PLANARITY TESTING IN PARALLEL*

Vijaya Ramachandran† and John Reif‡

Department of Computer Sciences
The University of Texas at Austin
Austin, Texas 78712-1188

TR-90-15 June 1990

* A preliminary version of this paper appeared in *Proceedings of the 30th IEEE Annual Symposium on Foundations of Computer Science*, 1989 [RR89].

† Supported by NSF Grant CCR 89-10707.

‡ Duke University. Supported by Air Force Contract AFOSR-87-0386, DARPA/ISTO Contract N00014-88-K-0458, DARPA/ARO Contract DAAL03-88-K-0195 and NASA/CESDIS Contract 550-63 NAS 5-30428 URSA.

Planarity Testing in Parallel*

Vijaya Ramachandran†
University of Texas at Austin

John Reif‡
Duke University

ABSTRACT

We present a parallel algorithm based on open ear decomposition to construct an embedding of a graph onto the plane or report that the graph is non-planar. Our parallel algorithm runs on a CRCW PRAM in logarithmic time with a number of processors bounded by that needed for testing graph connectivity and for performing bucket sort.

A sequential implementation of our algorithm gives a new linear-time sequential algorithm for planarity testing.

1. Introduction

1.1. The Planarity Problem

Informally, a graph is planar if it can be embedded onto the plane so that the edges do not cross (see section 2.1 for formal definition). Euler first defined this fundamental concept in 1736 and stated the Euler formula for planar embeddings.

Planar graphs appear naturally in many applications, -- for example, in the solution of 2 dimensional PDEs and in VLSI layout. Many difficult graph problems can be solved in polynomial time in the case of planar graphs.

The planarity problem is the following: given a graph G, test if G has a planar embedding and, if so, construct an embedding of G onto the plane. A planarity algorithm is one that solves the planarity problem. There has been a considerable amount of research on this problem, beginning with the characterization theorems for planarity of Whitney [Wh30], Kuratowski [Ku30] and Edmonds [Ed60] which led to exponential time planarity algorithms, followed by the first

*A preliminary version of this paper appeared in *Proceedings of the 30th IEEE Annual Symposium on Foundations of Computer Science*, 1989 [RR89].

†Supported by NSF Grant CCR 89-10707.

‡Supported by Air Force Contract AFOSR-87-0386, DARPA/ISTO Contract N00014-88-K-0458, DARPA/ARO Contract DAALO3-88-K-0195 and NASA/CESDIS Contract 550-63 NAS 5-30428 URSA.

polynomial time planarity algorithm of Tutte [Tu63] and culminating with the linear time sequential planarity algorithm of Hopcroft & Tarjan [HT74], which used depth first search and built on techniques developed in a triconnectivity algorithm [HT73]. Another planarity algorithm developed by Lempel, Even & Cederbaum [LEC67] was made to run in linear time by results in Booth & Lueker [BL76] for manipulating PQ trees and by the algorithm of Even & Tarjan [ET76] for computing an st-numbering.

1.2. Previous Parallel Algorithms for Planarity

Considerable previous work has been devoted to developing parallel planarity algorithms with respect to the Parallel Random Access Machine (PRAM). Ja'Ja' & Simon [JS82] first showed that testing planarity can be done in NC. Miller & Reif [MR85] later gave a parallel planarity algorithm with similar resource bounds that also gave a planar embedding of an arbitrary planar graph. Reif [Re84] gave a randomized logarithmic time NC algorithm for graphs of valence 3. Klein & Reif [KR88] gave the best previously known polylog time planarity algorithm in terms of processor efficiency, which required time $O(\log^2 n)$ using a linear number of processors; this algorithm is a parallelization of the sequential algorithm in [LEC67].

1.3. Our Parallel Planarity Algorithm

Our parallel planarity algorithm is a deterministic algorithm that runs in logarithmic time on a Concurrent Read Concurrent Write (CRCW) PRAM while performing almost linear work. (See Karp & Ramachandran [KR90] for a discussion of parallel algorithms on various PRAM models.) More precisely, let $C(n,m)$ be the bound on the work done by a parallel algorithm that finds the connected components of an n-node, m-edge graph in logarithmic time on a CRCW PRAM when the graph is represented by adjacency lists; currently the best bound is $C(n,m) = O((n+m)\cdot\alpha(n,m))$ [CV86], where α is the inverse Ackermann's function, which grows very slowly with n and m. Let $B(n)$ be the bound on the work to perform bucket sort on n $O(\log n)$ bit numbers in logarithmic time on a CRCW PRAM; currently $B(n) = O(n \cdot \log\log n)$. Finally, let $A(n,m) = \max(C(n,m), B(n))$. Our planarity algorithm runs in logarithmic time on a CRCW PRAM while performing $A(n,n)$ work. We will refer to such a performance bound as 'logarithmic time with A-optimal performance.'

Our algorithm uses a variety of techniques found in previous parallel graph algorithms. We search the graph using a parallel algorithm for open ear decomposition [MR86, MSV86, Ra90]. Open ear decomposition has proved to be a very useful technique in the efficient parallel solution of several problems on undirected graphs (see, e.g., [FRT89, KR87, KR90, MSV86, Ra90]). To further order our parallel searches, we make use of the parallel algorithm of [MSV86] for st-numbering. We use the local replacement graph and the bridges of ears computed in the parallel triconnectivity algorithm of Fussell, Ramachandran & Thurimella [FRT89]; for this material we will follow the treatment in the chapter [Ra90]. We extend the interlacing parity algorithm of Ramachandran & Vishkin [RV88] in order to obtain the planar embedding of the input graph; again, for this material, we will follow the treatment in the chapter [Ra90]. We also make use of optimal logarithmic time algorithms for computing tree functions [TV84, CV86, KD88] and for computing least common ancestors (lca) of pairs of vertices in a rooted tree [SV88].

Our algorithm differs from all previous planarity algorithms in its use of a general open ear decomposition for graph searching. However, it is somewhat similar in spirit to the algorithm of Hopcroft & Tarjan [HT74] in that it embeds paths rather than vertices; as in the case of the algorithm in [HT74], our algorithm makes extensive use of techniques developed in a triconnectivity algorithm, i.e., the parallel algorithm in Fussell, Ramachandran & Thurimella [FRT89] to find the triconnected components of a graph. At the same time, since our algorithm uses st-numbering to direct the embedding, it has some similarity to the Lempel, Even & Cederbaum algorithm [LEC67]. Our algorithm makes no use of parallel PQ tree techniques to represent planar embeddings, but instead makes a reduction to finding a 2 coloring of an undirected graph, a special case of which is used in Ramachandran & Vishkin [RV88] to find a planar embedding for a graph with a known Hamiltonian cycle. Similar, though less efficient, approaches have been used by Ja'Ja' & Simon [JS82] who gave an NC reduction of planarity testing to 2-SAT (satisfiability with 2 literals per clause) and by Reif [Re84] who gave a randomized NC reduction of trivalent planarity testing to 2-SAT with the two literals in exclusive-or form; this latter problem is equivalent to 2 coloring an associated undirected graph.

All of the steps in our algorithm can be performed in linear sequential time. Hence it gives a new linear time sequential algorithm for planarity.

1.4. Algorithmic Notation

The algorithmic notation in this paper is from Tarjan [Ta83]. We enclose comments between a pair of double curly brackets ('{{' and '}}'). We incorporate parallelism by use of the following statement that augments the **for** statement.

for iterator **in parallel** → statement list **rof**

The effect of this statement is to perform the **for** loop in parallel for each value of the iterator.

1.5. Organization of the Paper

The rest of the paper is organized as follows. Section 2 gives definitions and relevant earlier results. Section 3 gives a high-level description of our algorithm. Section 4 describes bunches, their hooks and the bunch graphs. Section 5 defines the constraint graph. Section 6 relates 2-colorings of the constraint graph to planar embeddings of the input graph, and gives a placement of each bunch on one side of its fundamental cycle. Section 7 refines this placement to obtain a combinatorial embedding of the graph. Finally, Section 8 gives the full algorithm.

2. Preliminaries

In this section we provide major definitions and previous results from the literature that we will need in later sections. Several other graph-theoretic definitions are given in the Appendix.

2.1. Planar Embeddings

2.1.1. Planar Topological Embeddings

We define here a *planar topological embedding* of an undirected graph $G=(V,E)$ (see, e.g., White [Wh73]). In such a topological embedding, each edge is associated with a simple segment on the plane, where the vertices of the edge are at the two distinct endpoints of the segment, and where no two such edges intersect except at endpoints in the case of common vertices. The *faces* of the embedding are the connected boundaries of the maximum connected regions obtained by deleting the embedding of G from the plane. Euler's formula gives $n-m+f=2c$, where m,n,f and c are the numbers of edges, vertices, faces, and connected components, respectively.

2.1.2. Planar Combinatorial Embeddings

The topological definition of planar embedding given above presents difficulties for computer algorithms and their proofs. Given an undirected graph $G=(V,E)$ with $|V|=n$, we will represent an embedding of graph G by a combinatorial representation that is attributed to Edmonds [Ed60] (see also White [Wh73]); this representation has size $O(n)$. Let $D(G)$ be the directed graph derived from G by substituting in place of each undirected edge (u,v), a pair of directed edges (u,v) and (v,u). A *combinatorial graph embedding* $I(G)$ of the graph G is an assignment of a cyclic ordering to the set of directed edges departing each vertex in $D(G)$. The faces of this combinatorial embedding are the orbits of a certain permutation of the directed edges; this permutation orders (w,v) before (v,u) if and only if the combinatorial embedding orders (v,u) immediately before (v,w) in the cyclic order around vertex v. The combinatorial embedding is planar if it satisfies the Euler's formula $n-m+f=2c$, calculated from the numbers of (undirected) edges m, vertices n, faces f, and connected components c. Edmonds [Ed60] showed that combinatorial embeddings onto the plane can be put in 1-1 correspondence to topological embeddings onto the plane.

Given a directed simple cycle $C=<v_0,v_1,\cdots,v_k=v_0>$ in G, and an edge (v_i,x) where v_i is in C but x is not, we will define (v_i,x) to be embedded *inside* C (and otherwise *outside* C) if in the cyclic order defined by $I(G)$ on directed edges departing vertex v_i, directed edge (v_i,x) appears after directed edge (v_i,v_{i+1}) and before directed edge (v_i,v_{i-1}). Hereafter, we will use the term planar embedding to denote a combinatorial embedding onto the plane.

We extend the above definition to the embedding of an edge relative to a directed path. Given a directed path $P=<v_0,v_1,\cdots,v_k>$ in G, and an edge (v_i,x) where x is not in P and v_i is an internal vertex on P, we will define (v_i,x) to be embedded *inside* P (and otherwise *outside* P) if in the cyclic order defined by $I(G)$ on directed edges departing vertex v_i, directed edge (v_i,x) appears after directed edge (v_i,v_{i+1}) and before directed edge (v_i,v_{i-1}).

2.2. Bridges of a Subgraph

Let $G=(V,E)$ be an undirected graph, and let Q be a subgraph of G. We define the *bridges of Q in G* as follows: Let V' be the vertices in $G-Q$, and consider the partition of V' into classes such that two vertices are in the same class if and only if there is a path connecting them which does not use any vertex of Q. Each such class K defines a *nontrivial bridge* $B=(V_B,E_B)$ of Q, where B is the subgraph of G with $V_B=K \cup$ {vertices of Q that are connected by an edge to a vertex in K}, and E_B containing the edges of G incident on a vertex in K. The vertices of Q which are connected by an edge to a vertex in K are called the *attachments* of B on Q; the connecting edges are called the *attachment edges.* An edge (u,v) in $G-Q$, with both u and v in Q, is a *trivial bridge* of Q, with attachments u and v. The nontrivial and trivial bridges of Q together form the *bridges* of Q.

Let $G=(V,E)$ be a graph and let $V' \subseteq V$ with the subgraph of G induced on V' being connected. The operation of *collapsing the vertices in V'* consists of replacing all vertices in V' by a single new vertex v, deleting all edges in G whose two endpoints are in V' and replacing each edge (x,y) with x in V' and y in $V-V'$ by an edge (v,y). In general the resulting graph is a multigraph even if the original graph G is not a multigraph.

Let $G=(V,E)$ be an undirected graph, and let Q be a subgraph of G. The *bridge graph of Q*, $S=(V_S,E_S)$ is obtained from G by collapsing the nonattachment vertices in each nontrivial bridge of Q and by replacing each trivial bridge $b=(u,v)$ of Q by the two edges (x_b,u) and (x_b,v) where x_b is a new vertex introduced to represent the trivial bridge b. Note that in general the bridge graph is a multigraph.

2.3. Interlacing Bridges

Let $P=<0,1,2, \cdots ,k>$ be a simple path in a graph G. A pair of bridges *interlace* on P if one of the following two holds:

1. There exist four distinct vertices a,b,c,d with $a<b<c<d$ such that a and c are attachments of one of the bridges on P and b and d are attachments of the other bridge on P; or
2. There are three distinct vertices of P that are attachments of both bridges.

If bridges S and T interlace on P, then they cannot be placed on the same side of P in a planar embedding. If S and T do not interlace, then they can be placed in a planar embedding on the same (opposite) side of P if and only if there exists no sequence of bridges $<S=S_0,S_1, \cdots ,S_r=T>$, with r odd (even) such that S_i interlaces with S_{i+1}, $0 \le i \le r-1$. If there is such a sequence with r even then S and T have *even interlacing parity* and if there is such a sequence with r odd, then S and T have *odd interlacing parity*. If no such sequence exists for r either odd or even, then S and T have *null interlacing parity:* in this case S and T can be placed either in the same side or in opposite sides of P in a planar embedding (provided G is planar). It is possible for S and T to have both odd and even parity, -- in this case, no planar embedding of G is possible if every star is to be placed completely on one side of P.

