order, and $A_i = \text{leaves}(v_i)$, then the sets

$$A_1, ..., A_s \tag{12}$$

are also constraint set of v.

- Finally, if v is a Q-node, the sets

$$A_1 \cup A_2, A_3 \cup A_4, ..., A_{2\lfloor s/2 \rfloor - 1} \cup A_{2\lfloor s/2 \rfloor} \tag{13}$$

and the sets

$$A_2 \cup A_3, A_4 \cup A_5, ..., A_{2\lceil s/2 \rceil - 2} \cup A_{2\lceil s/2 \rceil - 1} \tag{14}$$

are also constraint sets of v.

Remark 1. The sets (12) are all disjoint, as are the sets (13) and the sets (14). Hence the collection of constraint sets of v can be devided into four subcollections, where each subcollection consists of disjoint sets.

Remark 2. If T_0 is any subtree of T (i.e., any connected subgraph of T), v is an internal node of T_0, and A_0 is a constraint set of v in T_0, then the corresponding constraint set of v in T is $A = \bigcup_{w \in A_0} \text{leaves}_T(w)$.

LEMMA 2.17. *For a PQ-tree T over the groud set S, an ordering λ of S is in $L(T)$ iff λ satisfies every constraint set of every node of T.*

Proof. Note that the constraint sets of v in T are the same as the constraint sets of v in any $T' \cong T$, and for any $T' \cong T$, fr(T') satisfies every constraint set of every node. It is easy to show by induction on the height of v that if an ordering τ of leaves(v) satisfies all the constraint sets of v and all its descendents, then τ is the frontier of some *PQ*-tree equivalent to the subtree of T rooted at v. ∎

COROLLARY 2.4. *Let T and T' be two PQ-trees over the same ground set. Then*

$$L(T) \cap \Psi(T') = \Psi(T', \{C_1, ..., C_t\}),$$

where $C_1, ..., C_t$ are all the constraint sets of all of the nodes of T.

Corollary 2.4 suggests an algorithm for finding the intersection of T with T': reduce T' with respect to all the constraint sets of all the nodes of T. While this approach would work, it would require many calls to MREDUCE, which expects its sets to be disjoint. Instead, we achieve the same effect using a decomposition of T'; we invoke MREDUCE in parallel on many different parts of T'. We then make use of the following lemma:

LEMMA 2.18. *Let T be a PQ-tree, and let T_0 be a tree obtained from T by deleting all the proper descendents of some nodes of T. The ground set of T_0 consists of leaves of T_0, which are nodes (not necessarily leaves) of T. Suppose we reduce T_0 with respect to a subset A_0 of its ground set. The effect on T is the same as if we had reduced T with respect to the set $A = \bigcup_{v \in A_0} \text{leaves}_T(v)$.*

Proof. Denote by $\hat{T}$ the tree T after the subtree T_0 has been reduced with respect to A_0. Also, let $\hat{T}_0$ be the subtree T_0 after reduction. Let $\hat{T}'$ be any tree equivalent to $\hat{T}$, and let $\hat{T}'_0$ be the corresponding subtree. For each leaf v of $\hat{T}'_0$, let $\lambda_v = \text{fr}_{\hat{T}'}(v)$. If $\text{fr}(\hat{T}'_0) = v_1 \cdots v_k$, then $\text{fr}(\hat{T}') = \lambda_{v_1} \cdots \lambda_{v_k}$. By definition of A, if $v_i \in A_0$, then λ_{v_i} consists entirely of elements of A. It follows that $v_1 \cdots v_k$ satisfies A_0 iff $\lambda_{v_1} \cdots \lambda_{v_k}$ satisfies A. The lemma then follows from the correctness of the reduction procedure. ∎

To ensure that in our intersection algorithm, only a few parallel invocations are necessary, we use a well-known technique of parallel algorithm design: a tree *separator*.

Fix a tree T of n nodes. A node v of T with s children determines a separation of T into $s+1$ subtrees: let $T_{(1)}, \ldots, T_{(s)}$ be the subtrees of T rooted at the children of v, and let $T_{(0)}$ be the subtree obtained from T by deleting $T_{(1)}, \ldots, T_{(s)}$. We say v is a *good separator* of T if each subtree $T_{(0)}, \ldots, T_{(s)}$ has no more than $n/2$ nodes. It is well known that every tree with at least two nodes has a good separator. To see this, let $\text{size}(v)$ be the number of descendents of v, for each node v of T. Since $\text{size}(\text{root of } T) = n$ and $\text{size}(\text{any leaf of } T) = 1$, there must be some node v of maximal depth such that $\text{size}(v) \geqslant 1 + (n/2)$. Then v is a good separator. Using, e.g., the Euler tour technique of [19], $\text{size}(\cdot)$ can be computed in $O(\log n)$ time using n processors. We have, therefore,

LEMMA 2.19. *A good separator for a tree with $n \geqslant 2$ nodes can be found in $O(\log n)$ time using n processors.*

Finally, we call upon another technique that is specific to *PQ*-trees. Once we have reduced a *PQ*-tree with respect to a set E, Lemma 2.1 implies that E is contiguous in T. Recall from Definition 2.4 that this means that

- either $E = \text{leaves}(\text{lca}(E))$, if $\text{lca}(E)$ is a P-node or a leaf,
- or else there is a consecutive subsequence $v_p \cdots v_q$ of the children $v_1 \cdots v_s$ of $\text{lca}(E)$ such that $E = \bigcup_p^q \text{leaves}(v_i)$.

We need to separate the portion of the tree pertinent to E from the rest of the tree. We call this *segregation.*

DEFINITION 2.14. We say E is *segregated* in T if $E = \text{leaves}(\text{lca}_T(E))$.

If E is contiguous in T, we can modify T, obtaining T', so that E is segregated in T' and $L(T') = L(T)$. Namely, if E is not already segregated, then $v = \text{lca}_T(E)$ is a Q-node, by contiguity. Insert an R-node z between the children $v_p \cdots v_q$ and v. (R-nodes are defined at the end of Subsection 2.2.) In the resulting tree T', $z = \text{lca}_{T'}(E)$ and $\text{leaves}(z) = \bigcup_p^q \text{leaves}(v_i) = E$. Moreover, it follows from the definition of the R-node that $L(T') = L(T)$. We can easily carry out this modification or undo it in $O(\log n)$ time and n processors, where n is the number of nodes in T. In fact, if $E_1, ..., E_k$ are disjoint subsets of the ground set of T, all contiguous in T, we can segregate T with respect to all these sets within the same resource bounds.

The algorithm for PQ-tree intersection follows. In the way we describe it, the algorithm modifies T' to be the intersection. In applying the algorithm, it may be desirable to copy T' first and work with this copy, so the original T' remains available. Also, the intersection tree resulting may have R-nodes, but these can be eliminated as described at the end of Subsection 2.2.

INTERSECT(T, T').

I1 If T has only one node, return.

I2 Find a good separator v of T, with children $v_1 \cdots v_s$. Using at most four invocations of MREDUCE (in accordance with Remark 1 above), reduce T' with respect to all the constraint sets of v.

I3 Segregate T' with respect to $\text{leaves}_T(v)$, and let $v' = \text{lca}_{T'}(\text{leaves}_T(v))$.

I4 For $i = 1, ..., s$, segregate T' with respect to $\text{leaves}_T(v_i)$, and let $v_i' = \text{lca}_{T'}(\text{leaves}_T(v_i))$.

I5 Let $T_{(0)}, ..., T_{(s)}$ be the subtrees of T determined by v, as defined above. For each subtree $T_{(i)}$, there is a corresponding subtree T_i' of T': for $i = 1, ..., s$, let T_i' be the subtree of T' rooted at v_i', and let T_0' be the subtree of T' obtained by deleting all proper descendents of v'. Thus v' is a leaf of T_0', just as v is a leaf of $T_{(0)}$. If we identify v with v', then corresponding subtrees have identical ground sets. Recursively intersect corresponding subtrees of T and T'.

To prove the correctness of the intersection procedure, a simple induction on the number of nodes of T shows that the effect of *INTERSECT*(T, T') is that of reducing T' with respect to all the constraint sets of all the nodes of T. (The correctness then follows from Corollary 2.4.) The basis, where T has only one node, is trivial. The induction step follows from Remark 2 and Lemma 2.18.

The time for each recursive call is dominated by the time for reduction, which is $O(\log n)$ if n processors are used, where n is the number of nodes. The recursive calls on the subtrees may be done in parallel; the node sets of the various subtrees

are disjoint, so we may again assign one processor per node. Because a good separator is used in step I2, the recursion depth $\leqslant \log n$. Hence the total time is $O(\log^2 n)$.

This concludes our description of the intersection algorithm and the proof of Theorem 2.3. ▮

2.4. *Representation of Cycles with PQ-Trees*

In this section, we prove lemmas 2.2 and 2.3, which are reproduced below:

LEMMA 2.2. *Let T be a PQ-tree whose ground set is the disjoint union of nonempty sets A, B, C. Suppose that each of these sets is contiguous in T. Then T may be modified to be a PQ-tree T' in which $A \cup B$ and $B \cup C$ are contiguous and such that* $\mathrm{CYC}(T') = \mathrm{CYC}(T)$.

Lemma 2.2 allows us to place a PQ-tree T in a special form, if T is being used to represent cycles.

LEMMA 2.3. *Suppose A is a proper subset of the ground set of T, and is contiguous in T. If $A_1 \subseteq A$, then for each $\lambda \in L(T)$, λ satisfies A_1 iff* $\mathrm{cyc}(\lambda)$ *satisfies* A_1.

Lemma 2.3 allows us to reduce a PQ-tree T used for representing cycles. For suppose $A_1, ..., A_k$ are disjoint subsets of a proper subset A of the ground set of T, and A is contiguous in T. It follows from Lemma 2.3 that

$$\mathrm{cyc}(\Psi(T, \{A_1, ..., A_k\})) = \{\sigma \in \mathrm{CYC}(T) : \sigma \text{ satisfies } A_1, ..., A_k\}$$

and the left-hand side is just $\mathrm{CYC}(\hat{T})$, where $\hat{T}$ is the PQ-tree computed by MREDUCE.

Proof of Lemma 2.3. Certainly if $\lambda = w_1 \cdots w_n$ satisfies A_1 then $\mathrm{cyc}(\lambda)$ satisfies A_1. For the converse, suppose $\mathrm{cyc}(\lambda)$ satisfies A_1. Suppose both endpoints of λ were in A. By Lemma 2.1, since A is contiguous, λ satisfies A, so all the elements of λ are in A, contradicting the fact that A is a proper subset of the ground set of T. Thus at least one endpoint of λ, say the left endpoint w_1, does not belong to A. It follows that the consecutive subsequence of $\mathrm{cyc}(\lambda)$ consisting of the elements of A_1 is contained entirely in $w_2 \cdots w_n$. This prove the lemma. ▮

We next proceed with the proof of Lemma 2.2. We begin with a simple observation.

Observation. If $v_1 \cdots v_s$ are *some* children of a node r, and $\mathrm{leaves}(r) = \bigcup_1^s \mathrm{leaves}(v_i)$, then r has no other children.

LEMMA 2.20. *Let T be a PQ-tree with root r having children $v_1 \cdots v_s$ in order. Let T_1 be the same as T except that the order of r's children has been cyclically shifted; i.e., the order is $v_i \cdots v_s v_1 \cdots v_{i-1}$. Then* $\mathrm{CYC}(T_1) = \mathrm{CYC}(T)$.

Lemma 2.20 follows directly from our redundant representation of cycles.

LEMMA 2.21. *Let T be a PQ-tree with root r having exactly two children v_1 and v_2, both Q-nodes. Let T_1 be the same as T except that one child, say v_2, has been renamed an R-node. Then* $\mathrm{CYC}(T_1) = \mathrm{CYC}(T)$.

(R-nodes are defined at the end of Subsection 2.2.)

Proof. Clearly $\mathrm{CYC}(T_1) \subseteq \mathrm{CYC}(T)$. To prove the reverse containment, let T' be any PQ-tree equivalent to T; we show that $\mathrm{cycfr}(T) \in \mathrm{CYC}(T_1)$. Obtain T'' from T' as follows: if $T'\,\mathrm{flip}_{v_2}\,T = T'\,\mathrm{flip}_r\,T$ then let $T'' = T'$; otherwise, obtain T'' from T' by flipping the root r. We then have $T''\,\mathrm{flip}_{v_2}\,T = T''\,\mathrm{flip}_r\,T$. This means that the equivalence transformations used to obtain T'' from T may be applied to T_1 while respecting the R-node in T_1. Hence $\mathrm{fr}(T'') \in L(T_1)$, so certainly $\mathrm{cycfr}(T'') \in \mathrm{CYC}(T_1)$. It remains to show that $\mathrm{cycfr}(T'') = \mathrm{cycfr}(T')$. Let $\lambda_1 = \mathrm{fr}_{T'}(v_1)$ and let $\lambda_2 = \mathrm{fr}_{T'}(v_2)$. Then $\mathrm{fr}(T')$ is either $\lambda_1\lambda_2$ or $\lambda_2\lambda_1$, so $\mathrm{fr}(T'')$ is either $\lambda_1\lambda_2$ or $\lambda_2\lambda_1$. In either case, $\mathrm{cycfr}(T'') = \mathrm{cyc}(\lambda_1\lambda_2) = \mathrm{cycfr}(T')$. ∎

Proof of Lemma 2.2. Segregate A, B, and C in T, and let $a = \mathrm{lca}(A)$, $b = \mathrm{lca}(B)$, and $c = \mathrm{lca}(C)$. We first want to make a, b, and c all children of the root r. Assume they are not already; then at least two of these nodes, say b and c, are descendents of a single node v which is a child of r. Using the observation above and the fact that every node has at least two children, one can verify that r has exactly two children, a and v, and v has exactly two children, b and c. Carry out the following three modifications on T:

1. If a is an R-node and v is not, rename a to be a Q-node, obtaining T_1.
2. Rename v to be an R-node, obtaining T_2.
3. Eliminate v and attach its children b and c to the root, obtaining T_3.