2.4. The Star Graph and Its Interlacing Parity Graph

Let P be a simple path in a graph $G=(V,E)$. If each bridge of P in G contains exactly one vertex not on P, then we call G the *star graph of* P and denote it by $G(P)$. We denote the bridges of $G(P)$ by *stars*, i.e., a *star* is a connected graph in which at most one vertex has degree greater than 1. The unique vertex of a star that is not contained in P is called its *center*. If $P=<0,1,\cdots,n>$ then given a star S of $G(P)$ with attachments $v_0<v_1<\cdots<v_r$ on P, we will call v_0 and v_r the *end attachments* of S and the remaining attachments the *internal attachments* of S; the vertex v_0 is the *leftmost attachment* of S, and the vertex v_r is its *rightmost attachment.*

Note that, in a connected graph G, the bridge graph of any simple path in G is a star graph. We will sometimes refer to a star graph $G(P)$ by G if the path P is clear from the context.

We now define the *interlacing parity graph* G_I of a star graph $G(P)$. Let $P=<0,1,\cdots,n>$. We replace each star S on $G(P)$ by a collection of edges as follows: Let the attachments of S on P be $a_0,a_1,\cdots,a_k$ with $a_0<a_1<\cdots<a_k$. We replace S by the edges $(a_0,a_i), i=1,\cdots,k$ and the edges $(a_i,a_k), i=1,\cdots,k-1$ (see figure 1a). We will refer to these edges as the *chords of* S.

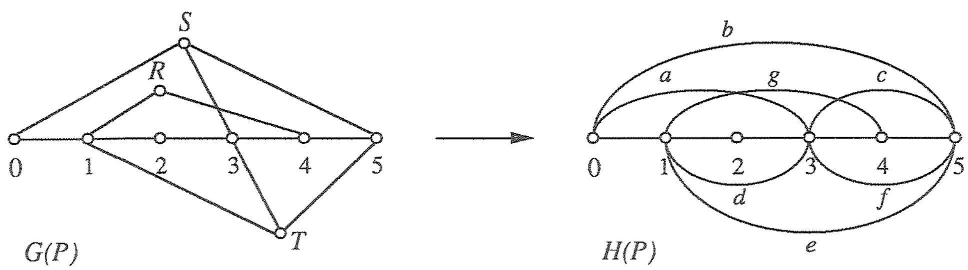


Figure 1a
Forming $H(P)$ from $G(P)$

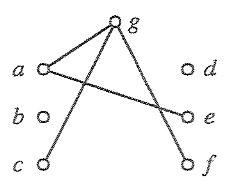


Figure 1b
Edges in E_1

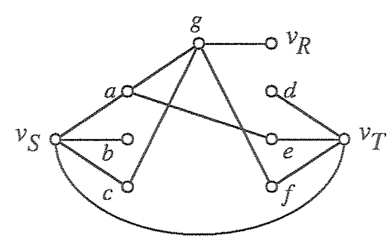


Figure 1c
Interlacing parity graph G_I of $G(P)$

Figure 1
Constructing the interlacing parity graph of a star graph.

Let $H(P)$ be the graph obtained from $G(P)$ by replacing each star in $G(P)$ by its chords. We will say that chords c and d in $H(P)$ are *related* if they are chords of the same star S in $G(P)$ and are *unrelated* otherwise. We construct $G_I=(V',E')$, the *interlacing parity graph of* $G(P)$ as follows:

$V' = V_1 \cup V_2$, where

$V_1 = \{v_e \mid e$ is a chord in $H(P)\}$ and

$V_2 = \{v_S \mid S$ is a star in $G(P)\}$; we will refer to a vertex in V_2 as a *star vertex*.

$E' = \{(v_S, u_e) \mid S$ is a star in $G(P)$ and u_e is a vertex in V' representing a chord e of $S\}$ $\cup E_1 \cup F$,

where E_1 and F is defined as follows:

DEFINITION OF E_1:

For each chord c in $H(P)$ we first define its *left chord* l_c and its *right chord* r_c. Let $c=(u,v)$, $u<v$ and let c be a chord of star S in $G(P)$.

Left chord of c *:* Let u_l be the minimum numbered vertex on P such that c interlaces with an unrelated chord d incident on u_l. If $u_l<u$ then choose an unrelated chord (u_l, v_l) with maximum v_l that interlaces with c to be the left chord l_c of c.

Right chord of c *:* Let v_r be the maximum numbered vertex on P such that c interlaces with an unrelated chord incident on v_r. If $v_r>v$ then choose an unrelated chord (u_r, v_r) with minimum u_r that interlaces with c to be the right chord r_c of c.

Then $E_1 = \{(v_c, v_{l_c}) \mid c$ is a chord of $H(P)$ and l_c is its left chord (if it exists)$\}$ $\cup$ $\{(v_c, v_{r_c}) \mid c$ is a chord of $H(P)$ and r_c is its right chord (if it exists)$\}$. (See figure 1b)

DEFINITION OF F:

For each vertex i on P let

$F_i = \{(v_S, v_T) \mid S$ is a star in $G(P)$ with an internal attachment on i and T ranges over all other stars in $G(P)$ with an internal attachment on $i\}$

Then $F = \bigcup_{i=1}^{n-1} F_i$.

Figure 1c give the interlacing parity graph G_I of the graph $G(P)$ in figure 1a.

It is shown in [Ra90, RV88] that a two-coloring of G_I exists if and only if there exists a planar embedding of $G(P)$ with each star in $G(P)$ being embedded entirely on one side on P. Further, a planar embedding of $G(P)$ can be obtained by embedding all stars corresponding to star vertices of one color inside P and all stars corresponding to star vertices of the other color outside P.

2.5. Open Ear Decomposition and st-Numbering

An *ear decomposition* $D=[P_0, P_1, \cdots, P_{r-1}]$ of an undirected graph $G=(V,E)$ is a partition of E into an ordered collection of edge disjoint simple paths $P_0, \cdots, P_{r-1}$ such that P_0 is an edge, $P_0 \cup P_1$ is a simple cycle, and each endpoint of P_i, for $i>1$, is contained in some $P_j, j<i$, and none of the internal vertices of P_i are contained in any $P_j, j<i$. The paths in D are called *ears*. A *trivial ear* is an ear containing a single edge. An ear $P_i, i>1$, is *open* if it is noncyclic and is *closed* otherwise. D is an *open ear decomposition* if all of its ears are open.

Let $D=[P_0, \cdots, P_{r-1}]$ be an ear decomposition for a graph $G=(V,E)$. For a vertex v in V, we denote by *ear* (v), the index of the lowest-numbered ear that contains v; for an edge $e=(x,y)$ in E, we denote by *ear* (e) (or *ear* (x,y)), the index of the unique ear that contains e. A vertex v will *belong to* $P_{ear(v)}$.

Let G be a biconnected graph with an open ear decomposition $D=[P_0, \cdots, P_{r-1}]$. Two ears are *parallel to each other* if they have the same endpoints; an ear P_i is a *parallel ear* if there exists another ear P_j such that P_i and P_j are parallel to each other.

An open ear decomposition can be obtained in logarithmic time with A-optimal performance [MR86, MSV86, Ra90]. The parallel open ear decomposition algorithm constructs a collection of auxiliary graphs in order to ensure that all ears are open. This construction is used several times in our planarity algorithm. We present the construction below. It will be used in sections 4 and 7.

```
set function auxgraphs (graph G=(V,E), spanning tree T=(V,D)) of graphs;
    vertex v, y, z, z', z''; edge e, e', e'', f;

    for each vertex v in parallel →
    {{Construct a graph H_v}}
        create a vertex for each edge of G incident on v;
        for each fundamental cycle C of T with v as the lca of the nontree edge in C in
        parallel →
            if C has two base edges e', e'' → create an edge (z',z'') in H_v where z' and z''
            are the vertices created to represent e' and e'' respectively
            {{Recall that the base edges of C are the edges in C that share the vertex v}}
             | C has only one base edge e → create an edge in H_v between y and z, where
            y and z are the vertices created to represent edges e and f respectively, f being
            the nontree edge in C
            fi
        rof
    rof;
    return {H_v | v∈V}
end;
```

An *st-numbering* of a graph G is a numbering of the n vertices of G from $s=1$ to $t=n$, such that every vertex v other than s and t has adjacent vertices u,w with $u<v<w$. Given an open ear decomposition $D=[P_0, \cdots, P_{r-1}]$ for a biconnected graph $G=(V,E)$ with $P_0=(s,t)$, it is possible to direct each ear in D from one endpoint to the other in such a way that the edge (s,t) is directed from s to t, the resulting directed graph is acyclic, and every vertex lies on a path

from s to t [MSV86]. Let G_{st} be this graph, which we will call the *st-graph of G*. The graph T_{st}, the *st-tree of G*, is the directed spanning tree obtained from G_{st} by deleting the last edge in each ear except P_0. We can similarly construct G_{ts} and its directed spanning tree T_{ts} by considering P_0 to be directed from t to s. We will refer to G_{ts} as the *reverse directed graph of* G_{st} and vice versa. These graphs can be obtained in logarithmic time with A-optimal performance using the algorithm in [MSV86].

The following two facts are well-known [Wh30, Ev79]:

1. A graph has an open ear decomposition if and only if it is biconnected.
2. A graph has an st-numbering if and only if it is biconnected.

2.6. The Local Replacement Graph

We describe a transformation of a biconnected graph G with an open ear decomposition $D=[P_0, \cdots ,P_{r-1}]$ into a new graph G_l, called the *local replacement graph* of $(G;D)$ [FRT89]. In the graph G_l, each ear P_i in D is converted into a path P'_i with P_i being P'_i with its end edges deleted. This treatment is from [Ra90].

Consider any vertex v in G. Let the degree of v be d $(d \geq 2)$. Of the d edges incident on v, two belong to $P_{ear(v)}$. Each of the remaining $d-2$ edges incident on v is an end edge of some ear P_j, with $j > ear(v)$. In the local replacement graph G_l we will replace v by a rooted tree with $d-1$ vertices, with one vertex for each ear containing v. The root of this tree will be the copy of v for the ear containing v. The actual form of the tree is computed from T_{st} and T_{ts} as in the algorithm below. The tree representing vertex v will be called the *local tree of* v and will be denoted by T_v.

Algorithm 2.1: Constructing the Local Replacement Graph

Input:

A biconnected graph $G=(V,E)$;

an open ear decomposition $D=[P_0, \cdots ,P_{r-1}]$ for G, with $P_0=(s,t)$;

the st-graph G_{st} with its spanning tree T_{st} and the ts-graph G_{ts} with its spanning tree T_{ts}.

Output: The local replacement graph G_l of $(G;D)$.

integer i,j; {{These integers range in value from 0 to $r-1$.}}
vertex a, q, u, v, w; {{q, u, v and w may be subscripted by an integer.}}
edge a, e, f, n; {{e and f will be subscripted by an integer.}}

rename each vertex v in G by v_j, where $ear(v)=j$;
{{We will refer to the vertex $v_{ear(v)}$ interchangeably as either v or $v_{ear(v)}$.}}

1. **for** each outgoing ear P_i at each vertex v in G_{st} **in parallel** $\rightarrow$

    let the edge in P_i incident on v be e_i and let the nontree edge in P_i be f_i;

    detach edge e_i from v and label the detached endpoint as v_i;

    let a be a base edge of the fundamental cycle created by f_i in T_{st} with $ear(a) \neq i$;

    if $ear(a) \leq ear(v) \rightarrow v_{ear(v)} := parent(v_i)$

    $\mid$ $ear(a) > ear(v) \rightarrow v_{ear(a)} := parent(v_i)$ **fi**;

    direct this edge from $parent(v_i)$ to v_i

    rof;

    let the undirected version of the graph obtained in step 1 be G^1, the directed version be G_{st}^1 and its associated spanning tree be T_{st}^1 and the reverse directed graph be G_{ts}^1 and its associated spanning tree be T_{ts}^1;

2. repeat step 1 using G_{ts}^1 and T_{ts}^1 and let the resulting undirected graph be G^2, the resulting directed graph be G_{ts}^2 and its associated spanning tree be T_{ts}^2, and the reverse directed graph be G_{st}^2 and its associated spanning tree be T_{st}^2;

    {{In the following we process parallel ears by constructing a new graph H.}}

    for each parallel ear P_i **in parallel** $\rightarrow$ **create** a vertex q_i **rof**;

    for each nontree edge n in T_{st}^2 **in parallel** $\rightarrow$

    if the base edges of the fundamental cycle of n belong to ears P_i and P_j, where P_i and P_j are parallel to each other $\rightarrow$ **create** an edge between q_i and q_j **fi**

    rof;

    call the resulting graph H;

    find a spanning tree in each connected component of H and root it at the vertex corresponding to the minimum numbered ear in the connected component;

3. **for** each vertex q_i in H that is not a root of a spanning tree **in parallel** $\rightarrow$

    let P_i be directed from endpoint u to endpoint w in G_{st}; let q_j be the parent of q_i in the spanning tree in H;

    replace the parent of u_i in T_{st}^2 by u_j and the parent of w_i in T_{ts}^2 by w_j

    rof;

    denote the undirected version of the graph formed in step 3 by G_l, the directed graph from s to t by G'_{st} and its associated spanning tree by T'_{st} and the reverse directed graph by G'_{ts} and its associated spanning tree by T'_{ts}; call G_l the *local replacement graph* of G;

    call the underlying undirected tree constructed in steps 1, 2 and 3 from each vertex v in G the *local tree* T_v*;* call $v_{ear(v)}$ the root of T_v, and consider T_v to be an out-tree rooted at $v_{ear(v)}$. Call the part of T_v constructed by assigning parents in T_{st}^2 the *o-tree* OT_v *of* T_v and the part of T_v constructed by assigning parents in T_{ts}^2 the *i-tree* IT_v *of* T_v*;*

{{In G_{st}^2, OT_v is an out-tree rooted at $v_{ear(v)}$ and IT_v is an in-tree rooted at $v_{ear(v)}$ and vice-versa in G_{ts}^2.}}

denote by P'_i the ear P_i, together with the edge connecting each endpoint of P_i to its parent in its local tree in G_l;

{{Note that the innard of P'_i (i.e., the path P'_i excluding its two end edges) is P_i.}}

denote the first vertex on P'_i when directed as in G'_{st} by $L(P'_i)$, *the left endpoint of* P'_i, and the last vertex on P'_i when directed as in G'_{st} by $R(P'_i)$, *the right endpoint of* P'_i.

end.

Figure 2 gives an example of the construction of the local replacement graph. For the rest of the paper we assume that the vertices in G_l, G'_{st} and T'_{st} are numbered with their st-numbering.

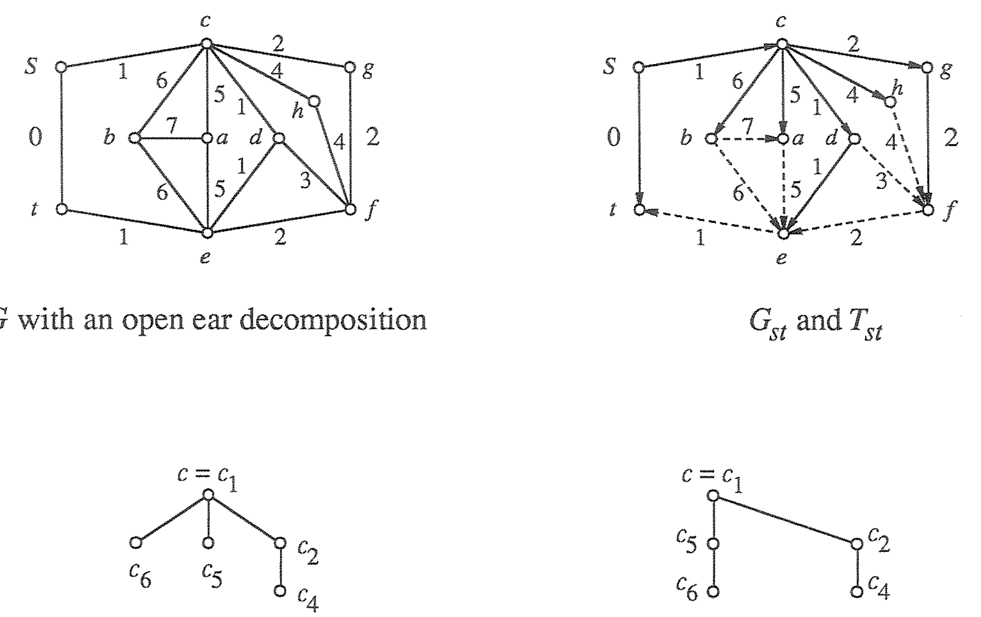
We will need the following lemma about the paths P'_i that are constructed in the local replacement graph G_l. The proof of this lemma is immediate if G_l contains no parallel ears. The proof for the case when G_l contains parallel ears is not difficult and is left as an exercise.

Lemma 2.1 There exists a permutation π of the indices 0 through $r-1$ such that $[P'_{\pi(0)}, \cdots, P'_{\pi(r-1)}]$ is an open ear decomposition for G_l.

3. Overview of the Planarity Algorithm

Our planarity algorithm finds an open ear decomposition $D = [P_0, \cdots, P_{r-1}]$ in the input graph G and derives from it the local replacement graph G_l together with its associated paths P'_i, $i=0, \cdots, r-1$ and its st-numbering directed graph G'_{st} and spanning tree T'_{st}. For each path P'_i, let C'_i be the fundamental cycle formed with respect to T'_{st} by the unique nontree edge in P'_i. The direction of C'_i will be the direction of P'_i in G'_{st}.

For each i, our algorithm finds certain approximations to the bridges of C'_i with attachments on P'_i, called the *bunches* of P'_i, together with a *hook* for each bunch. Some of the bunches are called the *anchor bunches* of P'_i; these are defined in the next section. The algorithm constructs a star graph J_i for each i, that (roughly speaking) consists of P'_i together with its bunches, and then forms $G_{i,J}$, the interlacing parity graph of J_i. The algorithm then links these graphs $G_{i,J}$ with some additional edges that are derived from the hooks of anchor bunches. This gives the *constraint graph* G^* which we describe in Section 5. The vertices of graph G^* are the union of the vertices in the $G_{i,J}$, together with some dummy vertices. We show that the graph G^* has the property that if G_l is planar, then any legal coloring of the vertices of G^* with $\{0,1\}$ gives a planar embedding of G_l with edge (s,t) on the outer face. This planar embedding is obtained by embedding bunch B inside C'_i if and only if the star vertex in $G_{i,J}$ corresponding to B is colored 0 in G^*. We then give a method to obtain the cyclic order of edges embedded inside C'_i, and of edges embedded outside C'_i, for each i. Finally we show that a planar embedding of G can be obtained from a planar embedding of G_l by collapsing the vertices in each local tree.



G with an open ear decomposition

G_{st} and T_{st}

Out-tree at vertex c
after step 1 of
Algorithm 4.2

Out-tree at vertex c
after step 3 of Algorithm 4.2
(P_2, P_5 and P_6 are parallel ears)

Figure 2
Constructing the local replacement graph.

All steps in our algorithm can be implemented in logarithmic time with A-optimal performance.

4. The Bunches and Their Hooks

Let G be a biconnected graph with an open ear decomposition $D=[P_0,P_1,\cdots,P_{r-1}]$ and let G_l, G'_{st}, T'_{st}, and P'_i, $i=0,\cdots,r-1$ be as described in Section 2.6. Let the vertices of G_l be numbered with their st-numbering.

Let C'_i be the fundamental cycle formed in G_l by adding the unique nontree edge in the path P'_i to T'_{st} and let l be the lca of the unique nontree edge in C'_i. Note that l is the lowest-numbered vertex in C'_i and $R(P'_i)$ is the highest-numbered vertex in C'_i. A *nonanchor bridge of* C'_i is a bridge of C'_i, all of whose attachments are internal vertices of P'_i. An *anchor bridge of* C'_i is a bridge of C'_i that has an attachment on an internal vertex of P'_i and either a) an attachment on $C'_i - P'_i$ or b) a nonattachment vertex v with $v < l$ or $v > R(P'_i)$. A *spanning bridge* of C'_i is a bridge of C'_i that has an attachment on $L(P'_i)$, has no attachment on $C'_i - P'_i$, and for each nonattachment vertex v has $L(P'_i) < v < R(P'_i)$.

4.1. The Bunches

The following algorithm finds certain objects that we call the *bunches* of the P'_i.

Algorithm 4.1: Forming the Bunches

Input:

Biconnected graph $G=(V,E)$;

an open ear decomposition $D=[P_0, \cdots, P_{r-1}]$ for G, with $P_0=(s,t)$;

the local replacement graph G_l of $(G;D)$, together with the associated G'_{st}, T'_{st} and the paths P'_i, $i=0, \cdots, r-1$.

Output: The bunches of each P'_i.

integer i, j, k; {{These integers range in value from 0 to $r-1$.}}

vertex $a, b, l, p, u, v, w, x, y, z$; {{The vertex p will be subscripted by an integer.}}

edge e, f, e_1, e_2;

set X, X' **of edges;**

1. in G'_{st} collapse the internal vertices of each path P'_i to form vertex p_i; let vertex t be p_0; call the resulting graph G''_{st}, and the resulting spanning tree derived from T'_{st} as T''_{st}; call the resulting underlying undirected graph G'';
2. $H := auxgraphs(G'', T''_{st})$; {{function *auxgraphs* is given in Section 2.5.}}

   let $H = \{H_i \mid i=1, \cdots, r-1\}$;

   {{H_i is the auxiliary graph whose vertices represent the edges incident on vertex p_i in G'' }};
3. **for** each i **in parallel** $\rightarrow$ compute the connected components of H_i and call the set of edges of G_l corresponding to the vertices in each connected component a *cluster of* P'_i;
4. **for** each nontree edge $e=(u,v)$ in G''_{st} whose fundamental cycle contains both base edges **in parallel** $\rightarrow$

   {{In the following we compute attachments to $L(P'_i)$, the left endpoint of P'_i for each P'_i.}}

   let $l=lca(u,v)$;

   let $e_1=(l,a)$ and $e_2=(l,b)$ be the two base edges of the fundamental cycle created by (u,v), with a an ancestor of u and b an ancestor of v; let $a=p_k$ and $b=p_j$;

   a. if edge e_2 is incident on $L(P'_k)$ in G_l $\rightarrow$

      if $u=a$ $\rightarrow$ add a copy of e_2 to X where X is the cluster of P'_k that contains e

| $u \neq a \rightarrow$ add a copy of e_2 to X', where X' is the cluster of P'_k that contains edge (a,y), where y is the unique child of a which is an ancestor of u

fi

fi;

b. **if** edge e_1 is incident on $L(P'_j)$ in $G_l \rightarrow$

{{This is symmetric to step a.}}

if $v=b \rightarrow$ add a copy of e_1 to X, where X is the cluster of P'_j that contains e

| $v \neq b \rightarrow$ add a copy of e_1 to X', where X' is the cluster of P'_j that contains edge (b,z), where z is the unique child of b which is an ancestor of v

fi

fi

rof;

$H' := auxgraphs(G_l, T'_{st})$;

let $H' = \{H_v \mid v \in G_l\}$;

5. **for** each P'_i **in parallel** $\rightarrow$

let $w = L(P'_i)$; let $ear(w)$ be j; let f be the edge in P'_i that is incident on w; let x be the vertex in H_w that represents the edge f;

for each connected component C in $H_w - \{x\} \rightarrow$ union all of the clusters of P'_i to which an edge corresponding to a vertex in C was added in step a or step b **rof**

rof;

denote each set of edges generated at the end of step 5 to be a *bunch*

{{Each bunch of P'_i is a set of attachment edges on P'_i of a bridge of C'_i in G_l. Some edges of G_l can appear in the bunches of several different P'_i because of step 4.

We will denote a bunch of P'_i by (B,i) where B denotes the set of edges in the bunch; if the index i is clear from the context we will let B denote the bunch.}}

end.

The following observation is a simple consequence of the construction of the local replacement graph.

Observation 4.1 In the graph G''_{st} constructed in step 1 of Algorithm 4.1,

a) Every outgoing edge from p_i is a tree edge in T''_{st} except for the unique outgoing edge that lies on P'_i.

b) Every incoming edge to p_i is a nontree edge except for the unique incoming edge that lies on P'_i.

The next lemma shows that every bunch of P'_i is part of a single bridge of C'_i.

Lemma 4.1 Let (X,i) be a bunch as computed in Algorithm 4.1. Then X is a subset of the attachment edges of a single bridge of C'_i. Further none of the edges in X appear in any other bunch of P'_i.

Proof It is straightforward to see that the edges in each cluster of P'_i as computed in step 3 belong to a single bridge of C'_i and that these clusters are disjoint. It is also straightforward to see that if edge e is added to cluster X in step 4a or step 4b then e belongs to the same bridge of C'_i as the edges in X. Also, all edges in G_l corresponding to vertices in a connected component of $H_w - \{x\}$ must belong to the same bridge of C'_i. Hence the sets of edges that are unioned in step 5 are all part of the same bridge of C'_i. Finally if (X,i) and (Y,i) are two bunches of P'_i with $X \neq Y$ they must be disjoint since the clusters in X are disjoint from the clusters in Y and the added edges in step 4 must also be disjoint, since otherwise, the X and Y would have been combined in step 5.[]

We will refer to a bunch (B,i) as a *nonanchor bunch*, an *anchor bunch* or a *spanning bunch*, depending on whether the bridge of C'_i that contains the edges in B is a nonanchor bridge, an anchor bridge or a spanning bridge of C'_i respectively.

We now give a series of observations about bridges and bunches. The proof of the first observation makes use of material from [Ra90].

Observation 4.2 Every spanning bridge of C'_i has attachments on both $L(P'_i)$ and $R(P'_i)$.

Proof It is shown in [Ra90] (Lemma 4.4) that any bridge of P'_i that has an attachment on $L(P'_i)$ either contains a nonattachment vertex on a path P'_j, $j<i$ or is a 2-attachment bridge with only the endpoints of P'_i as attachments. In the former case the bridge is called an *anchor bridge* of P'_i. Since P'_i is properly contained in C'_i, each bridge of C'_i is part of a bridge of P'_i. Hence any spanning bridge of C'_i is part of either an anchor bridge of P'_i or a bridge of P'_i that has attachments only on its two endpoints. In the latter case the Observation is immediate so for the rest of the proof we consider only the former case.

Let S be a bridge of C'_i that is part of an anchor bridge of P'_i and let l be the lca of the nontree edge in C'_i. If every nonattachment vertex in S is an internal vertex on a path P'_j, $j>i$, then S must have an attachment on $C'_i - P'_i$ and thus cannot be a spanning bridge of C'_i. Otherwise, let v be a vertex in S that is an internal vertex on a path P'_k, $k<i$. Consider the tree path p from s to v in T'_{st}. This path must intersect the path q from s to $L(P'_i)$ since $j<i$. Let the lowest point of intersection of p and q (i.e., $lca(v, L(P'_i))$) be x. If $x < L(P'_i)$ then either S is an anchor bridge of C'_i. If $x = L(P'_i)$, consider the edge e on p that is outgoing from $L(P'_i)$. Let e be contained in P'_m. By the construction of the local replacement graph, $R(P'_m)$ is not a descendant of an internal vertex of P'_i in T'_{st}, hence we again have the case that either S is an anchor bridge of C'_i or S has an attachment on $R(P'_i)$.[]

Observation 4.3 Let (X,i) be a spanning bunch of C'_i as computed in Algorithm 4.1. Then X has an attachment on $L(P'_i)$.