By Lemma 2.21, $\mathrm{CYC}(T_1) = \mathrm{CYC}(T)$ and $\mathrm{CYC}(T_2) = \mathrm{CYC}(T_1)$. Finally, $\mathrm{CYC}(T_3) = \mathrm{CYC}(T_2)$ by definition of an R-node (defined at the end of Subsection 2.2).

In T_3, the root r has exactly three children a, b, and c, in some order. Using Lemma 2.20, we can obtain T_4 such that $\mathrm{CYC}(T_4) = \mathrm{CYC}(T_3)$ and the order of children is either a, b, c or c, b, a. Hence T_4 satisfies the lemma. ∎

The R-nodes can be eliminated as in Fig. 8.

2.5. *Join*

In this subsection, we show how to compute the join of two PQ-trees. The reader is referred to Subsection 2.1 for notation and definitions.

To review from Subsection 2.3, we say a subset E of the ground set of T is *segregated* in T if $E = \mathrm{leaves}(\mathrm{lca}_T(E))$. If E is contiguous (Definition 2.4) in T but not segregated, we can introduce an R-node into T, obtaining T', so that E is segregated in T', and $L(T') = L(T)$.

If E is segregated in T, we denote by $T \mid E$ the subtree of T rooted at $\text{lca}_T(E)$. Note that $T \mid E$ is a PQ-tree whose ground set is E.[5]

We use the following notation: For a node z, we let $\text{CYC}_{\text{fix}\,x}(T)$ denote the set

$$\{\text{cycfr}(T'): T' \cong T \text{ and the order of children of } z \text{ is the same in } T' \text{ and } T\}.$$

In the case in which z is a leaf, the last clause in the definition of $\text{CYC}_{\text{fix}\,z}(T)$ is trivially satisfied, so $\text{CYC}_{\text{fix}\,z}(T) = \text{CYC}(T)$.

If z is a Q-node or an R-node, we let $\text{CYC}_{\text{flip}\,z}(T)$ denote the set

$$\{\text{cycfr}(T'): T' \cong T \text{ and } T' \text{ flip}_z\, T = \textit{true}\}.$$

Note that if z is a Q-node or an R-node,

$$\text{CYC}(T) = \text{CYC}_{\text{fix}\,z}(T) \cup \text{CYC}_{\text{flip}\,z}(T).$$

In this section, z will typically be $\text{lca}_T(E)$.

Define the one-place function splice$(\cdot)$ on pairs $\langle \alpha, \beta \rangle$ of linear orderings by $\text{splice}(\langle \alpha, \beta \rangle) = \text{cyc}(\alpha\beta)$. (We choose to define *splice* on pairs, rather than make it a two-place function, for reasons that will be apparent later.) For example, if σ is a cycle of the disjoint union of non-empty sets E and D, and σ satisfies E (and hence D as well), $\sigma = \text{splice}(\langle \sigma \mid D, \sigma \mid E \rangle)$. More specifically, if T is a PQ-tree whose ground set is the disjoint union of E and D, and $\text{leaves}_T(z) = E$, then $\text{fr}(T)$ has the form $\alpha\beta\gamma$, where $\beta = \text{fr}_T(z)$, and hence $\text{cycfr}(T) = \text{splice}(\langle \gamma\alpha, \beta \rangle)$.

LEMMA 2.22. *Suppose E is contiguous and segregated in PQ-trees T and T^*, and $T^* \mid E$ is identical to $T \mid E$. Then* $\text{cycfr}(T^*) \mid E = \text{cycfr}(T) \mid E$.

Proof. For any PQ-tree T in which E is segregated, $\text{cycfr}(T) \mid E = \text{fr}_T(\text{lca}_T(E))$. ∎

DEFINITION 2.15. Suppose the ground set of T is the disjoint union of non-empty sets D and E, and E is contiguous in T. We say E is *rigid* in T if the following two conditions are satisfied:

1. if E is segregated, then $\text{lca}(E)$ is an R-node, and
2. if D is segregated, then $\text{lca}(D)$ is an R-node.

Otherwise, if E is not rigid, we say E is *hinged* in T.

If 1 holds and 2 does not, we can first segregate E if it is not already segregated, and then use Lemma 2.21 to modify T so that 1 no longer holds. We therefore make the following assumption in this subsection:

[5] $T \mid E$ is essentially what Booth and Lueker called the *pertinent subtree* of T, in [3], although they did not consider the notion of segregation.

Assumption. If E is hinged in T, then E is segregated and $\text{lca}_T(E)$ is not an R-node.

Given a PQ-tree T with n nodes, it is easy to find $\text{lca}_T(E)$ and determine whether E is rigid in T in $O(\log n)$ time using n processors. The significance of the term "hinged" arises from the following corollary.

COROLLARY 2.5. *If E is contiguous and hinged in T, then*

$$\text{CYC}(T) = \text{splice}(\text{CYC}(T) \mid D \times \text{CYC}(T) \mid E).$$

In other words, for any $\alpha, \beta \in \text{CYC}(T)$, there is a $\gamma \in \text{CYC}(T)$ such that $\gamma \mid D = \alpha \mid D$ and $\gamma \mid E = \beta \mid E$.

Proof. For any $T_\alpha, T_\beta \cong T$, construct T_γ from T_α by reordering the children of each node v of $T_\alpha \mid E$ as they are ordered in T_β. Then $T_\gamma \mid E$ is identical to $T_\beta \mid E$, so $\text{cycfr}(T_\gamma) \mid E = \text{cycfr}(T_\beta) \mid E$ by Lemma 2.22. Also, for each node v whose children were reordered, $\text{leaves}(v)$ contains no elements of D, so $\text{cycfr}(T_\gamma) \mid D = \text{cycfr}(T_\alpha) \mid D$. Finally, because E is hinged in T, we assume that $z = \text{lca}_T(E)$ is not an R-node, so $T_\gamma \cong T$. ▮

Corollary 2.5 shows that when T is E-hinged, there is independence between the induced ordering on E and the induced ordering on D: each may be chosen independently of the other. In the case in which E is rigid in T, there is still a partial independence: each may be chosen *almost* independently of the other (viz., up to a reversal).

COROLLARY 2.6. *If E is contiguous and rigid in T, and $z = \text{lca}_T(E)$, then*

$$\text{CYC}_{\text{fix}\,z}(T) = \text{splice}(\text{CYC}_{\text{fix}\,z}(T) \mid D \times \text{CYC}_{\text{fix}\,z}(T) \mid E \tag{15}$$

$$\text{CYC}_{\text{flip}\,z}(T) = \text{splice}(\text{CYC}_{\text{flip}\,z}(T) \mid D \times \text{CYC}_{\text{flip}\,z}(T) \mid E. \tag{16}$$

In particular, for any $\alpha, \beta \in \text{CYC}(T)$, there is a $\gamma \in \text{CYC}(T)$ such that $\gamma \mid D = \alpha \mid D$ and $\gamma \mid E$ is either $\beta \mid E$ or $(\beta \mid E)^R$.

Proof. The proof resembles that of Corollary 2.5. Choose PQ-trees $T_\alpha, T_\beta \cong T$ such that the order of children of z in T_α and T_β is the same as the order in T. Obtain T_γ from T_α as in the proof of Corollary 2.5; we need only note that this does *not* change the order of children of z. This ensures that $T_\gamma\, \text{flip}_z\, T_\alpha = \mathit{false} = T_\gamma\, \text{flip}_{p(z)}\, T_\alpha$, where $p(z)$ is the parent of z, so the R-node z is respected; thus $T_\gamma \cong T$. This proves (15); the proof of (16) is analogous. ▮

The next two lemmas show that the partial independence of Corollary 2.6 is the best we can prove, since $\text{cycfr}(T) \mid E$ depend on the orientation of the node z and $\text{cycfr}(T) \mid D$ does as well if E is rigid in T.

DEFINITION 2.16. Suppose E is segregated in T, and $z = \text{lca}_T(E)$. We define T's

partition of E to be the partition $\{\text{leaves}(v_i)\colon\ v_i \text{ is a child of } z\}$. The order of children of z in T provides a natural ordering among the blocks of this partition: say the children of z are $v_1, \ldots, v_s$ in order. If $a \in \text{leaves}(v_i)$ and $b \in \text{leaves}(v_j)$, we say *a precedes b* in the partition ordering if $i<j$, and *a follows b* if $i>j$.

LEMMA 2.23. *Suppose E is segregated in T, and $z=\text{lca}_T(E)$ is a Q-node or an R-node. Suppose a precedes b in T's partition ordering of E. Then*

$$\text{CYC}_{\text{fix}\,z}(T) \mid E = \{\lambda \in \text{CYC}(T) \mid E : \textit{a precedes b in the ordering } \lambda\}. \tag{17}$$

Proof. Let the children of z in T be $v_1, \ldots, v_s$ in order. Suppose $T' \cong T$, and let $\lambda_i = \text{fr}_{T'}(v_i)$ for $i=1, \ldots, s$. Then $\text{cycfr}(T') \mid E = \text{fr}_{T'}(z)$ is either $\lambda_1 \cdots \lambda_s$ or $\lambda_s \cdots \lambda_1$, depending on whether $T'\ \text{flip}_z\ T$ is *false* or *true*. Since a precedes b in $\lambda_1 \cdots \lambda_s$ and follows b in $\lambda_s \cdots \lambda_1$, this proves that $T'\ \text{flip}_z\ T = \textit{false}$ iff a precedes b in $\text{cycfr}(T') \mid E$. ∎

Using Lemma 2.23, we can test if a given $\lambda \in \text{CYC}(T) \mid E$ is in $\text{CYC}_{\text{fix}\,z}(T) \mid E$; if a and b are in different blocks of T's partition of E, they serve as a "test pair" for E. Note that unless z is a leaf, it has at least two children, so there exists such a test pair.

LEMMA 2.24. *Suppose E is segregated and rigid in T, and $z=\text{lca}(E)$. Then* $\text{CYC}_{\text{fix}\,z}(T) \mid D \cap \text{CYC}_{\text{flip}\,z}(T) \mid D = \varnothing$.

Proof. By definition of rigidity, z is an R-node, and hence has a parent $p(z)$, which in turn must have at least one other child u. Assume without loss of generality that u is immediately to the left of z as a child of $p(z)$ in T. If $\text{leaves}(u)$ is strictly contained in D, we choose $x \in \text{leaves}(u)$ and $y \in D - \text{leaves}(u)$. If $\text{leaves}(u) = D$, then $p(z)$ is the root of T. Moreover, condition 2 of the definition of rigidity implies that u is an R-node; we choose x to be a leaf of the rightmost child of u, and choose y to be a leaf of some other child of u.

Then for any PQ-tree $T' \cong T$ such that $T'\ \text{flip}_z\ T = \textit{false}$, we have $T'\ \text{flip}_{p(z)}\ T = \textit{false}$ as well, because z is an R-node, and additionally $T'\ \text{flip}_u\ T = \textit{false}$ if u is an R-node. Then $\text{fr}(T')$ has the form $\alpha\beta\gamma$, where α contains x, β consists of the elements of E, and either y precedes x in α or y is in γ. In either case, y precedes x in $\text{cycfr}(T') \mid D$. Similarly, we can show that if $T'' \cong T$ but $T''\ \text{flip}_z\ T = \textit{true}$, y follows x in $\text{cycfr}(T'') \mid D$, so $\text{cycfr}(T') \mid D \neq \text{cycfr}(T'') \mid D$. This proves the lemma. ∎

In the above proof, x and y serve as a "test pair" for D in the same way that a and b served as a "test pair" for E in the proof of Lemma 2.23.

We now make use of the new terminology and associated lemmas in showing how to compute the join of T_0 with $T_1, \ldots, T_k$. We assume for the following procedure that $T_0, \ldots, T_k$ are PQ-trees over the ground sets $S_0, \ldots, S_k$ respectively, and that $S_1, \ldots, S_k$ are disjoint. We also assume that for $j=1, \ldots, k$, $E_j = S_0 \cap S_j$ is

non-empty and contiguous in T_0 and T_j, and $D_j = S_j - E_j$ is also non-empty and contiguous in T_j.

We first describe a method for computing the join of T_0 with $T_1, ..., T_k$ in $O(\log n)$ time using n processors (where $n = \sum_0^k |S_j|$), assuming that the join is *not* the null tree. This is called the "provisional" join, because it is the correct join *provided* the correct join is not T_{null}. Then we show how to verify the correctness of the provisional join—that is, determine whether the correct join is T_{null}—in $O(\log^2 m)$ time using m processors, where $m = \sum_1^k |E_j|$. For this, we use the PQ-tree intersection algorithm of Subsection 2.3.

JOIN. For $j = 1, ..., k$ in parallel:

J1 Segregate T_0 and T_j with respect to E_j. If $|E_j| = 1$, let $\hat{T}_j$ be T_j and skip to step J4.

J2 Find two elements a_j and b_j of E_j that lie in different blocks of T_0's partition of E_j and different blocks of T_j's partition of E_j. Assume that a_j precedes b_j in the frontier of T_0; otherwise, swap a_j and b_j.

J3 Obtain the PQ-tree $\hat{T}_j$ from T_j as follows: if b_j precedes a_j in the frontier of T_j, then let $\hat{T}_j$ be T_j. Otherwise (if b_j follows a_j), let $\hat{T}_j$ be the tree obtained from T_j by flipping every node. This ensures that b_j precedes a_j in the frontier of $\hat{T}_j$.

J4 Segregate D_j in $\hat{T}_j$. Replace the subtree $T_0 \mid E_j$ of T_0 by $\hat{T}_j \mid D_j$, letting z be the root of $\hat{T}_j \mid D_j$ in the resulting tree. If E_j is rigid in both T_0 and T_j, then rename z to be an R-node. If E_j is hinged in T_0 but z is an R-node, rename it to be a Q-node. Let T_+ be the resulting PQ-tree.