Proof Straightforward.[]

Observation 4.4 Let (B, i) be a nonanchor bunch of C'_i as computed in Algorithm 4.1. Then B is the set of all attachment edges of a nonanchor bridge of C'_i.

Proof Straightforward.[]

Lemma 4.2 Algorithm 4.1 can be implemented to run in logarithmic time with A-optimal performance.

Proof We first show the total size of all of the bunches computed by the algorithm is $O(n)$. Edges are added to the bunches in step 3 and in step 4 (a and b) of the algorithm. Each edge of G_l is added to at most one bunch in step 3 so the total number is $O(n)$. In step 4, at most two edges are added to bunches for each nontree edge, hence the total number is again $O(n)$.

We now analyze the performance of the algorithm. The major computation before step 5 is in finding connected components which can be performed in logarithmic time with A-optimal performance, and in finding lcas of pairs of vertices in a rooted tree which can be performed optimally in logarithmic time using the algorithm of [SV88]. For the case $u \neq a$ in step 4a and the case $v \neq b$ in step 4b we need the second edge on the path from the lca of a nontree edge to one of its endpoints; this can be computed optimally in logarithmic time using the lca algorithm of [SV88].

In step 5 we need to determine for each edge that was added to a cluster in step 4, the connected component to which it belongs in the graph $H_w - \{x\}$. For this, as a preprocessing step we construct the blocktree T' of the graph H' (see Appendix for the definition of blocktree). Each connected component of $H_w - \{x\}$ corresponds to a segment, starting and ending with x of the Euler tour of T'. Hence with some simple preprocessing of T' using the Euler tour technique, each vertex in H' can determine its connected component in the graph $H_w - \{x\}$, for any x, in constant time with one processor.

Step 5 also required several unions to be performed in parallel. For this, we create a triple (i, C, X) for each end attachment e on P'_i that is added to cluster X in step 4, where C is the connected component of $H_w - \{x\}$ that contains the vertex corresponding to e. We then sort these triples using the algorithm in [Ha87]. We form an auxiliary graph with a vertex for each cluster X formed in step 3 for each P'_i and we connect up all such vertices with identical second entry in the triple. Each connected component in the resulting graph together with the edges corresponding to the second entries in their triples gives a set corresponding to the union. This can be computed in logarithmic time with A-optimal performance.[]

4.2. The Hooks of Bunches

In this section we compute, for each bunch found in Algorithm 4.1, an additional edge called its *hook*. We first need some definitions. Recall that the vertices of T'_{st} are numbered with their st-numbering. For each edge $e = (parent(v), v)$ in T'_{st}, $out(e)$ is the set of nontree edges that are either incoming to or outgoing from a descendant of v; we define $low(e)$ to be $\min_{n \in out(e)} lca(n)$.

Let $S = \{n \mid n \in out(e) \text{ and } lca(n) \neq low(e)\}$. Then we define $low2(e)$ to be $\min(v, \min_{n \in S} lca(n))$. Note that $low2(e)$ is the second smallest vertex that is the lca of an edge in $out(e)$ if such a vertex exists.

For a nontree edge $n=(u,v)$ in G_l, we define $out(n) = \{n\}$, $low(n) = lca(n)$ in T'_{st} and $low2(n)$ to be $\max(u,v)$.

Let X be a set of edges in G_l. Then, we define

$out(X) = \bigcup_{e \in X} out(e)$; and

$low(X) = \min_{e \in X} low(e)$.

Let $l1(X) = \min_{e \in X \text{ and } low(e) \neq low(X)} (low(e))$ and $l2(X) = \min_{e \in X} low2(e)$.

Then, $low2(X) = \min(l1(X), l2(X))$.

Note that $low2(X)$ is the second smallest vertex that is the lca of an edge in $out(X)$ if such a vertex exists.

The computation of the hooks of the bunches is performed in two stages. In the first stage we find a hook for some of the bunches of P'_i using their low and $low2$ values. In the second stage we compute hooks for those bunches of P'_i for which the first stage found a hook that was incident on $L(P'_i)$ or $R(P'_i)$.

Algorithm 4.2 is specified in a manner that makes it simple to understand and to prove correctness. In order to obtain logarithmic time, A-optimal performance, we will modify the algorithm slightly. The details of the modification are given in the proof of Lemma 4.7.

Algorithm 4.2: Finding Hooks of the Bunches

Input:

The local replacement graph G_l of $(G;D)$;

the bunches of each P'_i;

Output: A hook for each bunch.

> **integer** i, j; {{i and j range from 0 to $r-1$.}}
>
> **edge** f, h; **vertex** u, v, w, x;
>
> **set** D, X **of edges; set** U **of vertices;**
>
> **bunch** (B, i);
>
> **edge function** *hook* (**set** X **of edges in** G_l, **integer** i);
>
> > **vertex** l, u, v, x, y, z;
> >
> > **edge** d, e, f, m, n;

$l := lca(d)$ in T'_{st}, where d is the nontree edge of P'_i;

1. **if** $low(X) < l \rightarrow$ **return** $(parent(l), l)$ **fi;**

$n :=$ an edge in $out(X)$ with $lca(n) = low(X)$;

2. **if** $L(P'_i) > low(X) > l \rightarrow n :=$ an edge in $out(X)$ with $lca(n) = low(X)$

3. | $low(X) = l$ **and** $L(P'_i) > low2(X) \rightarrow n :=$ an edge in $out(X)$ with $lca(n) = low2(X)$

4. | $low(X) = l$ **and** $L(P'_i) \leq low2(X)$ **and** there is an edge m in $out(X)$ not incident on $R(P'_i)$ with $lca(m) = l \rightarrow n := m$

fi

5. let $n=(x,y)$ and let $e=(u,v)$ be an edge in X that lies in the fundamental cycle of n with u contained in P'_i; let x be a descendant of v (and hence y is not a descendant of v); let f be a base edge of the fundamental cycle of n with f not lying on the path from s to z, where z is the vertex on P'_i adjacent to $R(P'_i)$;

6. **if** f is not contained in $C'_i \rightarrow$ **return** f

7. | $y > R(P'_i) \rightarrow$ **return** $(parent(l), l)$

8. | $y \leq R(P'_i) \rightarrow$ **return** (x, y)

fi

end *hook*;

{{Main program}}

$H := auxgraphs(G_l, T'_{st})$;

let $H = \{H_v \mid v \text{ is a vertex in } G_l\}$;

for each bunch (B, i) in G_l **in parallel** $\rightarrow$

1. $h := hook(B, i)$;

if h is incident on $L(P'_i)$ **or** $R(P'_i) \rightarrow$

{{This computation is similar to step 5 of Algorithm 4.1.}}

let $w = L(P'_i)$; let $ear(w)$ be j; let f be the edge in P'_i that is incident on w;

$D :=$ the set of edges in B incident on w;

let x be the vertex in H_w that represents the edge f;

let U be the set of vertices in H_w that represent the edges in D;

$X := \bigcup_{u \in U}$ (the set of edges in G_l that correspond to the vertices in the connected component of $H_w - \{x\}$ that contains vertex u);

2. $h := hook(X, i)$

fi

rof

end.

Lemma 4.3 Let X be a set of edges not contained in C'_i but with each edge in X incident on a vertex in P'_i. Function *hook* (X,i) returns an edge f in a bridge of C'_i that contains an edge in X.

Proof If edge f is returned in step 1 of function *hook* let e be an edge in X with $low(e) = low(X)$ and let n be an edge in $out(e)$ with $lca(n) = low(X)$. Since $lca(n) < l$, $lca(n)$ must be an ancestor of $parent(l)$. Then the path in G_l consisting of the path from $parent(l)$ to $lca(n)$ followed by the path from $lca(n)$ to n followed by the path from n to e in T'_{st} shows that f is an attachment edge of the bridge of C'_i that contains e.

If f is not returned in step 1, consider the nontree edge $n=(x,y)$ in step 5 of function *hook*. The edge n is in $out(X)$ and $lca(n) > l$. Hence the edge f as computed in step 5 is incident on a vertex in C'_i. If this edge f is returned in step 6 then the path from f to n, followed by the path from n to e in T'_{st} shows that f is in the same bridge of C'_i as edge e, which is in X.

If edge f is not returned in steps 1 or 6 then f is the base edge of C'_i that lies on the path from s to $R(P'_i)$. If edge $(parent(l),l)$ is returned in step 7 then the path from edge e to n in T'_{st}, followed by the path from y to t that contains vertices in increasing order of their st-numbering, followed by the path from s to $parent(l)$ is a path between e and edge $(parent(l),l)$ that avoids all vertices in C'_i. Further, since l is the lca of the nontree edge in C'_i the edge $(parent(l),l)$ is incident on a vertex in C'_i.

Finally if edge (x,y) is returned in step 8, then y is a vertex on C'_i on the path from l to $R(P'_i)$ and hence edge (x,y) is incident on a vertex in C'_i. In this case the path from edge e to x in T'_{st} is a path between e and edge (x,y) that avoids all vertices in C'_i.[]

Lemma 4.4 Let (B,i) be a bunch of P'_i and let h be its hook as calculated in Algorithm 4.2. Then h is an attachment edge of the bridge of C'_i that contains the edges in B.

Proof If the hook of B is computed in step 1 of the main program, then the result follows immediately from Lemma 4.3. If the hook of B is computed in step 2 then consider the set X in the function call *hook* (X,i) in step 2. Each edge e in X is incident on $L(P'_i)$ and is in the same bridge of C'_i as an edge of B that is incident on $L(P'_i)$. Hence X is a subset of the edges in the bridge of C'_i that contains B, and the result follows from Lemma 4.3[]

Corollary to Lemma 4.4 Let (B,i) be a nonanchor bunch with hook h. Then h is incident on an internal vertex of P'_i.

The following two lemmas deal with the hooks of anchor and spanning bunches.

Lemma 4.5 Let (B,i) be an anchor bunch of P'_i with hook h. Then either h is incident on a vertex in $C'_i - P'_i$ or $h = (parent(l),l)$, where l is the lca of the nontree edge of P'_i.

Proof The proof is divided into two cases depending on whether or not B has an edge incident on $L(P'_i)$.

CASE 1: B contains no edge incident on $L(P'_i)$. Since B is part of an anchor bridge of C'_i there must be a path p from an edge in B to either an attachment edge on $C'_i - P'_i$ or a vertex v with $v<l$ or $v>R(P'_i)$, with path p avoiding all vertices in C'_i. Further we can find such a path p with exactly one nontree edge n. The edge n is in $out(B)$ and $lca(n) < L(P'_i)$; further, n is not incident on $R(P'_i)$. Hence function $hook(B,i)$ will return either an edge incident on $C'_i - P'_i$ or the edge $(parent(l),l)$.

CASE 2: B contains an edge incident on $L(P'_i)$. In this case the path p of CASE 1 may contain several nontree edges with lca $L(P'_i)$ before reaching an attachment edge on $C'_i - P'_i$ or a vertex v not having a value between l and $R(P'_i)$. But the vertices in H_w corresponding to all of the base edges of the fundamental cycles of these nontree edges will be in a connected component of the graph $H_w - \{x\}$ (as computed in step a of the main program) that contains a vertex corresponding to an edge in B. Hence all of these nontree edges are included in $out(X)$ and hence the argument of CASE 1 applies with B replaced by X.[]

Lemma 4.6 Let (B,i) be a spanning bunch of P'_i with hook h. Then h is incident on $L(P'_i)$ or $R(P'_i)$.

Proof By Observation 4.3 B contains an edge e incident on $L(P'_i)$. Let e be contained in P'_k. By the construction of the local replacement graph, $R(P'_k)$ is not incident on a descendant of an internal vertex of P'_i. Hence B would be part of an anchor bridge of C'_i unless $R(P'_k) = R(P'_i)$. Hence $out(B)$ contains a nontree edge incident on $R(P'_i)$. The lca of this edge equals $low(B)$ since if $low(B)$ is smaller then B would be an anchor bridge. Hence either an edge incident on $R(P'_i)$ is returned in step 8 on function call $hook(B,i)$ and $hook(X,i)$ or an edge incident on $L(P'_i)$ is returned in step 6 on function call $hook(B,i)$ and $hook(X,i)$ (this could happen if $l = L(P'_i)$).[]

The following lemma gives bounds on the parallel complexity of Algorithm 4.2.

Lemma 4.7 Algorithm 4.2 can be implemented to run in logarithmic time with A-optimal performance.

Proof The low and $low2$ values for all edges can be computed optimally in logarithmic time using the Euler tour technique [TV85]. We can also compute with the same bounds a collection $Z(e)$ of two or three edges in $out(e)$ with lca equal to $low(e)$ and such that for any vertex v in G_l one of these edges is not incident on v (if such a collection of edges exists in $out(e)$). This computation allows us to find in constant time, an edge in $out(e)$, not incident on $R(P'_i)$, and with lca equal to $low(e)$, as needed in step 4 of function hook. Once these values are known for set X, all steps of any single call to function $hook(X,i)$ can be computed in constant time with one processor.

The total size of all of the sets in the function call in step 1 of the main program is linear in the size of G_l, hence this step can be performed optimally in logarithmic time using the preprocessing described above. However, the total size of all of the sets in the function call in step 2 of the main program can be superlinear in the size of G_l since the graph H_w will be needed in processing the bunches of several different P'_i. In order to obtain A-optimal performance, we will preprocess by computing the *low*, *low* 2 and Z values for each block in each H_w. As in the proof of Lemma 4.2, the corresponding values for each connected component in $H_w - \{x\}$ can be computed from a segment of the Euler tour of the block tree of H_w. In our parallel implementation of Algorithm 4.2 we will pass only the *low*, *low* 2 and Z values to function *hook* rather than the entire set of edges. This results in a parallel algorithm that runs in logarithmic time with A-optimal performance.[]

4.3. The Bunch Graphs

Let Q_i be the path P'_i together with an edge from $L(P'_i)$ to a new vertex $U(P'_i)$. In the following we define for each path P'_i in G_l a star graph, $J_i(Q_i)$, called the *bunch graph of* P'_i. We create a star S_B for each bunch B of P'_i by creating a new vertex v_B and adding attachment edges as follows: we replace each edge (x,y) in B with y not on P'_i by the edge (x,v_B). If B contains an edge (x,y) with both x and y on P'_i then we replace (x,y) by two edges (x,v_B) and (y,v_B). If B is an anchor bunch we include an attachment edge $(U(P'_i),v_B)$ to represent the hook. If B is a spanning bunch we include an attachment edge $(R(P'_i),v_B)$. The center of star S_B is v_B and each edge in S_B corresponds to an attachment edge of B on a vertex in C'_i.

The *bunch graph* $J_i(Q_i)$ is the star graph consisting of the path Q_i, together with the star S_B for each bunch B of P'_i.

In the next section we will use the interlacing parity graph (defined in Section 2.4) of each $J_i(Q_i)$. We will denote this interlacing parity graph by $G_{i,J}$. Recall that the graph $G_{i,J}$ contains vertices for certain chords derived from the stars in $J_i(Q_i)$ as well as a vertex for each star of $J_i(Q_i)$. We will refer to the latter vertices as *bunch vertices* and we will denote the bunch vertex corresponding to (B,i) by $u_{B,i}$.

We now associate a triconnected component of G_l with each bunch (see Appendix for definitions relating to triconnected components). Recall that by Lemma 2.1 it is possible to rearrange the P'_i so that the resulting sequence of paths forms an open ear decomposition for G_l. We assume that the P'_i have been reordered so that $[P'_0, \cdots, P'_{r-1}]$ forms an open ear decomposition for G_l. If G_l contains no pair a,b separating P'_i such that the interval $[a,b]$ on P'_i contains all attachment vertices of B then let v be an attachment of B and let X be the triconnected component of G_l that contains the copy of v that remains when all upper split graphs corresponding to ear splits on adjacent and extremal separating pairs on $P'_j, j \geq i$ have been removed. If G_l contains a pair a,b separating P'_i such that the interval $[a,b]$ on P'_i contains all attachment vertices of B then let x,y be such an adjacent separating pair whose upper split graph does not contain any other adjacent pair of this form and let $X = TC(x,y,i)$. Then X is the *triconnected component of* (B,i), or equivalently, (B,i) *belongs to triconnected component* $TC(x,y,i)$.

The following lemma follows from the results of [MR87, Ra90] relating separating pairs on P'_i to the interlacings of stars in the bridge graph of P'_i.

Lemma 4.8

a) Let X be a connected component of $G_{i,J}$ that contains no bunch vertex corresponding to an anchor bunch and let Y be the triconnected component of a bunch whose bunch vertex is in X. Then Y is the triconnected component of a bunch (B,i) if and only if the bunch vertex $u_{B,i}$ is in X.

b) All bunches corresponding to bunch vertices in connected components of $G_{i,J}$ that contain an anchor bunch belong to a single triconnected component of G_l, and this triconnected component is not part of the upper split graph of any ear split corresponding to a pair separating P'_i.

5. The Constraint Graph

In this section we define the *constraint graph* G^* of G_l. G^* consists of two parts. One part consists of the union over all i of the interlacing parity graph, $G_{i,J}$, of the bunch graph $J_i(Q_i)$. The other part of G^* consists of certain edges linking the $G_{i,J}$, incorporating some additional *dummy* vertices as described below. The link edges are determined by the hooks of anchor bunches that we computed in Algorithm 4.2. In Section 6 we relate 2-colorings of G^* to planar embeddings of G_l.

Algorithm 5.1: Forming the Links of the Constraint Graph

Input:

A biconnected graph G with an open ear decomposition $D=[P_0, \cdots, P_{r-1}]$;

the local replacement graph G_l of $(G;D)$ together with G'_{st} and T'_{st};

the bunches of each P'_i in G_l;

a hook for each anchor bunch;

the interlacing parity graph $G_{i,J}$ for each bunch graph $J_i(Q_i)$.

Output: The constraint graph G^* of G_l.

> **integer** i, j, k, m; {{The range of the integers is from 0 to $r-1$.}}
>
> **vertex** l, p, q, w, x, y, z; **edge** e, f, h, n;
>
> **bunch** (A,j), (B,i);
>
> **procedure** *odd*(**bunch** (B,i), (A,j));
>
> > **vertex** $u_{B,i}$, $u_{A,j}$;
> >
> > **create** an edge between the bunch vertex $u_{B,i}$ in $G_{i,J}$ and the bunch vertex $u_{A,j}$ in $G_{j,J}$
> >
> > {{We will refer to this edge as the *link path between the vertices $u_{B,i}$ and $u_{A,j}$*.}}

end *odd*;

procedure *even* (**bunch** (B, i), (A, j));

vertex v, $u_{B,i}$, $u_{A,j}$;

create a vertex v; {{We will refer to v as a *dummy* vertex.}}

create an edge between the vertex v and the bunch vertex $u_{B,i}$ in $G_{i,I}$;

create an edge between the vertex v and the bunch vertex $u_{A,j}$ in $G_{j,I}$

{{We shall refer to the path of length 2 formed by the two newly-created edges as the *link path between the vertices $u_{B,i}$ and $u_{A,j}$.*}}

end *even*;

{{Main program}}

for each anchor bunch B of each P'_i **in parallel** $\rightarrow$

let $n=(p, q)$ be the unique nontree edge in P'_i with $q=R(P'_i)$;

let $lca(n)=l$;

let h be the base edge of C'_i that lies on the path from l to q and let e be the other base edge of C'_i;

let $hook(B)$ be $f=(x, y)$;

1. **if** $f = (parent(l), l) \rightarrow$

   let $ear(h)=j$;

   A := the set of edges in the bunch of P'_j that contains edge e;

   {{Note that e must be contained in a bunch of P'_j even if $L(P'_j)=l$ because of the presence of nontree edge n whose fundamental cycle contains e and h as its base edges.}}

   even $((B, i), (A, j))$

2. | f is an edge incident on a proper ancestor y of $L(P'_i) \rightarrow$

   let w be the unique child of y on the tree path from l to $L(P'_i)$; let (y, w) be an edge in P'_j;

   A := the set of edges in the bunch of P'_j that contains edge f;

   even $((B, i), (A, j))$

3. | f is incident on a vertex y in the path from l to q $\rightarrow$

   let z be $parent(y)$ in T'_{st} and let w be the unique child of y on the tree path from l to q;

   let (y, z) be contained in P'_j, let (w, y) be contained in P'_k, let (x, y) be contained in P'_m;

a. **if** $m \neq j \rightarrow$

A := the set of edges in the bunch of P'_j that contains edge f ;

$odd((B,i), (A,j))$

b. $|\ m = j \rightarrow$

A := the set of edges in the bunch of P'_j that contains edge (w,y);

$even((B,i), (A,j))$

fi

fi

rof

end.

Lemma 5.1 Algorithm 5.1 can be implemented to run optimally in logarithmic time.

Proof Straightforward.[]

We now relate the link paths created in Algorithm 5.1 to a planar embedding of G_l. We first review some definitions. Let G_d be the directed graph obtained from the local replacement graph G_l by replacing each (undirected) edge (u,v) in G_l by two directed edges (u,v) and (v,u). Let $I(G_d)$ be a cyclic ordering of the edges outgoing from each vertex in G_d corresponding to a planar embedding of G_l. Given a directed path $P'_i = <v_0, v_1, \cdots, v_b>$ in G'_{st}, and an attachment edge (v_i,x) where x is not in P'_i but v_i is an internal vertex on P'_i, recall that in Section 2.2 we defined (v_i,x) to be embedded inside P'_i (and otherwise outside P'_i) if, in the cyclic order defined by $I(G_d)$ on the edges outgoing from vertex v_i, edge (v_i,x) appears after edge (v_i,v_{i+1}) and before edge (v_i,v_{i-1}). Thus (v_i,x) is defined to be embedded inside P'_i if and only if (v_i,x) is embedded inside the basis cycle C'_i.

The link paths introduced in Algorithm 5.1 are either of length 1 or length 2. The length of a link path is determined by the relative placements of the two bunches it connects with respect to their fundamental cycles in a planar embedding of G_l as described in the following lemma.

Lemma 5.2 Let (A,j) and (B,i) be a pair of bunches in G_l whose corresponding vertices u and v in G^* are connected by a link path p in G^*.

If G_l is planar and $\hat{G_l}$ is a planar embedding of G_l with edge (s,t) on the outer face then

a) If $p = <u,v>$ then in $\hat{G_l}$ the edges in bunch A are embedded inside C'_j if and only if the edges in bunch B are embedded outside C'_i.

b) If $p = <u,d,v>$ where d is a dummy vertex created by procedure *even*, then in $\hat{G_l}$ the edges in bunch A are embedded inside C'_j if and only if the edges in bunch B are embedded inside C'_i.

Proof The path p must have been introduced in step 1, 2, a or b of Algorithm 5.1. These four cases are shown in figure 3. We verify the lemma only for step a (the other steps are easier to verify).

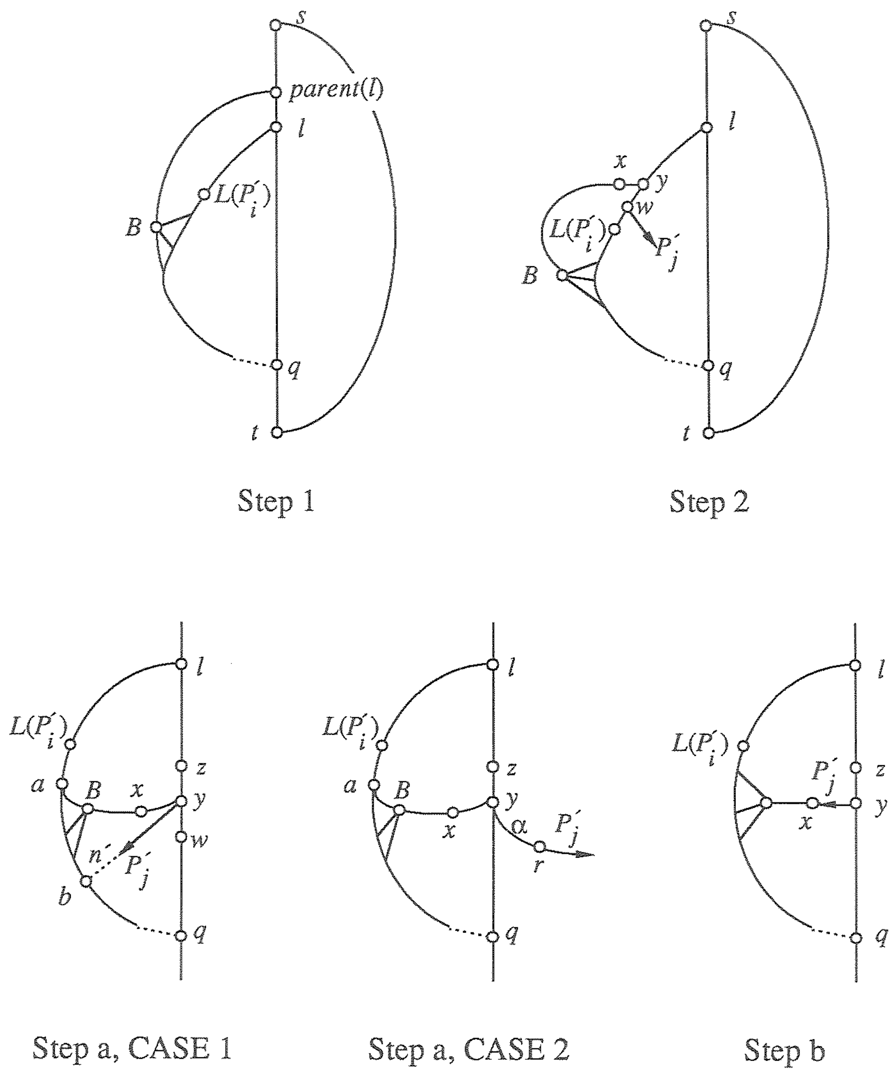


Figure 3
Illustrating the four cases in the proof of Lemma 5.2.

Let p be introduced in step a of Algorithm 5.1. Let n' be the nontree edge in C'_j. Let p' be the path in T'_{st} from s to x and let a be the last vertex on p' that is a descendant of $L(P'_i)$.

CASE 1: n' is incident on a descendant b of $L(P'_i)$. Since G'_{st} is acyclic, the vertex b must be a proper descendant of a. Let B' be the bridge of C'_j that contains edge (x,y) and let X be the bridge of C'_j that contains edge (s,t). The bridges B' and X interlace since B' has attachments on a and y and X has attachments on l and b. The bridge X contains the edge (y,w).