Before giving the proof of correctness of the *join* procedure, we discuss the implementation of step J2.

LEMMA 2.25. *Let π, ρ be two partitions of E, and assume that each is non-trivial (i.e., has at least two blocks). Then there exist two elements a_1 and b_1 of E that are in different blocks of π and different blocks of ρ.*

Proof. First let a be any element of E; it belongs to some block B_1 of π and C_1 of ρ. If there is an element $b \in E$ that lies outside of both B_1 and C_1, then we may let $a_1 = a$ and $b_1 = b$, satisfying the lemma. Otherwise, every element of $E - B_1$ lies in C_1. Then every block of π other than B_1 is contained in C_1. Let B_2 be such a block, and let $a_1 \in B_2$. Let b_1 be any element of $E - C_1$; then a_1 and b_1 satisfy the lemma. ∎

Note that the proof of Lemma 2.25 may be easily implemented in parallel to find a suitable pair a_1, b_1, in $O(\log |E|)$ time using $|E|$ processors. Hence each step in the JOIN procedure takes at most $O(\log n)$ time using n processors, where $n = \sum_0^k |S_j|$.

The *verification* of the previsional join is as follows. For $j = 1, ..., k$ in parallel, assuming T_0 and T_j are segregated with respect to E_j, we intersect $T_0 \mid E_j$ with

$T_j \mid E_j$. If the result of any of the intersections is a null tree, the correct join is the null tree. Otherwise, the provisional join is the correct join.

We commence the proof of correctness of the join procedure. For the sake of simplified notation, we will assume that $k = 1$. However, the proof directly generalizes to $k \geqslant 1$. We let E denote $E_1 = S_0 \cap S_1$, and let D_0 denote $S_0 - E$. Note that D_1 is contiguous and segregated in T_+, the tree resulting from J4, because $\text{lca}_{T_+}(D_1) = z$.

For $i = 0$, 1, and any linear ordering λ of E, let

$$A_i^\lambda = \{\sigma_i \mid D_i : \sigma_i \in \text{CYC}(T_i) \text{ and } \sigma_i \mid E = \lambda\}.$$

The following lemma states that the join procedure is correct.

LEMMA 2.26. *Let T_+ be the tree resulting from step* J4. *Then either*

$$\text{CYC}(T_+) = \text{splice}\left(\bigcup_\lambda A_0^\lambda \times A_1^{\lambda^R}\right) \tag{18}$$

or a null tree arises in the intersection performed in the verification step, in which case the right-hand side of (18) *is empty.*

We start by proving the following claim:

CLAIM. *The join is null (the right-hand side of* (18) *is empty) iff a null tree is the result of the intersection performed in the verification step.*

Proof. For the right-hand side of (18) to be non-empty, there must be some ordering λ of E such that A_0^λ and $A_1^{\lambda^R}$ are non-empty. Suppose there were such a λ. Then there exist $T_0' \cong T_0$ and $T_1' \cong T_1$ such that $\text{cycfr}(T_0') \mid E = \lambda = (\text{cycfr}(T_1'))^R$. If we let T_1'' be the tree obtained from T_1' by flipping every node, it follows that $\text{fr}(T_0' \mid E) = \lambda = \text{fr}(T_1'' \mid E)$, proving that the intersection of $T_0 \mid E_1$ with $T_j \mid E_1$ is *not* the null tree.

Next suppose that the intersection yields a non-null tree with frontier λ. Then there exist $T_0^* \cong T_0 \mid E$ and $T_1^* \cong T_1 \mid E$ such that $\text{fr}(T_0^*) = \lambda = (\text{fr}(T_1^*))^R$. If $z_0 = \text{lca}_{T_0}(E)$ is an R-node, we assume that $T_0^* \text{ flip}_{z_0} T_0 \mid E = \textit{false}$ (for otherwise consider λ^R). Obtain a tree T_0' from T_0 by ordering the children of each node of $T_0' \mid E$ as these children are ordered in T_0^*. Then $T_0' \cong T_0$; this holds even if z_0 is an R-node, because in this case z_0 did not have to be flipped to obtain T_0' from T_0. We can similarly obtain T_1' from T_1 (although in this case if $z_1 = \text{lca}_{T_1}(E)$ is an R-node, z_1's parent may have to be flipped to ensure $T_1' \cong T_1$). We then have $\text{cycfr}(T_0') \mid D_0 \in A_0^\lambda$ and $\text{cycfr}(T_1') \mid D_1 \in A_1^{\lambda^R}$, showing that the right-hand side of (18) is not empty. ∎

We now prove the correctness of the *provisional* join procedure, assuming that

the join is not empty. First suppose that E is hinged in T_0. Then by Corollary 2.5, A_0^λ is independent of λ and is always equal to $\mathrm{CYC}(T_0) \mid D_0$, so

$$\bigcup_\lambda A_0^\lambda \times A_1^{\lambda^R} = \mathrm{CYC}(T_0) \mid D_0 \times \bigcup_\lambda A_1^{\lambda^R}$$
$$= \mathrm{CYC}(T_0) \mid D_0 \times \mathrm{CYC}(T_1) \mid D_1.$$

Similarly if E is hinged in T_1. But if E is hinged in either T_0 or T_1, then D_1 is hinged in T_+, the tree resulting from step J4. Therefore, by Corollary 2.5,

$$\mathrm{CYC}(T_+) = \mathrm{splice}(\mathrm{CYC}(T_0) \mid D_0 \times \mathrm{CYC}(T_1) \mid D_1)$$

and we are done.

Now suppose that E is rigid in both T_0 and T_1. Let $z_0 = \mathrm{lca}_{T_0}(E)$, and let $z_1 = \mathrm{lca}_{T_1}(E)$. In this case, z_0 and z_1 are R-nodes. Also, D_1 is rigid in the tree T_+. By Corollary 2.6,

$$\mathrm{CYC}_{\mathrm{fix}\,z}(T_+) = \mathrm{splice}(\mathrm{CYC}_{\mathrm{fix}\,z_0}(T_0) \mid D_0 \times \mathrm{CYC}_{\mathrm{fix}\,z_1}(\hat{T}_1) \mid D_1)$$
$$\mathrm{CYC}_{\mathrm{flip}\,z}(T_+) = \mathrm{splice}(\mathrm{CYC}_{\mathrm{flip}\,z_0}(T_0) \mid D_0 \times \mathrm{CYC}_{\mathrm{flip}\,z_1}(\hat{T}_1) \mid D_1).$$

If we could show that

$$\bigcup_\lambda A_0^\lambda \times A_1^{\lambda^R} = (\mathrm{CYC}_{\mathrm{fix}\,z_0}(T_0) \mid D_0 \times \mathrm{CYC}_{\mathrm{fix}\,z_1}(\hat{T}_1) \mid D_1)$$
$$\cup (\mathrm{CYC}_{\mathrm{flip}\,z_0}(T_0) \mid D_0 \times \mathrm{CYC}_{\mathrm{flip}\,z_1}(\hat{T}_1) \mid D_1), \tag{19}$$

we would thereby show that $\mathrm{CYC}(T_+) = \mathrm{splice}(\bigcup_\lambda A_0^\lambda \times A_1^{\lambda^R})$, and would be done.

To prove (19), we start by showing that the left-hand side of (19) contains the right-hand side.

Let λ be any ordering of E in $\mathrm{CYC}(T_0) \mid E \cap \mathrm{CYC}(T_1) \mid E$ such that a_1 precedes b_1 in λ. (Since $\mathrm{CYC}(T_0) \mid E \cap \mathrm{CYC}(T_1) \mid E$ is closed under reversal and we assume it is non-empty, there exists such a λ.) By Lemma 2.23, $\lambda \in \mathrm{CYC}_{\mathrm{fix}\,z_0}(T_0) \mid E$. By Corollary 2.6, $\mathrm{CYC}_{\mathrm{fix}\,z_0}(T_0) = \mathrm{splice}(\mathrm{CYC}_{\mathrm{fix}\,z_0}(T_0) \mid D_0 \times \mathrm{CYC}_{\mathrm{fix}\,z_0}(T_0) \mid E)$. Hence for any $\alpha \in \mathrm{CYC}_{\mathrm{fix}\,z_0}(T_0) \mid D_0$, $\mathrm{splice}(\alpha, \lambda) \in \mathrm{CYC}_{\mathrm{fix}\,z_0}(T_0)$, proving that $\alpha \in A_0^\lambda$. This shows $\mathrm{CYC}_{\mathrm{fix}\,z_0}(T_0) \mid D_0 \subseteq A_0^\lambda$. Similarly $\lambda^R \in \mathrm{CYC}_{\mathrm{fix}\,z_1}(\hat{T}_1) \mid E$, so we have

$$\mathrm{CYC}_{\mathrm{fix}\,z_0}(T_0) \mid D_0 \times \mathrm{CYC}_{\mathrm{fix}\,z_1}(\hat{T}_1) \mid D_1 \subseteq A_0^\lambda \times A_1^{\lambda^R}.$$

An analogous argument shows that $\mathrm{CYC}_{\mathrm{flip}\,z_0}(T_0) \mid D_0 \times \mathrm{CYC}_{\mathrm{flip}\,z_1}(\hat{T}_1) \mid D_1 \subseteq A_0^{\lambda^R} \times A_1^\lambda$.

It remains to prove that the right-hand side of (19) contains the left-hand side. Every element of $\bigcup_\lambda A_0^\lambda \times A_1^{\lambda^R}$ is of the form $\langle \sigma_0 \mid D_0, \sigma_1 \mid D_1 \rangle$, where $\sigma_0 \in \mathrm{CYC}(T_0)$, $\sigma_1 \in \mathrm{CYC}(T_1)$, and $\sigma_0 \mid E = \lambda = (\sigma_1 \mid E)^R$ for some ordering λ of E.

Suppose a_1 precedes b_1 in λ. Then $\sigma_0 \in \mathrm{CYC}_{\mathrm{fix}\,z_0}(T_0)$ and $\sigma_1 \in \mathrm{CYC}_{\mathrm{fix}\,z_1}(\hat{T}_1)$ by Lemma 2.23. On the other hand, if a_1 follows b_1 in λ, then $\sigma_0 \in \mathrm{CYC}_{\mathrm{flip}\,z_0}(T_0)$ and

$\sigma_1 \in \mathrm{CYC}_{\mathrm{flip}\, z_1}(\hat{T}_1)$. This proves that the right-hand side of (19) contains the left-hand side, and we are done. This completes the proof of correctness of the join procedure. ▮

The following lemma is used in finding a planar embedding, once the planarity algorithm of Section 3 succeeds.

LEMMA 2.27. *Given $\sigma \in \mathrm{CYC}(T_+)$, we can use the intersections trees computed in the verification step to choose $\sigma_i \in \mathrm{CYC}(T_i)$ for $i = 0, \ldots, k$ such that $\sigma = \sigma_0$ join σ_1 join $\cdots$ join σ_k and $\sigma_0 \mid E_i = (\sigma_i \mid E_i)^R$.*

Proof. As before, assume for simplicity that $k = 1$. We have $\sigma_1 \mid D_1 = \sigma \mid D_1$ by definition of $\sigma = \sigma_0$ join σ_1. We need to choose an ordering $\sigma_1 \mid E_1 = (\sigma_0 \mid E_1)^R$ such that $\sigma_1 \in \mathrm{CYC}(T_1)$ and $\sigma_0 \in \mathrm{CYC}(T_0)$. The frontier of the intersection tree is a linear ordering λ_1 of E_1 such that λ_1—and hence also λ_1^R—are members of $\mathrm{CYC}(T_0) \mid E_1 \cap \mathrm{CYC}(T_1) \mid E_1$. Thus λ_1 and λ_1^R are two candidates for the desired ordering of E_1.

If E_1 is hinged in both T_0 and T_1, the choice between λ_1 and λ_1^R is arbitrary by Corollary 2.5—either choice will work. Suppose E_1 is rigid in T_0 or T_1. (If both, we need only work from one; the correctness of T_+ ensures that the result will be consistent with the other.) Say E_1 is rigid in T_1. To determine if λ_1 belongs to $\mathrm{CYC}_{\mathrm{fix}\, z_1}(\hat{T}_1) \mid E_1$, we can use the test pair a_1, b_1 as in Lemma 2.23. To determine if $\sigma_1 \mid D_1$ belongs to $\mathrm{CYC}_{\mathrm{fix}\, z_1}(\hat{T}_1) \mid D_1$ or to $\mathrm{CYC}_{\mathrm{flip}\, z_1}(\hat{T}_1) \mid D_1$, we use another test pair as in the proof of Lemma 2.24. If $\lambda_1 \in \mathrm{CYC}_{\mathrm{fix}\, z_1}(\hat{T}_1) \mid E_1$ and $\sigma_1 \mid D_1 \in \mathrm{CYC}_{\mathrm{fix}\, z_1}(\hat{T}_1) \mid D_1$, then we let $\sigma_1 = \mathrm{splice}(\langle \sigma_1 \mid D_1, \lambda_1 \rangle)$ so $\sigma_1 \in \mathrm{CYC}_{\mathrm{fix}\, z_1}(\hat{T}_1)$. Similarly if $\lambda_1 \in \mathrm{CYC}_{\mathrm{flip}\, z_1}(\hat{T}_1) \mid E_1$ and $\sigma_1 \mid D_1 \in \mathrm{CYC}_{\mathrm{flip}\, z_1}(\hat{T}_1) \mid D_1$. Otherwise, we let $\sigma_1 = \mathrm{splice}(\langle \sigma_1 \mid D_1, \lambda_1^R \rangle)$. In either case, σ_0 is then determined by the requirement that $\sigma_0 \mid E_1 = (\sigma_1 \mid E_1)^R$. ▮

We conclude this subsection with a technical lemma that is used in the planarity algorithm.