Assume that bridge B' is embedded inside C'_j (as shown in the figure). Hence edge (y,w) is embedded outside C'_j. Hence edge (x,y) is embedded inside the path from l to q in T'_{st}. However, in C'_i, this path is traversed in the direction from q to l. Edge (x,y) is embedded outside the path from q to l and hence the bunch B is embedded outside C'_i. A similar argument shows that if B' is embedded outside C'_j then the bridge containing (x,y) is embedded inside C'_i.

CASE 2: n' is not incident on a descendant of $L(P'_i)$. The edge n' cannot be incident on an ancestor of $L(P'_i)$ since G'_{st} is acyclic. Let $\alpha=(y,r)$ be the edge following edge (z,y) on P'_j. Let X be the bridge of C'_i that contains edge (x,y) and let Y be the bridge of C'_i that contains edge α. The bridges X and Y interlace on C'_i. This is because X has attachments on a and y and Y has attachment on $lca(n')$ which is a proper ancestor of y and an attachment on a vertex numbered larger than a or y by the st-numbering property.

In C'_i the path between l and q is directed from q to l. In C'_j the path between l and r is directed from l to r. Without loss of generality assume that X is embedded outside C'_i (as shown in the figure). Then Y is embedded inside C'_i. Hence edge (x,y) is embedded outside the path from l to r, i.e., inside the path from r to l. Thus the bridge of C'_j that contains edge (x,y) is embedded inside C'_j if (x,y) is embedded outside C'_i.[]

6. Planar Embeddings via 2-Colorings

In this section we correlate 2-colorings of the constraint graph G^* with planar embeddings of G_l. A 2-coloring of G^* assigns to each vertex a value (or *color*) in $\{0,1\}$ such that no two adjacent vertices are assigned the same color. This can be done A-optimally in logarithmic time by a simple algorithm (see , e.g., [Ra90]).

Observation 6.1 Let G_l be planar. In any planar embedding of G_l the edges in a bunch (B,i) are either all embedded inside P'_i or all embedded outside P'_i.

Proof The edges in B are a subset of a bridge of the cycle C'_i. Hence all edges in B must be embedded on one side of C'_i, and thus on one side of P'_i.[]

We now relate 2-colorings of the constraint graph G^* to planar embeddings of G_l.

Lemma 6.1 Let G_l be planar and let $\hat{G}_l$ be a planar embedding of G_l with edge (s,t) on the outer face. Then any 2-coloring of G^* that assigns a bunch vertex $u_{B,i}$ the color 0 if and only if the corresponding bunch (B,i) was embedded inside P'_i in $\hat{G}_l$ can be extended into a valid 2-coloring of G^*.

Proof By the results in [Ra90] we know that the two coloring can be extended to a valid two coloring for the graph $\bigcup_{i=1}^{r-1} G_{i,I}$. So we only need to verify that the coloring can be extended to the dummy vertices and that the link edges do not destroy the validity of the two-coloring. The result follows from Lemma 5.2 since the link edges only connect bunch vertices and dummy vertices and they force a pair of bunch vertices to be given the same color if and only if the corresponding bunches have to be embedded on the same side of their fundamental cycles.[]

Lemma 6.2 A pair of bunch vertices $u_{A,i}$ and $u_{B,j}$ lie in the same connected component of G^* if and only if bunches (A,i) and (B,j) belong to the same triconnected component of G_l.

Proof The proof is by induction on i and is applied to the bunches corresponding to the bunch vertices in the subgraph D_i of G^* induced on $\bigcup_{j\le i} G_{j,I} \cup$ {dummy vertices linking bunch vertices $u_{X,k}$, $u_{Y,l}$, $k<l\leq i$ }.

BASE: $D_1 = G_{1,I}$. By Lemma 4.8, the bunch vertices in D_1 satisfy the statement of the lemma, since P'_1 has no anchor bunches.

INDUCTION STEP: Assume that the result is true until $i-1$ and consider D_i. D_i is D_{i-1} together with $G_{i,I}$ and the link paths connecting vertices in $G_{i,I}$ to vertices in D_{i-1}. By construction the vertices corresponding to each anchor bunch incident on P'_i are connected to D_{i-1} by the link edges. By Lemma 4.8, if the connected components in $G_{i,I}$ that contain a bunch vertex corresponding to an anchor bunch are connected to one another then a pair of bunches (A,i) and (B,i) are in the same connected component of G^* if and only if the bunches belong to the same triconnected component of G_l. This verifies the induction step for bunches of P'_i.

To establish the induction step for bunches in P'_j, $j<i$, we need to verify the lemma for pairs of bunches (A,i) and (B,j), $j<i$, whose bunch vertices are connected by a link path. Let (A,i) be an anchor bunch of P'_i and let $u_{A,i}$ be connected to $u_{B,j}$, $j<i$ by a link path in G^*. Let x be an attachment vertex of bunch (A,i) on P'_i and let y be an attachment vertex of bunch (B,j) on P'_j. To show that (A,i) and (B,j) belong to the same triconnected component it suffices to show that there are 3 vertex-disjoint paths between x and y in G_l. For this we note that x and y are vertices on the fundamental cycle C'_i. This gives two vertex-disjoint paths between x and y on C'_i. Further by Lemma 4.4 x and y are attachments of the bridge X of C'_i that contains the edges in A. This provides the third vertex-disjoint path between x and y.

This concludes the induction step and the lemma is proved.[]

We now state and prove the main result of this section.

Theorem 6.1 Let G_l be biconnected and planar and let X be a 2-coloring of G^*. Then there exists a planar embedding of G_l with the edge (s,t) on the outside face that embeds a bunch (B,i) inside P'_i if and only if the bunch vertex $u_{B,i}$ in G^* is colored 0.

Proof The proof is by induction on the number of triconnected components in G_l.

BASE: G_l is triconnected. Then by Lemma 6.2 G^* is connected. Hence G^* has exactly two different 2-colorings, and each can be obtained from the other by interchanging zeros and ones. By Lemma 6.1 these two colorings must correspond to the two possible embeddings of G_l with (s,t) on the outer face.

INDUCTION STEP: Assume the lemma is true for up to $k-1$ triconnected components and let G_l have k triconnected components. Assume without loss of generality that the indices of the P'_i have been permuted so that $D=[P'_0, \cdots ,P'_{r-1}]$ forms an open ear decomposition for G_l (such a rearrangement was shown to exist in Lemma 2.1). Let x,y be a nontrivial adjacent pair separating P'_i and let G_1 and G_2 be the upper and lower split graphs obtained by the ear split (x,y,i). The open ear decomposition D induces an open ear decomposition D_1 in G_1 and D_2 in G_2, with the newly-introduced edge (x,y,i) serving as the initial ear in D_2 [MR87, Ra90]. Each triconnected component of G is contained entirely within one of G_1 or G_2, hence the connected components of G^* can be partitioned between G_1 and G_2. Further, each of G_1 and G_2 contains at most $k-1$ triconnected components, hence the induction hypothesis applies to both of them.

Let $\hat{G}_1$ and $\hat{G}_2$ be planar embeddings of G_1 and G_2 respectively, that are induced by the 2-coloring X. We only need to verify that these two embeddings can be combined into a planar embedding for G. In G_l the embeddings for G_1 and G_2 interact only on P'_i. However none of the bunches of P'_i in G_2 interlace with any of the bunches of P'_i in G_1 [MR87, Ra90]. Also, since x and y serve the place of s and t in G_2, $\hat{G}_2$ has x and y on the outer face. Hence $\hat{G}_1$ can be combined with $\hat{G}_2$ at x and y to form a planar embedding for G_l.[]

7. The Combinatorial Embedding

By Theorem 6.1 we can determine for each P'_i, the set of bunches that are embedded inside P'_i and the set that is embedded outside P'_i. In this section we show how to determine the relative ordering of the edges incident on a vertex in P'_i that are assigned to one side of P'_i. In Section 7.1 we describe this procedure for the local replacement graph G_l. In Section 7.2 we map this ordering back to the input graph G.

7.1. The Combinatorial Embedding of the Local Replacement Graph

In order to obtain a combinatorial embedding of G_l we need to obtain for each vertex v in G_l, the cyclic ordering of the edges incident on v in a planar embedding of G_l. In Section 6 we obtained some coarse information on this cyclic ordering, i.e., for each vertex v in P'_i we partitioned the set of edges incident on v (other than the two edges in P'_i that are incident on v) into two classes, those that are embedded inside P'_i and those that are embedded outside P'_i. In this section we obtain the cyclic ordering for each of these two sets. Since the procedure is identical for each of these two sets we describe only the procedure for the edges embedded inside P'_i.

A *segment* of a cycle C in a graph G is a connected subgraph of a bridge of C. Two segments of C are *disjoint* if their nonattachment vertex sets are disjoint. For instance, each bunch (B,i) is a segment of C'_i and two bunches (B,i) and (B',i) with $B \neq B'$ are disjoint. We will work with segments in this section, and the following observation and its corollary will be used.

Observation 7.1 If two disjoint segments of a cycle C in a graph G interlace then they must be placed on opposite sides of C in any planar embedding of G.

Corollary to Observation 7.1 If two disjoint segments of C that are derived from the same bridge of C interlace then G is nonplanar.

In the following we will assume, as before, that the vertices of G_l, G'_{st}, and T'_{st} are numbered with their st-numbering. Given an edge $e=(u,v)$ with $u>v$, the vertex u will be called the *high endpoint of e* and the vertex v will be called the *low endpoint of e*.

Observation 7.2 Let $n=(x,y)$ be a nontree edge in G'_{st} with respect to the tree T'_{st}. Let $x>y$. Then x is the largest-numbered vertex in the fundamental cycle of n.

Let v be an internal vertex on P'_i and let F be the set of edges incident on v that are embedded inside P'_i. Let $F=F_1 \cup F_2$ where F_1 is the set of edges in F that lie in the tree T'_{st} and F_2 is the set of remaining edges in F. We first obtain the cyclic ordering of edges in F_1 (for all vertices v) and then find the cyclic ordering of edges in F_2. The following lemma shows that all edges in F_1 must appear before any edge in F_2 in a cyclic ordering corresponding to a planar embedding of G_l. Hence we can concatenate the cyclic ordering of F_1 and F_2 to obtain the cyclic ordering of F.

Lemma 7.1 Let v be an internal vertex of P'_i in the local replacement graph G_l. Let e and e' be edges incident on v that are embedded inside P'_i in a planar embedding of G_l with e an edge in T'_{st} and e' a nontree edge. Let f be the unique incoming edge to v and g the unique outgoing edge from v in G'_{st} that are contained in P'_i. Then edge e appears before edge e' in the cyclic ordering of edges incident on v, starting with edge g.

Proof Let e be contained in P'_j and let $n=(u,w)$ be the nontree edge in P'_j. Let $w>u$. By Observation 7.2 the vertex w is the largest-numbered vertex in the fundamental cycle C'_j. Hence there is a path p from w to t that avoids all other vertices on C'_j including vertex v.

Let C be the cycle in G_l consisting of the path in T'_{st} from s to v, followed by the path q in C'_j from v to w that contains edge e, followed by the path p, followed by edge (t,s). Let this cycle have the direction of edge f (which is the same as that of edge e). Edge g is embedded outside C since e is embedded inside C'_i.

Let $g=(v,y)$. There is a path from y to t that contains vertices in increasing order of their st-numbering. Hence the bridge B of C containing edge g must have an attachment on a vertex $x \neq v$ that lies on path q or path p. Now consider the bridge B' of C that contains edge e'. Let m be the base edge of the fundamental cycle of e' not lying on C. The edge m is an attachment edge of B' and the attachment vertex, which is $lca(e')$, does not lie on either path p or path q since $lca(e')$ is a proper ancestor of v. If B' is embedded outside C then it must appear before g in the cyclic ordering starting with f. This is not possible since this would cause e' to be embedded outside C'_i. Hence e' is embedded inside C which means that e appears before e' in the cyclic ordering of edges incident on v, starting with edge g.[]

We now describe how to obtain the cyclic ordering of the tree edges that are attachment edges on an internal vertex v in P'_i. We will compute this ordering in two phases. The first phase makes use of the following lemma.

Lemma 7.2 Let v be an internal vertex on the path P'_i. Let H_v be the graph obtained for vertex v using the function call *auxgraphs* (G_l,T'_{st}) (from section 2.5). Let X_l, $l=0$ to k be the connected components of $F=H_v-\{z\}$, where z is the vertex in H_v representing the unique outgoing edge from v that is contained in P'_i. Let S_l be the simple graph obtained from X_l, for each l, by deleting multiple edges. Then each S_l is a simple noncyclic path.

Proof Let z' be a vertex in S_l whose corresponding edge in G_l is e'. Let (z',z'') be an edge in S_l with e'' being the edge in G_l corresponding to z'' and let n be a nontree edge in G_l that caused edge (z',z'') to be placed in H_v. Let e' be contained in P'_j and let (u,w) be the nontree edge in P'_j with $w>u$; by Observation 7.2 w is the largest-numbered vertex on C'_j. By the construction of G_l the fundamental cycle C'_j does not not contain e''.

Now consider the bridge B of C'_j that contains the attachment edge e''. The fundamental cycle of n will contribute an attachment edge for B on C'_j on a vertex x where $x \neq v$ and $x \neq w$. Further, if e'' is on P'_k then the fundamental cycle C'_k will contribute an attachment edge for B either on vertex w on C'_j or on a proper ancestor of v on C'_j. This results in a segment S of B that has 3 or more attachments on C'_j.

We have shown above that each edge (z', z'') in H_v results in a segment of C'_j that contains at least 3 attachments. The segments corresponding to different z'' are disoint. Any two segments with 3 attachments interlace on a cycle, and hence must be placed on opposite sides of the cycle in a planar embedding by Observation 7.1. Hence z' can have at most two neighbors in $H_v - \{z\}$. Hence each connected component of F must be a simple path. Finally, all of the edges in a connected component of v must be placed on the same side (either inside or outside) of C'_i. Hence no connected component of F is a simple cycle.[]

Lemma 7.3 Let X be a connected component of the graph $F = H_v - \{z\}$, as defined in the statement of Lemma 7.2 and let X be the path $<x_0, \cdots, x_k>$. Let the edge in G_l corresponding to x_l be e_l. Then in any planar embedding of G_l the cyclic ordering of the edges incident on v will contain the e_l as consecutive edges in order from e_0 to e_k or from e_k to e_0.

Proof Let v be an internal vertex of P'_i and let e_l be contained in P'_j. Then by the construction of the local replacement graph, the nontree edge in P'_j is not incident on a descendant of an internal vertex of P'_q, for any P'_q that contains one of the e_r. Hence the edges e_0 to e_{l-1} appear in one bridge of C'_j and the edges e_{l+1} to e_k appear in another bridge. This holds for each e_l, $l = 0$ to k. Hence in any planar embedding of G_l the edges e_0 to e_k appear in that order in the cyclic order of edges outgoing from v.

We now show that the e_l must occur as consecutive edges in the cyclic order. Let e be an outgoing edge from vertex v other than the e_l, and let e be contained in P'_m. Since $R(P'_m)$ is not incident on a descendant of an internal vertex of P'_q, for any P'_q that contains one of the e_l, all of the e_l are in a single bridge of C'_m. Hence they must all appear on one side of C'_m, i.e., the edge e cannot appear between the e_l in the cyclic ordering.[]

We will call each set of edges in G_l corresponding to vertices in a connected component of F (as defined in the statement of Lemma 7.2) a *tuft of vertex* v. In phase 2 of the algorithm to find the cyclic ordering of tree edges outgoing from v we determine the ordering of the tufts that are embedded inside P'_i. To do this, we determine, for each tuft S of v, an edge n in $out(S)$ with $lca(n) < v$ and we embed the tufts in decreasing order of the high endpoint of this edge. The following lemma shows that if the high endpoints of the edges chosen for different tufts are all distinct this will give us the correct ordering of the tufts.

Lemma 7.4 Let e', e'' be two tree edges outgoing from v that are embedded inside C'_i. Let $n' \in out(e')$ and $n'' \in out(e'')$ with $lca(n') < v$ and $lca(n'') < v$ and let u' and u'' be the high endpoints of n' and n'' respectively. If $u' > u''$ then e' is embedded before e'' in the cyclic ordering starting with g, the unique outgoing edge from v that lies on P'_i.

Proof Let C'' be the fundamental cycle of n''. The edge g is embedded outside C'' since e'' is embedded inside C'_i. Since $lca(n'') < v$, the vertex t is in the same bridge of C'' as edge g and hence t is embedded outside C''. By Observation 7.2 u'' is the highest-numbered vertex in C'' and hence by the st-numbering property any vertex x with $x > u''$ must be in the same bridge of C'' as vertex t. Hence vertex u' and edge e' are embedded outside the cycle C'' in the planar embedding, i.e., edge e' is embedded before e'' in the cyclic ordering starting at edge g.[]

In order to handle the case when the chosen edges for different tufts have the same high endpoint we choose two different nontree edges for each tuft. These edges are chosen by a strategy somewhat similar to the one used to find hooks for the bunches. We first present some definitions. These definitions are similar to the definitions of *low* and *low*2 given in Section 4.2, except that we now distinguish between outgoing nontree edges and incoming nontree edges.

Let the vertices of T'_{st} be numbered in st-numbering. For each edge $e = (parent(v), v)$ in T'_{st}, $a.out(e)$ is the set of nontree edges that are outgoing from a descendant of v and $b.out(e)$ is the set of nontree edges that are incoming to a descendant of v; note that $a.out(e)$ and $b.out(e)$ are disjoint and $out(e)$ as defined in Section 4.2 is $a.out(e) \cup b.out(e)$. We define $a.low(e)$ to be $\min_{n \in a.out(e)} lca(n)$ and $b.low(e)$ to be $\min_{n \in b.out(e)} lca(n)$.

Let $a.S = \{n \mid n \in a.out(e) \text{ and } lca(n) \neq a.low(e)\}$. Then we define $a.low2(e)$ to be $\min(v, \min_{n \in a.S} lca(n))$.

Let X be a set of edges in T'_{st}. Then, we define

$a.out(X) = \bigcup_{e \in X} a.out(e)$;

$b.out(X) = \bigcup_{e \in X} b.out(e)$;

$a.low(X) = \min_{e \in X} a.low(e)$;

$b.low(X) = \min_{e \in X} b.low(e)$; and

Let $a.l1(X) = \min_{e \in X \text{ and } a.low(e) \neq a.low(X)} (a.low(e))$ and $a.l2(X) = \min_{e \in X} a.low2(e)$.

Then $a.low2(X) = \min(a.l1(X), a.l2(X))$.

In the following algorithm we find for each tuft S of v, two vertices $big(S)$ and $nextbig(S)$ which are high endpoints of edges in $out(S)$. We use these to compute the cyclic ordering of the tufts of each vertex.

Algorithm 7.1: Finding the Cyclic Ordering for the Tufts

Input Graphs G_l, tree T'_{st}, and the tufts for each vertex.

Output For each vertex v, the cyclic ordering of its tufts that are embedded inside the fundamental cycle of the path P'_i that contains v as an internal vertex (vertices s and t are assumed to be internal vertices of P'_0).

vertex u, u', v; **edge** n, n';

tuft S;

1. **for** each tuft S of each vertex v **in parallel** $\rightarrow$

   $big(S) :=$ the high endpoint u of an edge n in $a.out(S)$ with $lca(n) = a.low(S)$;

   $nextbig(S) := v$;

   if there is an edge n' in $a.out(S)$ with $lca(n) = a.low(S)$ and with high endpoint $u' \neq u \rightarrow nextbig(S) := u'$

   $|$ $a.low2(S) < v \rightarrow nextbig(S) :=$ the high endpoint of an edge in $a.out(S)$ with $lca(n') = a.low2(S)$

   $|$ $b.low(S) < v \rightarrow nextbig(S) :=$ the high endpoint of an edge n' in $b.out(S)$ with $lca(n') = b.low(S)$

   fi;

   $pair(S) := (big(S), nextbig(S))$

   rof

2. **for** each vertex v **in parallel** $\rightarrow$

   sort the tufts of v embedded inside the path containing v as an internal vertex in lexicographically nonincreasing value of their pairs;

   output the tufts in the order computed by the sort

   rof

end.

Lemma 7.5 If G_l is planar then Algorithm 7.1 finds a cyclic ordering of the tufts corresponding to a planar embedding of G_l.

Proof If the pairs sorted in step 2 of Algorithm 7.1 are distinct then by Lemma 7.4 this cyclic ordering corresponds to a planar embedding of G_l. Otherwise, let S_1 and S_2 be two tufts with $pair(S_1) = pair(S_2)$. Let $pair(S_1) = (a, b)$. If $b \neq v$ then by Lemma 7.4 S_1 must be embedded before S_2 since $a > b$; also S_1 must be embedded after S_2 since $b < a$. Hence no planar embedding is possible if S_1 and S_2 are to be embedded on the same side of P'_i (where P'_i is the path that contains v as an internal vertex). If $b = v$ then by the computation in the for loop of step 1, every nontree edge in $out(S_1)$ and $out(S_2)$ is incident on a or has lca greater than v. In this case the pair (a, v) is a separating pair for G_l and S_1 and S_2 can appear in either order in a planar embedding of G_l.[]

Lemma 7.6 Algorithm 7.1 can be implemented to run in logarithmic time with A-optimal performance.

Proof The only nontrivial computation in Algorithm 7.1 is the computation of tree functions that can be computed using the Euler tour technique, lca computation and bucket sort. Hence the algorithm runs in logarithmic time with A-optimal performance.[]

Algorithm 7.1, together with the ordering of edges within each tuft, gives the cyclic ordering of tree edges outgoing at each vertex. We number these tree edges in cyclic order as $0,-1,-2, \cdots$; let this be the *cyclic tree number* of the edge. To find the cyclic ordering of the incoming nontree edges at each vertex, we assign each tree edge (x,y) that is outgoing from x, for each vertex x, the ordered pair (x,c), where c is the cyclic tree number of the edge. For each nontree edge incoming to a vertex v, we consider the base edge of the fundamental cycle of each such nontree edge that lies on the path from the lca to the low endpoint, and we embed nontree edges incoming to v in reverse order of the ordered pairs of these base edges. It is easy to see that this gives a cyclic ordering for the nontree edges corresponding to a planar embedding of G_l consistent with the ordering obtained for the tree edges.

7.2. The Combinatorial Embedding of the Input Graph

In this section we show that we can work with G_l in order to obtain a planar embedding of G.

Lemma 7.7 G is planar if and only if G_l is planar.

Proof If G_l is planar then clearly G is planar. For the reverse, let C_i be the fundamental cycle in G of the nontree edge in ear P_i with respect to tree T_{st} and let C'_i be its image in G_l. Let $B_1, \cdots, B_k$ be the bridges of C_i in G and let $B'_1, \cdots, B'_{k'}$ be the bridges of C'_i in G_l. By the results in Section 4.1.3 in [Ra90] there is a 1-1 correspondence between the B_j and the B'_j such that an edge e in $G - C_i$ is in B_j if and only if e is in the bridge corresponding to it in $G_l - C'_i$. (The results in Section 4.1.3 of [Ra90] are for bridges of P_i; however it is straightforward to extend them to bridges of C_i.)

Let G be planar and let $\hat{G}$ be a planar embedding of G. Let $\hat{C_i}$ be the embedding of C_i. Replace each vertex v on C_i by its image in T_v, together with its parent and children (if any) in T_v. The embedding $\hat{G}$ can be extended to a planar embedding in this new graph. This can be established by virtue of the correspondence between the bridges of C_i in G and those of C'_i in G_l and by using the properties of the local replacement graph; we omit the details. We now find a fundamental cycle inside C_i (similarly outside C_i) that intersects C_i and repeat this construction. Since planarity is preserved, we can continue to repeat this construction until we have exhausted all fundamental cycles at which point we obtain a planar embedding of G_l.[]

8. The Complete Algorithm and Its Complexity

We now present the complete algorithm for obtaining a planar embedding of a graph on n vertices if one exists.

Algorithm 8.1: Planarity Algorithm

Input: A biconnected graph $G=(V,E)$.

Output: A combinatorial embedding of G if G is planar.

vertex s , t , v ;

integer i ; {{The range of this integer is from 0 to $r-1$.}}

if $|E| > 3 \cdot |V| \rightarrow$ report G is nonplanar and **halt fi**;

1. fix an edge (s,t) in the graph; find an open ear decomposition $D=[P_0, \cdots, P_{r-1}]$ starting with (s,t); construct the directed st-numbering graph G'_{st}, its spanning tree T'_{st}, and the associated paths $P'_0, P'_1, \cdots, P'_{r-1}$;
2. find the bunches of each P'_i together with the hooks for the anchor bunches;
3. construct the constraint graph G^* by forming the interlacing parity graph for each bunch graph and adding in the link edges;

   if G^* is not 2-colorable $\rightarrow$ report G is nonplanar and **halt fi**;
4. find a 2-coloring of G^*;

   for each P'_i **in parallel** $\rightarrow$

   5. assign all bunches whose corresponding vertices on G^* were given color 0 in G^* inside P'_i and the remaining bunches outside P'_i
   6. find the cyclic ordering of the edges assigned to each side of P'_i and consequently the ordering of edges around each vertex

   rof;
7. compute the number of faces in this combinatorial embedding and verify Euler's formula to determine if G is planar;
8. **if** G_l is planar $\rightarrow$ collapse all vertices in T_v into a single vertex v for each v to obtain a combinatorial embedding of G

   | G_l is nonplanar $\rightarrow$ report G is nonplanar

   fi

end.

Step 1 is described in Section 2, step 2 in Section 4, step 3 in Section 5 and step 6 in Section 7. Steps 4 and 5 have easy optimal logarithmic time parallel algorithms and steps 7 and 8 can be computed with similar bounds using the Euler tour technique [TV84, CV86, KD88]. This gives us the main theorem of the paper.

Theorem 8.1 The planarity problem can be solved on a CRCW PRAM in logarithmic time with A-optimal performance.

REFERENCES

[BL76] K. Booth, G. Lueker, "Testing for the consecutive ones property, interval graphs, and graph planarity using PQ-tree algorithms," *J. Comp. Syst. Sci.*, vol. 13, 1976, pp. 335-379.

[CV86] R. Cole, U. Vishkin, "Approximate and exact parallel techniques with applications to list, tree and graph problems," *Proc. 27th Ann. IEEE Symp. on Foundations of Comp. Sci.*, 1986, pp. 478-491.

[Ed60] J. Edmonds, "A combinatorial representation for polyhedral surfaces," *Not. Am. Math. Soc.*, vol. 7, 1960, p. 646.

[Ev79] S. Even,. *Graph Algorithms*, Computer Science Press, Potomac, MD, 1979.

[ET76] S. Even, R. Tarjan, "Computing an st-numbering," *Theoretical Computer Science*, vol. 2, 1976, pp. 339-344.

[FRT89] D. Fussell, V. Ramachandran, R. Thurimella, "Finding triconnected components by local replacements," *Proc. ICALP 89*, Springer Verlag LNCS 372, 1989, pp. 379-393.

[Ga86] H. Gazit, "An optimal randomized parallel algorithm for finding connected components in a graph," *Proc. 27th Ann. IEEE Symp. on Foundations of Comp. Sci.*, 1986, pp. 492-501.