LEMMA 2.28. *Suppose that T_+ is the join of T_0 with $T_1, \ldots, T_k$, obtained using the procedure of this subsection. Let E_j be the intersection of the ground set S_0 of T_0 with the ground set S_j of T_j, for $j = 1, \ldots, k$:*

(1) *Suppose a proper subset A of S_0 is contiguous in T_0, and each $E_j \subset A$. Then $(A - \bigcup_1^k E_j) \cup \bigcup_1^k (S_j - E_j)$ is contiguous in T_+.*

(2) *Suppose a subset A_j of $S_j - E_j$ is contiguous in T_j. Then A_j is contiguous in T_+.*

Proof. The lemma follows from the procedure. To prove (1), assume for simplicity that A is segregated in T_0, and let $y = \mathrm{lca}_{T_0}(A)$. Since $E_j \subset A$, $\mathrm{lca}_{T_0}(E_j)$ is a descendent of y. In carrying out the join, we substitute $T_j \mid (S_j - E_j)$ for $T_0 \mid E_j$, so $\mathrm{lca}_{T_+}(S_j - E_j)$ becomes a descendent of y, and $\mathrm{leaves}_{T_+}(y) = (A - \bigcup_1^k E_j) \cup \bigcup_1^k (S_j - E_j)$.

To prove (2), assume for simplicity that A_j is segregated in T_j, and let $y = \mathrm{lca}_{T_j}(A_j)$. Then y is a node in $T_j \mid (S_j - E_j)$, and therefore y and its subtree appear in T_+. Thus $\mathrm{leaves}_{T_+}(y) = A_j$. ∎

3. Planarity Testing

3.1. *Preliminaries*

The graphs we discuss in this section will typically be "multigraphs," i.e., there may be many edges sharing the same endpoints. Moreover, we need to manipulate these graphs in ways that preserve the identity of edges while changing their endpoints. We therefore use a non-standard definition of a graph, in which an edge is not determined by its endpoints:

Definition 3.1. A graph G is a pair $G = \langle \rho, E \rangle$, where $E = \{e_1, ..., e_m\}$ is a finite set, the set of *edges* of G, and ρ is a partition of the set $E \times \{0, 1\}$ of *darts*. Each block v of the partition ρ is a *node*; an edges e is *incident* to a node $v \in \rho$ if v contains a dart of e, i.e. if $\langle e, 0 \rangle \in v$ or $\langle e, 1 \rangle \in v$. Paths, cycles, connectivity, etc. can be defined in terms of edge-node incidence. However, because a block of a partition is by definition non-empty, our definition of a graph disallows isolated points. We therefore assume in this section that a graph has no isolated points.

For any edge $e \in E$ and $\delta \in \{0, 1\}$, the pair $\langle e, \delta \rangle$ is a dart of e; the *other* dart is then $\langle e, 1 - \delta \rangle$. We define the permutation *other*: $(E \times \{0, 1\}) \to (E \times \{0, 1\})$ of the darts of G by $\mathrm{other}[\langle e, \delta \rangle] = \langle e, 1 - \delta \rangle$. For notational convenience, if d is a dart, we denote $\mathrm{other}(d)$ by d^R.

The ordinary definitions of graph embeddings and planarity are topological. However, a technique commonly used in the literature (see p. 22 of [21]) is the combinatorial representation of an embeding. We make use of this technique.

For a graph $G = \langle \rho, E \rangle$, an *embedding* of G is a permutation π of the set of darts $E \times \{0, 1\}$ whose orbits are exactly the blocks of ρ. Thus for every node $v \in \rho$, π determines a cycle on the set of darts belonging to v. We say $\langle \pi, E \rangle$ is an *embedded graph*, and we typically denote it by G_π. Its *underlying graph* is G. To define the *faces* of the embedded graph, we define another permutation π^* of the set of darts by composing π with *other*, so $\pi^* = \pi \circ \mathrm{other}$. Then the *faces* of the embedding π are the cycles of π^*.

As described in, e.g., Theorem 3.5 of [21], there is a correspondence between (combinatorial) embeddings and topological embeddings onto surfaces in which (intuitively) the darts incident to a node are cyclically arranged around that node clockwise in the sequence determined by the corresponding cycle of π. Moreover, if G is connected, our combinatorial "faces" are the boundaries of the faces of a corresponding topological embedding; in particular, the number of faces is just the number of orbits of π^*.

For an embedded graph G_π, we say that π is a *planar* embedding and G_π is a *planar embedded graph* if this version of Euler's formula is satisfied,

$$m - n + 2c = f \tag{20}$$

where m = number of edges of G_π, n = number of nodes, c = number of connected components, and f = number of faces. It follows from Theorem 3.5 of [21] that an embedding which is planar according to this definition corresponds to a topological planar embedding, and vice verse. A graph is planar if there exists a planar embedding.

We define the dual of an embedded graph $G_\pi = \langle \pi, E \rangle$ to be the embedded graph $(G_\pi)^* = \langle \pi^*, E \rangle$. Thus the edges of $(G_\pi)^*$ are the edges of G and its nodes are the faces of G_π. If G_π is connected and planar embedded, our definition of the dual corresponds to the topological definition. Since the topological dual is connected, it follows that if G_π is connected, the dual $(G_\pi)^*$ is connected.

FACT 3.1. *The dual of the dual of G_π is G_π.*

Proof. $(\pi^*)^* = (\pi \circ \text{other})^* = \pi \circ \text{other} \circ \text{other} = \pi$. ∎

FACT 3.2. *G_π is planar iff its dual is planar.*

Proof. Taking the dual exchanges the number of nodes with the number of faces. Substituting into (20) gives the result. ∎

Two natural operations on graphs are deletion and contraction. To *delete* an edge e from the graph $G = \langle \rho, E \rangle$ is to remove e from E, remove e's darts from ρ, and, if necessary, eliminate any resulting isolated nodes. We denote the result by $G - e$. To *contract* an edge e is to identify the endpoints of the edge e (i.e., union the blocks containing darts of e), and remove e from E and e's darts from ρ. (Again any resulting isolated nodes are eliminated.) We denote the result by G/e.

The operation of deletion of an edge e is also applicable to an embedded graph $G_\pi = \langle \pi, E \rangle$. The embedded graph $G_\pi - e$ is defined to be $\langle \pi', E - \{e\} \rangle$, where π' is obtained from π by removing e's darts. (To remove a dart d_1 from a permutation π, we write π as the product of its cycles, and remove d_1 from the cycle in which it appears, i.e., the cycle $(d_1 d_2 \cdots d_k)$ is replaced with $(d_2 \cdots d_k)$. Thus the cyclic order among the remaining darts in the cycle is preserved.)

FACT 3.3. *Deleting an edge from a planar embedded graph results in a planar embedded graph.*

Although the fact can be proved combinatorially, it follows directly from the topological definition of an embedding.

We will define the operation of contraction as applied to an embedded graph in terms of deletion in the dual. Say the contraction of an edge e in a graph G is a

proper contraction if e is not a self-loop in G. This definition is motivated by the following lemma.

LEMMA 3.1. *Suppose G_π is an embedded graph in which e is not a self-loop. Then the underlying graph of the dual $(G_\pi - e)^*$ is the graph obtained from the underlying graph of the dual $(G_\pi)^*$ by contracting the edge e.*

Note. The case in which G_π is planar embedded was proved by Whitney [22] for his version of a combinatorial dual.

Proof. Suppose $\pi[d_1] = d$ and $\pi[d^R] = d_2$. Then $\pi^*[d_1^R] = d$ and $\pi^*[d] = d_2$. If we delete the dart d from π^*, obtaining $\pi^{*\prime}$, we change the value of the function π^* only at d_1^R. Namely, $\pi^{*\prime}[d_1^R] = d_2$ but for any other dart $d_3 \neq d$, $\pi^{*\prime}[d_3] = \pi^*[d_3]$. Now let $\pi' = \pi^{*\prime} \circ \text{other}$. We have $\pi'[d_1] = d_2$ but for any other dart $d_3 \neq d^R$, $\pi'[d_3] = \pi[d_3]$.

Let d and d' be the darts of e, and suppose the (distinct) cycles of π containing these darts (i.e., the endpoints of e) are $\sigma = (d\, a_1\, a_2 \cdots a_k)$ and $\sigma' = (d'\, b_1\, b_2 \cdots b_l)$. By the above remarks, if $\pi^{*\prime}$ is obtained from π^* by deleting d and d', and $\pi' = (\pi^{*\prime})^*$, then π' contains the cycle $(a_1 \cdots a_k\, b_1 \cdots b_l)$ instead of σ and σ', but is otherwise identical to π. Thus the collection of darts $(\neq d, d')$ previously belonging to the endpoints of e now belong to a single node. ∎

DEFINITION 3.2. The result of contracting the edge e of the connected embedded graph G_π is defined to be $((G_\pi)^* - e)^*$, and is denoted by G_π/e.

By Lemma 3.1, if the contraction of e in G_π is proper, the underlying graph of G_π/e is the same as G/e, where G is the underlying graph of G_π. By Facts 3.2 and 3.3, if G_π is planar, G_π/e is planar.

DEFINITION 3.3. If G is a graph and the graph G' can be obtained from G by deletions and proper contractions, we say G' is a *minor* of G. If π is an embedding of G, then applying the deletions and contractions to the embedded graph G_π yields an embedded graph $G'_{\pi'}$. We say in this case that $G'_{\pi'}$ is an *embedded minor of* G_π.

By deletions and contractions of edges, a connected subgraph H of a planar embedded graph G_π can be reduced to a single node h. The resulting planar embedding associates a cycle σ with the node h. This cycle describes how the edges of $G - H$ were arranged around H in the original embedded graph G_π. We will use such cycles to represent embeddings in the planarity algorithm of Subsection 3.2. Here we develop machinery enabling us to show that under certain circumstances the cycle σ is determined uniquely by the embedded graph G_π, regardless of which edges of H are deleted and contracted to reduce H to a single node h.

We say a node v of a graph G is an *articulation point* if there are two edges such that any path connecting endpoints of the two edges also contains v. For example, if v has a self-loop in a graph containing at least two edges, then v is an articulation point.

We now state the two main results of this section.

UNIQUENESS LEMMA. *Suppose G_π is a connected planar embedded graph with a minor G'. There is a unique embedded minor of G_π having G' as underlying graph, if the following condition is satisfied:*

> *For any node v of G' resulting from the identification of at least two nodes, of G, v is* not *an articulation point.*

For example, if G' is biconnected, the condition is satisfied.

For the next lemma, refer to the definition of the operator *join*, Definition 2.8.

PLANAR CONTRACTION LEMMA. *Let G_π be a connected embedded graph. Let x and y be two adjacent nodes of G, and let A be the set of edges between x and y. Suppose $G-x-y$ is connected. Let σ_x and σ_y be the cycles of π corresponding to x and y, respectively. Then π is a planar embedding of G iff the following conditions are satisfied:*

- *the darts of edges of A are consecutive subsequences τ_x and τ_y in σ_x and σ_y,*
- *the order of darts of τ_x is the reverse of the order of the corresponding darts in τ_y, and*
- *the embedded graph $G'_{\pi'}$ is planar, where $G'_{\pi'}$ is obtained from G_π by deleting all but one edge of A and contracting the remaining edge, identifying x and y to form z.*

Moreover, if the first two conditions are satisfied, the cycle σ_z associated with z in the embedded graph $G'_{\pi'}$ is σ_x join *σ_y.*

We postpone the proofs of the Uniquness Lemma and the Planar Contraction Lemma until the end of this subsection.

DEFINITION 3.4. For a node-induced subgraph H, we call an edge e of G an *H-linking edge* if exactly one endpoint of e is in H. The endpoint not in H is called the edge's *outside endpoint*. Let link(H) be the set of H-linking edges.

DEFINITION 3.5. Let G be a connected graph. We say that a subgraph H of G is *bound in G* if H is node-induced and connected, and $G-H$ is connected. Given the connected graph G and a subgraph H bound in G, obtain the graph G/H from G by *contracting the subgraph H to a node h*, which we define by the following procedure: choose a spanning tree T of H and contract the edges of T. (Note that each contraction is proper.) At this point, H has been contracted to a single node h with some self-loops. Next, delete all self-loops of h. Let G/H denote the result. Note that the edges incident to h in G/H are the H-linking edges. Also, note that G/H is independent of the choice of spanning tree T. Moreover, $G/H-h$ is connected because $G-H$ is connected. Hence by the Uniqueness Lemma, for any planar

embedding π of G, there is a unique embedding π' of G/H. We denote the resulting embedded graph by G_π/H.

For each planar embedding π of G, we obtain a cycle of link(H), namely the cycle associated with h in the embedding of G_π/H. We call this cycle the *embedding rotation of H in G_π*. More generally, a cycle of link(H) is an embedding rotation of H in the (unembedded) graph G if it is the embedding rotation of H in G_π for *some* planar embedding π. In the planarity algorithm, we represent the set of all embedding rotations of certain subgraphs H in G, thereby characterizing in part the set of embeddings of G. For example, if G is not planar, there are no embedding rotations of H in G.

Formally, the cycles of embeddings are cycles of darts, not of edges. However, here and in Subsection 3.2, we will at times ignore the distinction between darts and edges, and treat a dart as the edge it corresponds to. Thus, for example, an edge e appears in two cycles of an embedding—the cycles corresponding to the endpoints of e.

LEMMA 3.2. *Suppose G_π is a planar embedded graph with a bound subgraph H. Let $G'_{\pi'}$ be an embedded minor of G_π, and let H' be the subgraph of G' corresponding to H. If H' is bound in G' and* link(H') = link(H) *then the embedding rotation of H' in $G'_{\pi'}$ is the embedding rotation of H in G_π.*

Since every planar embedding π of G induces a planar embedding π' of G', it follows from the lemma that every embedding rotation of H in G is an embedding rotation of H' in G'.

Proof of Lemma 3.2 (see Fig. 9). Let U be the minor of G obtained by first contracting H to a node h, and then contracting $G-h$ to a node g. The embedding π of G induces an embedding $\hat{\pi}$ of U. The above way of obtaining $U_{\hat{\pi}}$ as an embedded minor of G_π shows that the cycle that $\hat{\pi}$ associates with h is the embedding rotation of H in G_π. Another way to obtain U from G is to first obtain G' from G and then obtain U from G'. The resulting embedding of U is also $\hat{\pi}$, by the Uniqueness Lemma. This way of obtaining $U_{\hat{\pi}}$ as an embedded minor of G shows that the cycle associated with h is the embedding rotation of H' in $G'_{\pi'}$. This proves the lemma. ∎

LEMMA 3.3. *Let G be a graph containing a bound subgraph H. Let $G' = G/H$ be the minor of G obtained by contracting H to a node h. Let G'' be a minor of G con-*

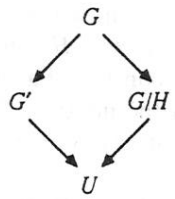

FIG. 9. The graph U is a minor of both G' and G/H. Moreover, G' and G/H are minors of G. Thus U can be obtained as a minor of G in two different ways.

taining H and all the H-linking edges. Let π' be a planar embedding of G' associating the cycle σ with h. Then two conditions are equivalent:

1. *There exists a planar embedding π of G such that $G'_{\pi'}$ is an embedded minor of G_π.*
2. *The cycle σ is an embedding rotation of H in G''.*

Proof. (1) $\Rightarrow$ (2) Suppose $G'_{\pi'}$ is an embedded minor of G_π. Let π'' be the embedding of G'' induced by the embedding π of G. Then by Lemma 3.2, the embedding rotation of H in $G''_{\pi''}$ equals the embedding rotation of H in G_π, which equals the embedding rotation of h in $G'_{\pi'}$, which is σ. This proves (2).

(2) $\Rightarrow$ (1) Suppose σ is an embedding rotation of H in G'', and let π'' be the corresponding planar embedding of G''. We obtain an embedding π of G from π' by replacing the cycle σ with the cycles π'' associates with nodes of H. Assume for a contradiction that π is not a planar embedding, and let $\hat{G}_{\hat{\pi}}$ be a minimal non-planar embedded minor of the embedded graph G_π. Let x and y be any two adjacent nodes of $\hat{G}$, and let A be the set of edges between them. Since $\hat{\pi}$ is not a planar embedding, it must violate one of the three conditions of the Planar Contraction Lemma. By minimality of $\hat{G}_{\hat{\pi}}$, the third condition is not violated; hence either the edges A are not consecutive in the cycles σ_x and σ_y associated with x and y, or they are consecutive but $\sigma_x \mid A \neq (\sigma_y \mid A)^R$. But if x and y are both in H, then this same violation holds for the planar embedded graph $G''_{\pi''}$. If x and y are not both in H, then the edges A between them exist in G', and the same violation holds in the planar embedded graph $G'_{\pi'}$. In either case, we have a contradiction. ∎

The remainder of this subsection contains proofs of the Uniqueness Lemma and the Planar Contraction Lemma. Understanding of these proofs is not vital for understanding the planarity algorithm.

Let G_π be a planar embedded graph, and let H be a bound subgraph. Define the permutation other_H of the darts of G by

$$\text{other}_H[d] = \begin{cases} \text{other}[d] & \text{if } d \text{ is a dart of an edge of } H \\ d & \text{otherwise.} \end{cases}$$

For a permutation π of the darts of G, define $\pi^H = \pi \circ \text{other}_H$. Note that $\pi^H = (\pi^*)^{G-H}$.

Define the *exfaces of H in G_π* to be those cycles of π^H that contain darts belonging to nodes of H. Note that each exface of H contains only darts of edges of H and H-linking edges. For example, if H consists of a single node v, then H has no edges. Hence for any dart d contained in the node v, $\pi^H(d) = \pi(d)$, which is another dart contained in v, so the only exface of H in G is the cycle of π corresponding to v.

LEMMA 3.4. (1) *If e is not an edge of H, then deleting e from G_π has the same effect as removing its darts from π^H.*

(2) *If e is an edge of H, then deleting e from G_π^* has the same effect as removing its darts from π^H.*

Proof. Let d be a dart of $e \notin H$. If π has the form $\pi = \cdots(\cdots d_1\ d\ d_2 \cdots)$ then π^H has the form $\pi^H = \cdots(\cdots \text{other}_H[d_1]\ d\ d_2 \cdots)$. Thus removing d from π leaves $\pi' = \cdots(\cdots d_1\ d_2 \cdots)$, so $(\pi')^H = \cdots(\cdots \text{other}_H[d_1]\ d_2 \cdots)$. Applying this observation twice, once for each dart of e, proves (1). Since $\pi^H = (\pi^*)^{G-H}$, (2) is the dual version of (1). ∎

A consequence of (1) is that the exfaces of H in G_π, with all edges of $G-H$ omitted (i.e., with their darts removed), are just the faces of H (with the induced embedding). For example, an embedded tree has only one face, so it has only one exface. The following lemma, which is a consequence of (2), is the principal purpose of exfaces.

LEMMA 3.5. *If T is a tree in G_π, and we contract the edges of T in G_π one by one, until T has become a single node t, the cycle corresponding to t in the resulting embedded graph is the exface of T in G_π, with edges of T omitted.*

The proof is by induction on the number of nodes of T, using (2) of Lemma 3.4.

The following lemma is a combinatorial analog of the Jordan Curve Theorem.

LEMMA 3.6. *Suppose H is connected, and darts $d, d' \in G-H$ are in different exfaces of H in G. Then every path between the node containing d^R and the node containing $(d')^R$ goes through a node of H.*

Proof. The lemma is trivial if $d' = d^R$, for then both the node containing d^R and the node containing $(d')^R$ are nodes of H. Suppose $F_1 = (d\ d_1\ d_2 \cdots d_s)$ and $F_2 = (d'\ d_1'\ d_2' \cdots d_t')$ are two different exfaces of H in G, with edges of $G-H-\{e, e'\}$ omitted, where e and e' are the edges associated with d and d'. Suppose there is a path between d and d' avoiding H, and let $d = b_1, b_2, \ldots, b_r = (d')^R$ be the corresponding sequence of darts; i.e., b_i^R and b_{i+1} are incident to the same node $v \notin H$, for $i = 1, \ldots, r-1$. Let G' be the union of H and the edges corresponding to $b_1, \ldots, b_r$. Then $(b_1 \cdots b_r\ d_1' \cdots d_t'\ b_r^R \cdots b_1^R\ d_1 \cdots d_s)$ is a face of G'. Thus the number of faces of G' is one fewer than the number of faces of H. But we added r edges and $r-1$ nodes to H to get G'. It follows that if H satisfies Euler's formula, then G' does not. This is a contraction, because G' can be obtained from G by deletions, which preserve planarity. ∎

We can use Lemma 3.6 to prove the Planar Contraction Lemma.

Proof of Planar Contraction Lemma. (⇐) Suppose that the first two conditions hold. Write $\sigma_x = (d_1 \cdots d_t\ \gamma)$ and $\sigma_y = (d_t^R \cdots d_1^R\ \gamma')$, where $d_1, \ldots, d_t$ are darts of edges in A, and γ and γ' are sequences of other darts. We then have faces $(d_i^R\ d_{i+1})$ for $i = 1, \ldots, t-1$. Deletion of all edges in A but one yields an embedded graph $\hat{G}_{\hat{\pi}}$ in which these $t-1$ faces have disappeared. Contraction of the remaining edge e does not change the number of faces but does reduce by one the number of

nodes and the number of edges. It follows that if the resulting graph $G'_{\pi'}$ satisfies Euler's formula, then so does G_π.

Moreover, if we let H be the subgraph of $\hat{G}$ consisting of x, y, and the edge e between them, then the exface of H in $\hat{G}_{\hat{\pi}}$ is $\sigma = (d_1\ y'\ d_1^R\ \gamma)$, where d_1 and d_1^R are the darts of e. Contracting the edge e is deleting e in the dual, which by part (2) of Lemma 3.4 transforms σ into $\sigma' = (\gamma'\ \gamma)$, which is σ_x join σ_y. But following contracting of e, H consists only of the node z. As observed before Lemma 3.4, the exface of a single node is the cycle associated with that node. Thus $\sigma_z = \sigma' = \sigma_x$ join σ_y.

($\Rightarrow$) Now suppose that π is a planar embedding of G. The third condition holds because deletions and proper contractions preserve planarity. Let H be the subgraph consisting of x, y, and the edges A between them. Let π' be the embedding of H induced by the embedding π of G (i.e., $H_{\pi'}$ is obtained from G_π be deleting all edges of $G-H$).

If $|A| = 1$, the first two conditions hold trivially. Suppose $A = \{e_1, e_2\}$. Then $H_{\pi'}$ has two faces. Because $G-H$ is connected, it follows from Lemma 3.6 that one of the two exfaces of H in G contains no darts of $G-H$, i.e., is just a face of $H_{\pi'}$, say $(\langle e_1, \delta_1\rangle \langle e_2, \delta_2\rangle)$. It follows that $\pi[\langle e_1, 1-\delta_1\rangle] = \langle e_2, \delta_2\rangle$ and $\pi[\langle e_2, 1-\delta_2\rangle] = \langle e_1, \delta_1\rangle$. This proves the first two conditions of the lemma.

Finally, suppose A contains more than two edges. If all but two edges of A are deleted from G_π, the remaining two edges satisfy the first and second conditions in the resulting graph, as we just proved. It follows that all the edges of A satisfy these conditions in G_π. ▮

We need one more lemma to prove the Uniqueness Lema.

LEMMA 3.7. *Let H be a node-induced connected subgraph of G, and let A be a subset of the H-linking edges. Suppose that for any two H-linking edges not in A, there is a path connecting endpoints of the two edges that avoids H. For any spanning tree T of H, let $\Phi(T)$ be the exface of T in G, with edges of $A \cup H$ omitted. Then $\Phi(T)$ is always the same, independent of choice of T.*

Proof. We first prove that $\Phi(T) = \Phi(T')$ for any two spanning trees T and T' such that $|T - T'| = 1$. Let $e \in T - T'$, and let $e' \in T' - T$. Then $T \cup T' = T \cup \{e'\}$ has a single cycle containing e and e'. Hence $T \cup T'$ has two faces, hence two exfaces in G. Let these two exfaces be $(d\ \alpha\ d'\ \beta)$ and $(d^R\ \gamma\ (d')^R\ \lambda)$, where d is a dart of e, d' is a dart of e', and α, β, γ, λ are sequences of darts. Then the exface of T in G is $F = (d\ \alpha\ d'\ \lambda\ d^R\ \gamma (d')^R\ \beta)$ while the exface of T' in G is $F' = (d'\ \beta\ d\ \gamma\ (d')^R\ \lambda\ d^R\ \alpha)$.

The exfaces of H contain only H-linking edges and edges of H. We assume that every pair of H-linking edges not in A are connected by a path avoiding H, so only one exface of $T \cup T'$ can contain darts not in $A \cup H$, by Lemma 3.6. Thus either α, β or γ, λ contain only darts of $A \cup H$. Then the cycles F and F' become identical when edges of $A \cup H$ are omitted. We have shown $\Phi(T) = \Phi(T')$.

We now prove the lemma for any two different spanning trees T and T', by

induction on $|T-T'|$. We proved the basis of the induction. Now suppose $|T-T'|>1$. Choose $e\in T-T'$, and let e' be an edge of $T'-T$ appearing in the unique cycle in $T'\cup e$. Let $T''=T\cup\{e\}-\{e'\}$. Then $|T''-T'|=1$ and $|T''-T|<|T'-T|$, so by the inductive hypothesis, $\Phi(T)=\Phi(T'')$ and $\Phi(T'')=\Phi(T)$. ∎

We restate and then prove the Uniqueness Lemma:

Suppose G_π is a connected planar embedded graph, and G' is a minor of G. Any sequence S of deletions and proper contractions that yields G' when applied to G yields the same embedding π' of G' when applied to G_π, if the following condition is satisfied:

> **connectivity condition.** *For any node v of G' resulting from the identification of at least two nodes of G, v is not an articulation point of G'.*

Proof of the Uniqueness Lemma. Assume the connectivity condition is satisfied. It is easy to see that the two operations *deletion* followed by *contraction* may be swapped in a sequence S without affecting the resulting embedding, so we may assume that all contractions happen before any deletion in each sequence S. The order among the deletions clearly does not matter to the resulting embeddings; moreover, since contractions are just deletions in the dual, the order among the contractions also does not matter.

Let S_1 and S_2 be two sequences of proper contractions followed by deletions that yield G' when applied to G. Let C_i be the set of edges contracted in S_i, and let D_i be the set of edges deleted in S_i, for $i=1, 2$.

Fix a node $v\in G'$, and let $H(v)=\{u\in G: \exists \text{ dart } d\in v\cap u\}$. Then the nodes of $H(v)$ are identified to form v in G', so the contracted edges C_i contain a tree T_i spanning $H(v)$ in G. (If $H(v)$ consists of a single node, the T_i contains no edges.)

The set of edges $A_i=(C_i\cup D_i)\cap\{\text{edges incident to } H(v)\}$ consists of those edges incident to $H(v)$ that are not incident to v, so $A_1=A_2$.

Let σ_i be the cycle that π_i associates with v. Our goal is to show that $\sigma_1=\sigma_2$. By Lemma 3.5, σ_i is the exface of T_i in G_π, with edges of A_1 omitted. If $H(v)$ consists of a single node u, the exface of T_i in G_π is just the cycle π associates with u, so $\sigma_1=\sigma_2$. Suppose therefore that $H(v)$ consists of at least two nodes. Then by our assumption that the connectivity condition holds, v is not an articulation point of G', so for any two edges incident to v, there is a path in G' connecting these edges that avoids v. Hence for any two edges incident to $H(v)$ and not in A_1, there is a path in G connecting these edges that avoids $H(v)$. Then by Lemma 3.7, the exface of T_1 in G_π with edges of A_1 omitted is the same as the exface of T_2 with edges of A_1 omitted. Thus $\sigma_1=\sigma_2$. ∎

3.2. *The Algorithm*

Our main theorem is

THEOREM 3.1. *A graph with n nodes and no multiple edges can be tested for planarity in $O(\log^2 n)$ time using n processors. If the graph is planar, a combinatorial representation of a planar embedding can be found within the same bounds.*

The basic strategy of our planarity-testing algorithm is to process the graph "from the bottom up," starting with embeddings of individual nodes and ending with embeddings of the whole graph. A basic step in the algorithm is combining embeddings of subgraphs to form an embedding of the larger subgraph. We cannot merely choose a single embedding for each subgraph, for the chosen embeddings of two subgraphs might be inconsistent, preventing the embeddings from being combined. Instead, we use PQ-trees to represent the set of all embeddings of each subgraph. Once the planarity-testing algorithm succeeds, a "top-down" process can obtain a combinatorial embedding of the graph from the PQ-trees just computed.

Note that it is sufficient to achieve $O(\log^2 n)$ time using $O(n)$ processors, for we can then reduce the number of processors by a constant factor at the expense of a constant factor increase in the time bound. Also, it follows from Euler's formula that if an n-node graph G with no multiple edges or self-loops has more than $3n$ edges, it is not planar, and our planarity-testing algorithm may immediately reject it. We assume therefore that G has $m \leqslant 3n$ edges; the processor bound for our algorithm will be $O(m) = O(n)$.

Note. In this section, we make use of the notation that identifies a set of nodes with the subgraph induced by that set of nodes.

The first step of the algorithm is to find the biconnected components of the input graph.

LEMMA 3.8 [19]. *A graph G on n nodes and m edges can be (edge) partitioned into its biconnected components in $O(\log^2 n)$ time on $n+m$ processors.*[6]

By the following lemma, the planarity of each biconnected component may be considered independently.

LEMMA 3.9 [22]. *A graph G is planar iff its biconnected components are planar. Moreover, the combinatorial representation of a planar embedding of G can be immediately obtained from the combinatorial representations of planar embeddings of its biconnected components.*

We therefore assume for the remainder of this section that G is biconnected.

The second step is to find an *st-numbering* for G. An assignment of distinct integers to the nodes of G is called an *st*-numbering[7] if two adjacent nodes s and t are the lowest and highest numbered, respectively, and every other node is adjacent to both a lower numbered and a higher numbered node. Note that an *st*-numbering

[6] This algorithm works in $O(\log n)$ time on a concurrent-write model of parallel computation. However, the algorithm may be run on a weaker model with exclusive-write at a $O(\log n)$ factor increase in the time bound, using a simulation result of Vishkin [20] and the fact that small integer sorting may be done in $O(\log n)$ time using n processors on this weaker model.

[7] As originally defined, an *st*-numbering was an assignment of integers from 1 to n to the n nodes, but we find it convenient to make this minor change.

of G induces a direction on the edges of G—namely, an edge points toward its higher numbered endpoint. Accordingly, we call an edge incident to v an *incoming* edge if its other endpoint is numbered lower than v and an *outgoing* edge if its other endpoint is numbered higher than v. Let $\text{in}(v)$ be the set of incoming edges of v, and let $\text{out}(v)$ be the set of outgoing edges. The important fact about an st-numbering is that in the resulting directed acyclic graph, for every node v, there is a directed path from s to t through v.

LEMMA 3.10 [13]. *G has an st-numbering iff G is biconnected and has at least two nodes.*

In [6], Even and Tarjan give a linear-time sequential algorithm for finding an st-numbering. This algorithm does not seem parallelizable. Fortunately, Maon, Schieber, and Vishkin have an efficient way to find an st-numbering in parallel, based in part on the parallel ear-decomposition technique of Lovasz [14].

THEOREM 3.2 [15]. *Given a biconnected graph G on n nodes and m edges, and an edge $\{s, t\}$, an st-numbering can be found in $O(\log^2 n)$ time on $m+n$ processors.*[8]

The remainder of our planarity algorithm may be viewed as a contraction process on the st-numbered graph, taking place over a series of stages. We start with the original st-numbered graph $G^{(0)} = G$. In stage $i+1$, we choose a collection of bound subgraphs of the graph $G^{(i)}$ in accordance with the st-numbering. We contract these subgraphs, and we update the st-numbering, producing the graph $G^{(i+1)}$.

We say a node $v \neq s, t$ of $G^{(i)}$ is *joinable* if v is adjacent to some node $u \neq s, t$ in $G^{(i)}$. In each stage, we contract some of the edges connecting joinable *nodes*, reducing the number of joinable nodes. We stop after stage i if $G^{(i)}$ contains no joinable nodes; let $\mathcal{F}$ be this last stage. Thus every node in $G^{(\mathcal{F})}$ except s and t is adjacent only to s and t.

For each node $v \in G^{(i)}$ and each $j \leqslant i$, we let $H^{(j)}(v)$ denote the subgraph of $G^{(j)}$ that was contacted over stages $j+1, \ldots, i$ to form v. We write $H(v)$ for $H^{(0)}(v)$. If $u \in H^{(j)}(v)$ for $v \in G^{(i)}$, we let $u^{(i)}$ denote v.

Note that $G^{(i)}$ is actually a multi-graph, not a graph. That is, $G^{(i)}$ may have multiple edges with the same endpoints. The reason is that two nodes adjacent to a common node u may have been identified to form a node v, in which case the node v will have two edges to u.

We choose our bound subgraphs to contract at each stage so that

- Neither s nor t is ever identified with any other node; i.e., $s^{(i)} = s$ and $t^{(i)} = t$ for all i.
- Only $O(\log n)$ stages are needed.
- The st-numbering is easy to update, following contraction of edges.

[8] See Note 6.

• For each node $v \neq s, t$ in $G^{(i)}$, the subgraph $H(v)$ permits a PQ-tree representation of the set of its embeddings.

We first show how the subgraphs are chosen and show that our method of choosing subgraphs has the first three properties. Then we describe the method for representing the set of embeddings of a subgraph with a PQ-tree, and we show how this representation is updated when edges are contracted. Finally, we show how to obtain an embedding of the original graph G.

To ensure that only $O(\log n)$ stages are needed, we use a sequence of four stages, called a *phase*, to reduce the number of joinable nodes by a factor of two. In particular, during a phase every joinable node is identified with some other joinable node. Thus, if $G^{(i)}$ is the graph immediately preceeding the beginning of a phase, then for any joinable node v of $G^{(i)}$, $|H^{(i)}(v^{(i+4)})| \geq 2$. This shows that each phase reduces the number of joinable nodes by a factor of two, so only $\lceil \log n \rceil$ phases $= 4\lceil \log n \rceil$ stages are needed.

A phase has two parts, an *s-rooted* part and a *t-rooted* part, and each part consists of two subparts, a *main* stage and a *clean-up* stage.

For the s-rooted part of a phase, we construct a spanning tree of $G^{(i)} - \{t\}$ rooted at s. By the definition of st-numbering, every node $v \in G^{(i)}$ other than s and t is adjacent to some lower numbered node and to some higher numbered node. For each such v, let its parent $p(v)$ be the highest numbered neighbor of v whose number is less than that of v. We thereby define a "multi-tree," a graph that would be a tree if multiple edges were identified. The root of the multi-tree is s. Using parallel pointer-jumping, compute for each node v the distance from s to v in the multi-tree. Call a node "even" or "odd," according to whether this distance is even or odd. In the main stage, we identify even nodes with their (necessarily odd) parents. In the clean-up stage, we identify odd leaves with their (necessarily non-leaf) parents, except for those leaves whose parent is s. In each case, we identify children with parent and assign to the resulting node the parent's number.

The t-rooted part of a phase is similar; the parent of v is chosen to be the lowest numbered neighbor of v with a higher number than v. Analogous properties hold of this part of the phase.

Note the following properties:

(a) There are no edges between children of the same parent.

(b) Edges are directed from parent to child during the s-rooted part and from child to parent during the t-rooted part.

(c) Each node adjacent to some lower numbered node other than s is identified with another node during one of the two stages in the s-rooted part. Each node adjacent to some higher numbered node other than t is identified with another node during one of the two stages in the t-rooted part.

(d) Let f be the st-numbering function. For any $v \in G^{(i)}$, $f(v^{(i+1)}) \leq f(v)$ if the ith stage belongs to the s-rooted part and $f(v^{(i+1)}) \geq f(v)$ if the ith stage belongs to the t-rooted part.

We will make use of properties (a) and (b) later when we show how to implement a stage. Property (c) shows that in each phase the number of joinable nodes is halved. Property (d) makes possible the following lemma.

LEMMA 3.11. *For any i, suppose that the numbering f of $G^{(i)}$ is an st-numbering. Then, the numbering f of $G^{(i+1)}$ is also an st-numbering.*

Proof. Let v be any node of $G^{(i+1)}$. Consider the case in which the ith stage belongs to the s-rooted part of a phase. Suppose $H^{(i)}(v) = \{v_1, ..., v_k\}$, where we assume without loss of generality that v_1 was a parent and the other nodes (if any) were children. By the inductive hypothesis, v_1 has an incoming edge (u, v_1) in $G^{(i)}$, so there is an edge $(u^{(i+1)}, v)$ in $G^{(i+1)}$. We have $f(u^{(i+1)}) \leqslant f(u)$, $f(u) < f(v_1)$ by definition of an incoming edge, so $f(u^{(i+1)}) < f(v_1)$. Since $f(v_1) = f(v)$ by choice of v_1, we conclude that $(u^{(i+1)}, v)$ is an incoming edge of v. It remains to show that v has an outgoing edge. But v_k has an outgoing edge (v_k, w) in $G^{(i)}$. There are two cases. If w was identified with its parent $p(w)$ in stage i, $f(w^{(i+1)}) = f(p(w)) > f(v_k)$ by choice of $p(w)$. If w was not identified with its parent in stage i, then $f(w^{(i+1)}) = f(w) > f(v_k) \geqslant f(v_k^{(i)})$. In either case, $(v_k^{(i+1)}, w^{(i+1)})$ is an outgoing edge of $v_k^{(i)} = v$. ∎

We now consider the representation of the set of embeddings of a subgraph. Fix a node $v \neq s, t$ in $G^{(i)}$. For $j \leqslant i$, let $\widehat{H^{(j)}}(v)$ be the graph obtained from $H^{(j)}(v)$ by adding in the nodes s and t, the edge $\{s, t\}$, and the $H^{(j)}(v)$-linking edges, identifying the outside endpoints of all incoming (outgoing) edges with s (t). (Note that this identification may cause some of the $H^{(j)}(v)$-linking edges to become multiple edges.) It follows from the st-numbering of $G^{(0)}$ that $\widehat{H^{(j)}}(v)$ is a minor of $G^{(0)}$. (See Definition 3.3 for the definition of *minor*.)

Because v was formed by contraction of $H^{(j)}(v)$, $H^{(j)}(v)$ is connected. The subgraph $\widehat{H^{(j)}}(v) - H^{(j)}$ consists of the nodes s and t and the edge between them, so it is also connected. It follows that $H^{(j)}(v)$ is bound in $\widehat{H^{(j)}}(v)$. By similar reasoning one can show that $H^{(j)}(v)$ is also bound in $G^{(0)}$.

We call the embedding rotations of $H(v)$ in $H^{(0)}(v)$ the *arrangements* of v. Note that the arrangements of v are cycles of $\text{link}(H(v)) = \text{link}(v) =$ the set of edges incident to v.

Note that by Lemma 3.2, we have

LEMMA 3.12. *Any embedding rotation of $H(v)$ in $G^{(0)}$ is an arrangement of v.*

From the Planar Contraction Lemma of Subsection 3.1, we may obtain

COROLLARY 3.1. *In any arrangement of v, the incoming edges* $\text{in}(v)$ *are consecutive (and hence the outgoing edges* $\text{out}(v)$ *are consecutive).*

We will use a *PQ*-tree $T(v)$ to represent the set of arrangements of v. Recall from

Subsection 2.1 that a PQ-tree T represents a set $\mathrm{CYC}(T)$ of cycles over its ground set. Recall from Definition 2.4 that a subset A of the ground set of T is said to be *contiguous* in T if T has either a node v such that $A = \mathrm{leaves}(v)$ or a consecutive subsequence $v_p \cdots v_q$ of the children of a Q-node such that $A = \bigcup_p^q \mathrm{leaves}(v_j)$. It follows from Lemma 2.1 that if A is contiguous in T, the elements of A form a consecutive subsequence in every cycle in $\mathrm{CYC}(T)$.

We say the PQ-tree $T(v)$ over the ground set $\mathrm{link}(H(v))$ is *valid* if

- $\mathrm{CYC}(T(v)) =$ the set of arrangements of v, and
- $\mathrm{in}(v)$ and $\mathrm{out}(v)$ are contiguous in $T(v)$.

If $v \in G^{(0)}$, we can directly construct a valid PQ-tree $T(v)$. For in this case $H(v) = \{v\}$, so any cycle of the edges incident to v is an arrangement of v, provided that the incoming edges are consecutive and the outgoing edges are consecutive. In this case, therefore, we let $T(v)$ be the tree (depicted in Fig. 10) whose root is a Q-node with two P-node children, v_{in} and v_{out}, where the children of v_{in} are the edges $\mathrm{in}(v)$ and the children of v_{out} are the edges $\mathrm{out}(v)$.

At every stage i, we compute the PQ-trees for each new node $v \in G^{(i+1)}$ in parallel from the PQ-trees for the nodes $H^{(i)}(v)$ identified to form v.

If a null tree arises as $T(v)$ for some node v, it follows that there are no arrangements of v, i.e., no embedding rotations of $H(v)$ in $\hat{H}(v)$, hence no embedding rotations of $H(v)$ in $G^{(0)}$ by Lemma 3.2. But then $G^{(0)}$ is not planar. We therefore halt the algorithm when a null tree arises.

Assume, on the other hand, that the contraction process continues until there are no joinable nodes remaining in $G^{(\mathcal{F})}$; every node of $G^{(\mathcal{F})}$ other than s and t is adjacent only to s and t. Let $v_1, \ldots, v_k$ be these nodes. Since they are not joinable, there are no edges between them, so no edges between the corresponding $H(v_1), \ldots, H(v_k)$. For $j = 1, \ldots, k$, if $T(v_j)$ is not T_{null}, then there is a planar embedding π_j of $\hat{H}(v_j)$. Using the fact ([16]) that a graph G is planar iff its triconnected components are, it follows that $G^{(0)}$ is planar, for s and t form a separation pair whose blocks are $\hat{H}(v_1), \ldots, \hat{H}(v_k)$.

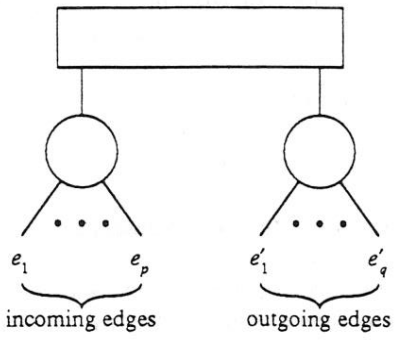

FIG. 10. The PQ-tree $T(v)$ representing the set of arrangements of a node $v \in G^{(0)}$.

To obtain an algorithm that works in $O(\log^3 n)$ time using $O(n)$ processors (where $n =$ number of nodes in $G^{(0)}$), it suffices to prove the following lemma.

LEMMA 3.13. *For each stage $i+1$ $(i=0, 1, ..., \mathscr{I}-1)$, for each node $v \in G^{(i+1)}$, a valid PQ-tree $T(v)$ representing the set of arrangements of v, can be computed from valid PQ-trees $\{T(v_j)\colon v_j \in H^{(i)}(v)\}$ in $O(\log^2 n(v))$ time using $n(v)$ processors, where $n(v) = \sum_{0 \leqslant j \leqslant k} |\text{link}(v_j)|$.*

When $T(v)$ is computed in parallel for each $v \in G^{(i+1)}$, the resulting processor bound is $\sum_{v \in G^{(i+1)}} n(v)$, which is easily seen to be $\leqslant 2$ (number of edges in $G^{(0)}$). Since we assumed that the number of edges is $\leqslant 3n$, the overall processor bound is $O(n)$.

For the time bound, certainly $n(v) \leqslant n$ for every node $v \in G^{(i+1)}$, hence by Lemma 3.13, each stage can be computed in $O(\log^2 n)$ time. There are $O(\log n)$ stages, for a total of $O(\log^3 n)$ time.

We first prove Lemma 3.13, and then show how the algorithm may be improved to work in $O(\log^2 n)$ time using n processors, proving Theorem 3.1.

Suppose that $H^{(i)}(v) = \{u_0, u_1, ..., u_k\}$, where $u_0 = p(u_j)$ for $j = 1, ..., k$. We first consider embeddings of $\widehat{H^{(i)}(v)}$. Assume that stage i belongs to the s-rooted part of a phase so that the parent u_0 is lower numbered than its children. (See Fig. 11.) Let $A = \text{in}(u_0)$. For $j = 1, ..., k$, let $E_j \subset \text{in}(u_j)$ be the non-empty set of edges between u_j and u_0, let $F_j = \text{out}(u_j)$, and let $B_j = \text{in}(u_j) - E_j$. Let $D = \text{out}(u_0) - \bigcup_{1 \leqslant j \leqslant k} E_j$. In $\widehat{H^{(i)}(v)}$, the outside endpoint of each edge in D and in each F_j is t, and the outside endpoint of each edge in A and in each B_j is s. Note that $\text{in}(v) = A \cup (\bigcup_{1 \leqslant j \leqslant k} B_j)$ and $\text{out}(v) = D \cup (\bigcup_{1 \leqslant j \leqslant k} F_j)$.

We first make an assumption.

ASSUMPTION 3.1. *If B_j is non-empty, there is a path between u_0 and t avoiding s and u_j.*

CLAIM. *Assumption* 3.1 *can be made without loss of generality.*

Proof. Suppose B_j is not empty. If $k > 1$ then u_0 has a neighbor $u_p \neq u_j$. Moreover, u_p is adjacent to t, satisfying Assumption 2. In the case $k = 1$, we can exchange the roles of u_0 and u_1 and exchange the roles of s and t for the proof of Lemma 3.13. Following this exchange, the new u_0 is adjacent to the new t. ∎

A planar embedding of $\widehat{H^{(i)}(v)}$ is shown in Fig. 11. We next establish that *every* embedding of $\widehat{H^{(i)}(v)}$ resembles that depicted in Fig. 11. Recall from Definition 2.6 that a cycle σ of elements of S *satisfies* a proper subset $A \subset S$ if the elements of A form a consecutive subsequence τ of σ, and in this case $\sigma \mid A$ denotes the subsequence τ.

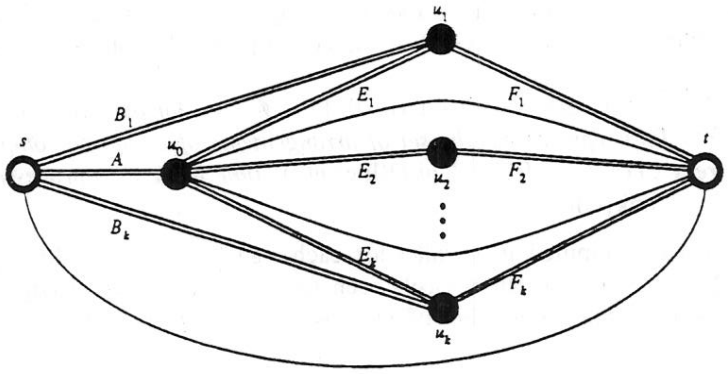

FIG.11. An embedding of $\widehat{H^{(i)}(v)}$.

LEMMA 3.14. *Let σ_j be a cycle of* $\text{link}(u_j)$ *for $j=0, ..., k$, There is a planar embedding π of $\widehat{H^{(i)}(v)}$ associating the cycle σ_j with u_j for each $j=0, ..., k$ iff the following conditions hold*:

(i) *For $j=1, ..., k$, σ_j satisfies B_j, E_j, and F_j.*

(ii) *σ_0 satisfies $E_1, ..., E_k$, $D \cup E_1 \cup \cdots \cup E_k$, and A.*

(iii) *For $j=1, ..., k$, $\sigma_0 \mid E_j = (\sigma_j \mid E_j)^R$.*

(iv) *The cycle $\sigma = \sigma_0$ join $\cdots$ join σ_k of* $\text{link}(v)$ *satisfies* $\text{in}(v)$ *and* $\text{out}(v)$.

Moreover, if π is such a planar embedding, then the cycle σ is the embedding rotation of $H^{(i)}(v)$ in $\widehat{H^{(i)}(v)}$ determined by the embedding π.

Proof. ($\Rightarrow$) Assume there is a planar embedding π of $\widehat{H^{(i)}(v)}$ associating the cycle σ_j with u_j ($\forall j$). Applying the Planar Contraction Lemma with $x=u_j$ ($j\neq 0$) and $y=t$ shows that σ_j satisfies F_j. Applying it with $x=s$ and $y=u_0$ shows σ_0 satisfies $\text{in}(u_0)=A$. Applying it with $x=u_0$ and $y=u_j$ ($j\neq 0$) shows σ_0 and σ_j satisfy E_j and $\sigma_0 \mid E_j = (\sigma_j \mid E_j)^R$. Applying it to the contracted graph $\widehat{H^{(i)}(v)}/\{t, u_1, ..., u_k\}$ with $x=u_0$ and $y=t$ shows that σ_0 satisfies $\text{out}(u_0)=D\cup E_1\cup\cdots\cup E_k$. Finally, suppose B_j is not empty. By Assumption 3.1, there is a path between u_0 and t avoiding s and u_j. Hence we can apply the Planar Contraction Lemma with $x=s$ and $y=u_j$ to show that σ_j satisfies B_j.

Using k applications of the last statement of the Planar Contraction Lemma, one can show that σ is the embedding rotation of $H^{(i)}(v)$ in $\widehat{H^{(i)}(v)}$ determined by the embedding π. This means that when we contract the subgraph $\{u_0, ..., u_k\}$ of the embedded graph $(\widehat{H^{(i)}(v)})_\pi$, the cycle associated with the resulting node v is σ. Applying the Planar Contraction Lemma to this contracted graph with $x=s$ and

$y = v$ shows that σ satisfies in(v); applying it with $x = v$ and $y = t$ shows that σ satisfies out(v).

($\Leftarrow$) Suppose conditions (i), (ii), (iii), and (iv) are satisfied. Define the embedding π as follows: π associates σ_j with u_j ($\forall j$), and associates $(e\ (\sigma \mid \text{in}(v))^R)$ with s and $(e\ (\sigma \mid \text{out}(v))^R)$ with t, where e is the edge between s and t. We use induction on k to show that π is a planar embedding.

First suppose $k = 1$. The first two conditions of the Planar Contraction Lemma with $x = u_0$ and $y = u_1$ follow from (i), (ii), and (iii). We must verify the third condition. Starting from the embedded graph $\widehat{(H^{(i)}(v))}_\pi$, contract the subgraph $\{u_0, u_1\}$ to form v. By the last statement in the Planar Contraction Lemma, the cycle associated with v is σ. It is then easy to verify that the resulting embedding, consisting of σ, $(e\ (\sigma \mid \text{in}(v))^R)$, and $(e\ (\sigma \mid \text{out}(v))^R)$ is a planar embedding. Thus the third conditions of the Planar Contraction Lemma is verified, so the embedding π is planar.

Next, suppose $k > 1$. Write $\sigma_0 = (\alpha\ \delta_1\ \varepsilon_1\ \delta_2\ \varepsilon_2 \cdots \delta_k\ \varepsilon_k\ \delta_{k+1})$, where α is an ordering of A, ε_j is an ordering of E_j (renumbering $u_1, \ldots, u_k$ if necessary), and δ_j is an ordering of the subset $D_j \subset D$ (writing D as the disjoint union of $D_1, \ldots, D_{k+1}$). Write $\sigma_1 = (\varepsilon_1^R\ \beta_1\ \phi_1)$, where β_1 is an ordering of B_1 and ϕ_1 is an ordering of F_1. Then $\sigma = \sigma_0 \text{ join} \cdots \text{join } \sigma_k$ has the form $(\alpha\ \delta_1\ \beta_1\ \phi_1\ \delta_2 \cdots)$. Now A and B_1 are subsets of in(v), and σ satisfies in(v), so either α and β_1 must be adjacent, and hence D_1 is empty, or else B_j is empty.

Let $\sigma_0' = \sigma_0 \text{ join } \sigma_1 = (\alpha\ \delta_1\ \beta_1\ \phi_1\ \delta_2\ \varepsilon_2 \cdots)$. Let $H_{\pi'}$ be the embedded graph obtained from $\widehat{(H^{(i)}(v))}_\pi$ by contracting the subgraph $\{u_0, u_1\}$ to form u_0'. It follows from the last statement of the Planar Contraction Lemma that σ_0' is the cycle π' associates with u_0'. The incoming edges of u_0' are $A \cup B$, and the outgoing edges are $D \cup F_1 \cup E_2 \cup \cdots \cup E_k$. From the fact that either D_1 or B_1 is empty, it follows that σ_0' satisfies the incoming edges of u_0' and the outgoing edges of u_0'. We may therefore use the inductive hypothesis to show that π' is a planar embedding. We have satisfied the third condition of the Planar Contraction Lemma applied with $x = u_0$ and $y = u_1$; the first and second follow from (i), (ii), and (iii). Hence π is a planar embedding. ∎

Every planar embedding of $\hat{H}(v)$ induces a planar embedding of $\widehat{H^{(i)}(v)}$, namely, that obtained by contracting the disjoint subgraphs $H(u_j)$ to u_j for $j = 0, \ldots, k$. A given planar embedding π of $\widehat{H^{(i)}(v)}$ is induced by some planar embedding of $\hat{H}(v)$ iff for $j = 0, \ldots, k$, the cycle σ_j of π associated with the node u_j is an arrangement of u_j. (This follows from Lemma 3.3 of Subsection 3.1.) We conclude that the arrangements of v are those cycles arising as $\sigma_0 \text{ join} \cdots \text{join } \sigma_k$, where σ_j is an arrangement of u_j for $j = 0, \ldots, k$ and conditions (i), (ii), (iii), and (iv) of Lemma 3.14 are satisfied.

We make a second assumption:

ASSUMPTION 3.2. *The sets $B_2, ..., B_{k-1}$ are empty.*

CLAIM. *Assumption can be made without loss of generality.*

Proof. By the *st*-numbering, each F_j is non-empty. Suppose more than two B_j's are non-empty, say B_1, B_2, B_3. Then in $\widehat{H^{(i)}}(v)$, the nodes s, t, and u_0 are all adjacent to the nodes u_1, u_2, u_3. Thus the Kuratowski subgraph $K_{3,3}$ is a subgraph of $\widehat{H^{(i)}}(v)$, proving that $\widehat{H^{(i)}}(v)$ is not planar and hence that $G^{(0)}$ is not planar, by Kuratowski's theorem (see [2]). In this case, we terminate the algorithm. Otherwise, by renumbering the nodes $u_1, ..., u_k$, we can ensure Assumption 3.2. ∎

We break up the process of computing a valid *PQ*-tree $T(v)$ into the following three steps:

P1 Process each $T(u_j)$ to get a *PQ*-tree $T'(u_j)$ such that $\mathrm{CYC}(T'(u_j)) = \{\sigma_j \in \mathrm{CYC}(T(u_j))$: σ_j satisfies (i) and (ii) of Lemma 3.14$\}$.
To make the next step possible, ensure that for $j = 1, ..., k$, the set $\mathrm{link}(u_j) - E_j$ is contiguous in $T'(u_j)$.

P2 Let $T'(v)$ be the join of $T'(u_0)$ with $T'(u_1), ..., T'(u_k)$.

P3 Process $T'(v)$ to get a *PQ*-tree $T(v)$ such that

$$\mathrm{CYC}(T(v)) = \{\sigma \in \mathrm{CYC}(T'(v))\text{: } \sigma \text{ satisfies in}(v) \text{ and out}(v)\}$$

and in fact $\mathrm{in}(v)$ and $\mathrm{out}(v)$ are contiguous in $T(v)$.

A procedure for carrying out step P2 is described in Subsection 2.5. It remains to fill out further details in steps P1 and P3.

First suppose $j \in \{1, ..., k\}$ and $B_j = \varnothing$. In this case, $E_j = \mathrm{in}(u_j)$ and $F_j = \mathrm{out}(u_j)$. We assumed that $T(u_j)$ is valid, hence every cycle in $\mathrm{CYC}(T(u_j))$ satisfies E_j and F_j, and, moreover, E_j and $\mathrm{link}(u_j) - E_j = F_j$ are contiguous in $T(u_j)$. Hence for such a j, we merely let $T'(u_j) = T(u_j)$ for step P1.

Next, suppose j is 1 or k, and B_j is non-empty. Note that because $T(u_j)$ is valid, the two sets $F_j = \mathrm{out}(u_j)$ and $E_j \cup B_j = \mathrm{in}(u_j)$ are contiguous in $T(u_j)$. Calling upon Lemma 2.3, we reduce $T(u_j)$ with respect to E_j and B_j using MREDUCE. Next we carry out ROTATE(E_j, B_j, F_j) to ensure that $B_j \cup F_j = \mathrm{link}(u_j) - E_j$ is contiguous. (See Lemma 2.2). We let $T'(u_j)$ be the resulting *PQ*-tree.

Now we consider the *PQ*-tree $T(u_0)$. Note that because $T(u_0)$ is valid, $\mathrm{in}(u_0) = A$ and $\mathrm{out}(u_0) = D \cup E_1 \cup \cdots \cup E_k$ are contiguous in $T(u_0)$. Calling upon Lemma 2.3, we use MREDUCE to reduce $T(u_0)$ with respect to $E_1, ..., E_k \subseteq \mathrm{out}(u_0)$, letting $T'(u_0)$ be the reduced tree. Because of the validity of $T(u_0)$, every ordering in $\mathrm{CYC}(T'(u_0))$ already satisfies A.

Having processed each *PQ*-tree $T(u_j)$ to obtain a *PQ*-tree $T'(u_j)$, we now compute the join of $T'(u_0)$ with $T'(u_1), ..., T'(u_k)$, obtaining a *PQ*-tree $T'(v)$.

Step P3 remains. By (1) of Lemma 2.28, $B_1 \cup B_k \cup D \cup F_1 \cup \cdots \cup F_k$ is contiguous in $T'(v)$. If B_1 and B_k are empty, we are done, for then it follows that

$\text{in}(v) = A$ and $\text{out}(v) = D \cup F_1 \cup \cdots \cup F_k$ are contiguous in $T'(v)$. Therefore, assume without loss of generality that B_1 is non-empty. Calling upon Lemma 2.3, we use MREDUCE to reduce $T'(v)$ with respect to $\text{out}(v)$. By Lemma 2.1, $\text{out}(v)$ is now contiguous in $T'(v)$. Also, since $B_k \cup F_k$ was contiguous in $T'(u_k)$, the same set is contiguous in $T'(v)$ by (2) of Lemma 2.28. Since $\text{out}(v)$ contains F_k, it follows by Corollary 2.1 that $B_k \cup \text{out}(v)$ is contiguous in $T'(v)$. Carry out ROTATE(A, B_1, $B_k \cup \text{out}(v)$) to make $A \cup B_1$ contiguous. Then if B_k is nonempty, carry out ROTATE($A \cup B_1$, B_k, $\text{out}(v)$) to make $A \cup B_1 \cup B_k$ contiguous. Let the resulting PQ-tree be $T(v)$.

Steps P1 and P3 can be computed in $O(\log n(v))$ time. Step P2, however, takes time $O(\log^2 n(v))$. This is sufficient to prove Lemma 3.13.

We have shown how the planarity algorithm may be carried out in $O(\log^3 n)$ time using $O(n)$ processors. To improve the time bound to $O(\log^2 n)$ time, we reduce the time for step P2 from $O(\log^2 n(v))$ time to $O(\log n(v))$, by computing a "provisional" join (described in Subsection 2.5) that is correct unless the correct result is the null PQ-tree. This permits us to quickly proceed to step P3 and then to the next stage. We need not wait for the join to be verified, because if a join fails the verification, it means that v has no arrangements, so $\hat{H}(v)$ is not planar and hence $G^{(0)}$ is not planar. We delay the verification of all the joins until after the last stage is completed. The processor bound for verification of the join of step P2 is proportional to the number $l = |E_1 \cup \cdots \cup E_k|$ of common elements. These are elements of $\text{link}(u_0)$ and $\text{link}(u_j)$ (for $j = 1, ..., k$) not appearing in $\text{link}(v)$. Since each edge of $G^{(0)}$ occurs at most once as a common element in a join, the total number of processors required to verify all joins simultaneously is proportional to the number of edges. The time is $O(\log^2(\text{number of common elements})) = O(\log^2 n)$. Thus the time for all verifications merely adds $O(\log^2 n)$ time to the total time for the planarity-testing algorithm.

Finally, we sketch the method for obtaining a combinatorial embedding of each $\hat{H}(v)$, assuming the planarity-testing algorithm successfully terminated. That algorithm consisted of a sequence of $\mathscr{I} = O(\log n)$ stages; in each stage subgraphs were contracted to single nodes. In carrying out this process, we defined a *contraction forest* with trees rooted at the nodes of $G^{(\mathscr{I})}$, where the children of a node $u \in G^{(i+1)}$ are the nodes $H^{(i)}(v)$. The trees are all of height $\leqslant \mathscr{I} + 1 = O(\log n)$. For each node $r \in G^{(\mathscr{I}+1)}$, we have a cycle $\text{cycfr}(T(r))$ of the $H(r)$-linking edges that is consistent with some embedding of $G^{(\mathscr{I})}$. We first show how to choose cycles σ_s and σ_t of the edges incident to s and t respectively, such that σ_s, σ_t, and the σ_r's are all consistent with some embedding of $G^{(0)}$. Then we show how to find such an embedding by processing each tree of the contraction forest from the root down.

Say the nodes of $G^{(\mathscr{I}+1)}$ are $w_1, ..., w_k$, and $\sigma_1, ..., \sigma_k$ are the corresponding cycles. The incoming edges of w_i form a consecutive subsequence λ_i of σ_i, and the outgoing edges form a consecutive subsequence λ_i' of σ_i. Let $\sigma_s = \text{cyc}(\lambda_k^R \lambda_{k-1}^R \cdots \lambda_1^R e)$ and $\sigma_t = \text{cyc}(\lambda_1'^R \lambda_2'^R \cdots \lambda_k'^R \varepsilon)$, where e is the edge between s and t. It follows from the Planar Contraction Lemma that all the cycles are consistent with an embedding of $G^{(0)}$.

Next we find cycles for all the nodes of $G^{(0)}$ other than s and t, thereby defining an embedding of $G^{(0)}$. We do this in $\mathcal{I}$ stages, by processing each tree of the contraction forest in parallel, starting at its root r and ending at its leaves, which are nodes of $G^{(0)}$. Say we are at height $i+1$ in the contraction tree rooted at a node $r \in G^{(\mathcal{I}+1)}$. Assume inductively that there is some planar embedding of $G^{(0)}$ such that σ_v is the corresponding embedding rotation of $H(v)$, and we are given σ_v for each node v at height $i+1$ (i.e., each $v \in G^{(i+1)}$) in the tree. We operate on each such v in parallel. Suppose $H^{(i)}(v) = \{u_0, ..., u_k\}$ as in Fig. 11 and in Lemma 3.14 and the preceeding text. To carry out the induction step, we must find $\sigma_0, ..., \sigma_k$ such that $\sigma_v = \sigma_0 \text{ join } \sigma_1 \text{ join } \cdots \text{ join } \sigma_k$, such that (i), (ii), (iii), and (iv) of Lemma 3.14 are satisfied, and such that σ_i is an arrangement of u_i, for $i = 0, 1, ..., k$. In a sense, we wish to "invert" the join operation. A procedure for doing this is sketched in Lemma 2.27 of Subsection 2.5. Using this procedure, we can descend a single level of the contraction forest in $O(\log n)$ time using n processors. The forest has $O(\log n)$ levels, so after $O(\log^2 n)$ time, we end up with cycles for each node $v \in G^{(0)}$ other than s and t.

ACKNOWLEDGMENTS

We would like to thank colleagues at MIT's Theory of Computation Group for their generous assistance, espcially David Shmoys, who advised the Masters' thesis based on the research reported here, and also including others who read early versions of the paper: David Barrington, Bard Bloom, Lenny Heath, and Tom Leighton. Thanks also to Ray Hirschfeld and Mark Reinhold for their help in document preparation.

REFERENCES

1. M. AJTAI, J. KOMLÓS, AND E. SZEMERÉDI, An $O(n \log n)$ sorting network, *in* "Proceedings, 15th Annual Symposium on the Theory of Computing," 1983, pp. 1–9.
2. J. BONDY AND U. MURTY, "Graph Theory with Applications," North-Holland, New York, 1976.
3. K. BOOTH AND G. LUEKER, Testing for the consecutive ones property, interval graphs, and graph planarity using *PQ*-tree algorithms, *J. Comput. System Sci.* **13**, No. 3 (1976), 335–379.
4. R. COLE, Parallel merge sort, *in* "27th Annual IEEE Symposium on Foundations of Computer Science," 1986, pp. 511–516.
5. R. EDMONDS, A combinatorial representation for polyhedral surfaces, *Amer. Math. Soc. Notices* **7** (1960), 646.
6. S. EVEN AND R. TARJAN, Computing and *st*-numbering, *Theoret. Comput. Sci.* **2** (1976), 339–433.
7. S. FORTUNE AND J. WYLLIE, Parallelism in random, access machines, *in* "Proceedings, 10th Annual IEEE Symposium on the Foundations of Computer Science," 1978, pp. 98–108.
8. J. HOPCROFT AND R. TARJAN, An efficient algorithm for graph planarity, *J. Assoc. Comput. Mach.* **21** (1974), 549–568.
9. J. JA' JA' AND J. SIMON, Parallel algorithms in graph theory: Planarity testing, *SIAM J. Comput.* **11**, No. 2 (1982), pp. 313–328.
10. P. KLEIN, "An Efficient Parallel Algorithm for Planarity," Master's thesis, MIT, 1986.

11. P. KLEIN AND J. REIF, An efficient parallel algorithm for planarity, *in* "Proceedings, 27th Annual IEEE Symposium on Foundations of Computer Science," 1986, pp. 465–477.
12. F. T. LEIGHTON, Tight bounds on the complexity of parallel sorting, *in* "Proceedings, 16th Annual Symposium on Theory of Computing," 1984, pp. 71–80.
13. A. LEMPEL, S. EVEN, AND I. CEDERBAUM, An algorithm for planarity testing of graphs, *in* "Theory of Graphs: International Symposium: Rome, July, 1966" (P. Rosenstiehl, Ed.), pp. 215–232, Gordon & Breach, New York, 1967.
14. L. LOVASZ, Computing ears and branchings in parallel, *in* "26th Annual IEEE Symposium on Foundations of Computer Science, 1985," pp. 464–467.
14a. G. LUEKER AND K. BOOTH, A linear time algorithm for deciding interval graph isomorphism, *J. Assoc. Comput. Mach.* **26** (1979), 183–195.
15. Y. MAON, B. SCHIEBER, AND U. VISHKIN, "Parallel Ear Decomposition Search (EDS) and *st*-Numbering in Graphs," Tel Aviv University Technical Report 46/86; "Aegean Workshop on Computing VLSI Algorithms and Architectures, Loutraki, Greece, July, 1986," Springer-Verlag, Berlin/New York.
16. S. MCLANE, A combinatorial condition for planar graphs, *Fund. Math.* **28** (1937), 22–32.
17. G. MILLER AND J. REIF, Parallel tree contraction and its application, *in* "Proceedings, 26th IEEE Annual Symposium on Foundations of Computer Science," 1985, pp. 478–489.
18. J. REIF, An optimal parallel algorithm for integer sorting, *in* "Proceedings, 26th IEEE Symposium on the Foundations of Computer Science," pp. 496–503.
19. R. TARJAN AND V. VISHKIN, Finding biconnected components and computing tree functions in logarithmic parallel time, *in* "Proceedings, 25th IEEE Annual Symposium on Foundations of Computer Science, 1984," pp. 12–22.
20. U. VISHKIN, Implementation of simultaneous memory access in models that forbid it, *J. Algorithms* **4** (1983), 45–50.
21. A. T. WHITE AND L. W. BEINEKE, Topological graph theory, *in* "Selected Topics in Graph Theory" (L. W. Beineke and R. J. Wilson, Eds.), pp. 15–49, Academic Press, New York/London, 1978.
22. H. WHITNEY, Non-separable and planar graphs, *Trans. Amer. Math. Soc.* **34** (1930), 339–362.

Printed by Catherine Press, Ltd., Tempelhof 41, B-8000 Brugge, Belgium