[Ha88] T. Hagerup, "Optimal parallel algorithms on planar graphs," *Proc. 3rd Aegean Workshop on Computing*, Springer-Verlag LNCS 319, 1988, pp. 24-32.

[HT73] J. E. Hopcroft, R. E. Tarjan, "Dividing a graph into triconnected components," *SIAM J. Comput.*, vol. 2, 1973, pp. 135-158.

[HT74] J. E. Hopcroft, R. E. Tarjan, "Efficient planarity testing," *J. ACM*, vol. 21, 1974, pp. 549-568.

[JS82] J. Ja'Ja', J. Simon, "Parallel algorithms in graph theory: planarity testing," *SIAM J. Computing*, vol. 11, 1982, pp. 314-328.

[KR87] A. Kanevsky, V. Ramachandran, "Improved algorithms for graph four-connectivity," *Jour. Computer and Syst. Sci.*, to appear; preliminary version in *Proc. 28th Ann. IEEE Symp. on Foundations of Comp. Sci.*, 1987.

[KR90] R. M. Karp, V. Ramachandran, "Parallel algorithms for shared-memory machines," *Handbook of Theoretical Computer Science*, North-Holland, 1990, to appear.

[KR88] P. Klein, J.H. Reif, "An efficient parallel algorithm for planarity," *J. Comp. Syst. Sci.*, vol. 37, 1988.

[KD] S. R. Kosaraju, A. L. Delcher, "Optimal parallel evaluation of tree-structured computations by raking," *Proc. 3rd Aegean Workshop on Computing*, Springer-Verlag LNCS 319, 1988, pp. 101-110.

[Ku30] Kuratowski, "Sur le problem des courbes gauches en topologie," *Fund. Math.*, vol. 15, 1930, pp. 271-283.

[LEC67] A. Lempel, S. Even, I. Cederbaum, "An algorithm for planarity testing of graphs," *Theory of Graphs: International Symposium*, Gordon and Breach, New York, NY, 1967, pp. 215-232.

[Lo85] L. Lovasz, "Computing ears and branchings in parallel," *Proc. 26th Ann. IEEE Symp. on Foundations of Computer Science*, 1985, pp. 464-467.

[MSV86] Y. Maon, B. Schieber, U. Vishkin, "Parallel ear decomposition search (EDS) and st-numbering in graphs," *Theoretical Computer Science*, vol. 47, 1986, pp. 277-296.

[MR86] G. L. Miller, V. Ramachandran, "Efficient parallel ear decomposition with applications," manuscript, MSRI, Berkeley, CA, January 1986.

[MR87] G. L. Miller, V. Ramachandran, "A new graph triconnectivity algorithm and its parallelization," *Proc. 19th Ann. ACM Symp. on Theory of Computing*, 1987, pp. 335-344.

[MR85] G. L. Miller, J. H. Reif, "Parallel tree contraction and its applications," *Proc. 26th Ann. IEEE Symp. on Foundations of Comp. Sci.*, 1985, pp. 478-489; also, Part I, Journal of Computer and System Sciences, to appear, and Part II, Randomness and Computation, JAI Press, S. Micali, ed., (1989), to appear.

[RV88] V. Ramachandran, U. Vishkin, "Efficient parallel triconnectivity in logarithmic time," *Proc. 3rd Aegean Workshop on Computing*, Springer-Verlag LNCS 319, 1988, pp. 33-42.

[Ra90] V. Ramachandran, "Parallel open ear decomposition with applications to graph biconnectivity and triconnectivity," invited chapter for the textbook *Synthesis of Parallel Algorithms*, J. H. Reif, ed., Morgan-Kaufmann.

[RR89] V. Ramachandran, J. H. Reif, "An optimal parallel algorithm for graph planarity," *Proc. 30th Ann. IEEE Symp. on Foundations of Comp. Sci.*, 1989, pp. 282-287.

[Re84] J. H. Reif, "Symmetric Complementation," *J. ACM*, vol. 31, 1984, pp. 401-421.

[SV88] B. Schieber, U. Vishkin, "On finding lowest common ancestors: simplification and parallelization," *Proc. 3rd Aegean Workshop on Computing*, Springer-Verlag LNCS 319, 1988, pp. 111-123.

[Ta83] R. E. Tarjan, *Data Structures and Network Algorithms*, SIAM Press, Philadelphia, PA, 1983.

[TV84] R. E. Tarjan, U. Vishkin, "An efficient parallel biconnectivity algorithm," *SIAM J. Computing*, vol. 14, 1984, pp. 862-874.

[Tu63] W. T. Tutte, "How to draw a graph," *Proc. London Math. Soc.*, vol. 3, 1963, pp. 743-768.

[Tu66] W. T. Tutte, *Connectivity in Graphs*, University of Toronto Press, 1966.

[Wh73] A. T. White, *Graphs, Groups, and Surfaces*, North-Holland, Amsterdam, 1973.

[Wh30] H. Whitney, "Non-separable and planar graphs," *Trans. Amer. Math. Soc.*, vol. 34, 1930, pp. 339-362.

APPENDIX

Basic Definitions

A *graph* (or *undirected graph*) $G=(V,E)$ consists of a *vertex set* V and an *edge set* E containing unordered pairs of distinct elements from V. An edge (u,v) is *incident* on vertices u and v. Vertices u and v are *adjacent* in G if G contains edge (u,v). The *degree* of a vertex is the number of edges incident on the vertex.

A *directed graph* $G=(V,E)$ consists of a vertex set V and an edge set E containing ordered pairs of elements from V. An edge (u,v) in a directed graph is directed from u to v and is *outgoing* from u and *incoming* to v.

A *multigraph* $G=(V,E)$ is an undirected graph in which there can be several edges between the same pair of vertices, as well as *self-loop edges* of the form $(v,v), v \in V$. An edge e in a multigraph may be denoted by (a,b,i) to distinguish it from other edges between a and b; in such cases the third entry in the triplet may be omitted for one of the edges between a and b.

A *path* P in G is a sequence of vertices $<v_0, \cdots, v_k>$ such that $(v_{i-1}, v_i) \in E, i=1, \cdots, k$; P is directed or undirected depending on whether G is directed or undirected. The path P *contains* the vertices $v_0, \cdots, v_k$ and the edges $(v_0, v_1), \cdots, (v_{k-1}, v_k)$ and has *endpoints* v_0, v_k, and *internal vertices* $v_1, \cdots, v_{k-1}$. The path P is a *simple path* if $v_0, \cdots, v_{k-1}$ are distinct and $v_1, \cdots, v_k$ are distinct, and all edges on P are distinct. A simple path $P=<v_0, \cdots, v_k>$ is a *simple cycle* if $v_0=v_k$; otherwise P is *noncyclic*. The path $<v>$ is a *trivial path* with no edges. If $P=<v_0, \cdots, v_k>$ is a noncyclic path, the *innard of* P is the path $<v_1, \cdots, v_{k-1}>$, i.e., the path obtained from P by deleting the first and last vertices.

A graph $G'=(V',E')$ is a *subgraph* of a graph $G=(V,E)$ if $V' \subseteq V$ and $E' \subseteq E$. The *subgraph of* G *induced by* V' is the graph $H=(V',F)$ where $F=\{(u,v) \in E \mid u,v \in V'\}$.

An undirected graph $G=(V,E)$ is *connected* if there exists a path between every pair of vertices in V. A *connected component* of a graph G is a maximal induced subgraph of G which is connected.

Let $G=(V,E)$ and $H=(W,F)$ be a pair of graphs. The graph $G \cup H$ is the graph $G'=(V \cup W, E \cup F)$. If $W \subseteq V$ then the graph $G-H$ is the graph $H'=(V,E-F)$.

A *tree* is a connected graph containing no cycle. A *leaf* in a tree is a vertex of degree 1. Let $T=(V,E)$ be a tree and let $r \in V$. The *out-tree* T *rooted at* r (or simply the *tree* T *rooted at* r) is the directed graph obtained from T by directing each edge such that every path from r to any other vertex is directed away from r. The *in-tree rooted at* r is the directed graph obtained from T by directing each edge such that the path from every vertex to r is directed towards r.

Let (x,y) be a directed edge in a rooted tree T. Then, x is the *parent* of y and y is a *child* of x in T. Vertex v is a *descendant* of vertex u (and equivalently, u is an *ancestor* of v) if there is a directed path from u to v in T. Vertex v is a *proper descendant* of u (and u a *proper ancestor* of v) if v is a descendant of u and $u \neq v$. Given a pair of vertices $u,v \in V$, the *least common ancestor of* u *and* v, denoted by $lca(u,v)$ is the vertex $w \in V$ that is an ancestor of both u and v with no child of w being an ancestor of both u and v. For an edge $e=(u,v)$ the *least common ancestor of* e, denoted by $lca(e)$, is the vertex $lca(u,v)$.

Let $G=(V,E)$ be a connected graph. A spanning tree T of G is a subgraph of G with vertex set V such that T is a tree. An edge in $G-T$ is a *nontree edge with respect to* T.

Let T be a spanning tree of G. Any nontree edge e of G creates a cycle in the graph $T\cup\{e\}$, called the *fundamental cycle of* e *with respect to* T. Let $r\in V$, and let T be rooted at r.

Let $e=(u,v)$ be a nontree edge in $T=(V,E,r)$ and let $lca(e)=l$. The fundamental cycle of e with respect to T consists of the path from l to u, followed by edge e, followed by the path from v to l. Let (l,a) be the first edge on the path from l to u and (l,b) be the first edge on the path from l to v (it is possible for one of these edges to be missing). Then edges (l,a) and (l,b) are the *base edge(s) of the fundamental cycle of* e (when they exist) and the vertices a and b are the *base vertice(s) of the fundamental cycle of* e (when they exist).

An edge $e\in E$ in a connected graph $G=(V,E)$ is a *cutedge* if e does not lie on a cycle in G. An undirected graph $G=(V,E)$ is *2-edge connected* if it contains no cutedge. A *2-edge connected component of* G is a maximal induced subgraph of G which is 2-edge connected.

A vertex $v\in V$ is a *cutpoint* of a connected undirected graph $G=(V,E)$ if the subgraph induced by $V-\{v\}$ is not connected. G is *biconnected* (or *two-vertex connected)* if it contains no cutpoint. A *biconnected component* (or *block)* of G is a maximal induced subgraph of G which is biconnected.

By *Menger's theorem* a graph is 2-edge connected if and only if there are at least two edge-disjoint paths between every pair of distinct vertices, and a graph is biconnected if and only if there are at least two vertex-disjoint paths between every pair of distinct vertices.

Let G be a connected graph. The *blocktree* of G is a tree with a vertex for each block and each cutpoint in G, and an edge between each cutpoint and the blocks that contain it. A *rooted blocktree of* G is the blocktree of G that is rooted at one of the vertices that represent a cutpoint of G.

Finally we give some definitions on the *triconnected components* of a biconnected graph (see, e.g., [Tu66, HT73, FRT89, Ra90]).

A pair of vertices a,b in a multigraph $G=(V,E)$ is a separating pair if and only if there are two nontrivial bridges, or at least three bridges, one of which is nontrivial, of $\{a,b\}$ in G. If G has no separating pairs then G is triconnected. The pair a,b is a *nontrivial* separating pair if there are two nontrivial bridges of a,b in G.

Let $\{a,b\}$ be a separating pair for a biconnected multigraph $G=(V,E)$. For any bridge X of $\{a,b\}$, let $\overline{X}$ be the induced subgraph of G on $(V-V(X))\cup\{a,b\}$. Let B be a bridge of G such that $|E(B)|\geq 2, |E(\overline{B})|\geq 2$ and either B or $\overline{B}$ is biconnected. We can apply a *Tutte split* $s(a,b,i)$ to G by forming G_1 and G_2 from G, where G_1 is $B\cup\{(a,b,i)\}$ and G_2 is $\overline{B}\cup\{(a,b,i)\}$. The graphs G_1 and G_2 are called *split graphs of* G *with respect to* a,b. The *Tutte components* of G are obtained by successively applying a Tutte split to split graphs until no Tutte split is possible. Every Tutte component is one of three types: i) a triconnected simple graph; ii) a simple cycle (a *polygon);* or iii) a pair of vertices with at least three edges between them (a *bond);* the Tutte components of a biconnected multigraph G are the unique *triconnected components* of G.

If a pair of vertices of G appear in a triconnected component of G then by Menger's theorem there must be 3 vertex-disjoint paths in G between x and y. Conversely if there are 3 vertex-disjoint paths between x and y then there must be a triconnected component of G that contains a copy of both x and y.

Let $G=(V,E)$ be a biconnected graph with an open ear decomposition $D=[P_0, \cdots, P_{r-1}]$. A separating pair a,b in G is a *pair separating* P_i if a and b are contained in P_i and the vertices between a and b on P_i are separated from the vertices on ears numbered lower than i. A *candidate list* for P_i is a sequence of vertices on P_i in increasing order of their values such that each pair of vertices on the list is either adjacent on P_i or a pair separating P_i. It is known that every separating pair in a graph G with an open ear decomposition D is contained in a candidate list for some ear in D [MR87, Ra90].

Let a,b be a pair separating P_i. Let $B_1, \cdots, B_k$ be the bridges of P_i with no attachments outside the interval $[a,b]$ on P_i, and let $T_i(a,b)=(\bigcup_{j=1}^{k} B_j) \cup P_i(a,b)$, where $P_i(a,b)$ is the segment of P_i between and including vertices a and b. Then the *ear split* $e(a,b,i)$ consists of forming the *upper split graph* $G_1=T_i(a,b) \cup \{(a,b,i)\}$ and the *lower split graph* $G_2=\overline{T}_i(a,b) \cup \{(a,b,i)\}$. An ear split $e(a,b,i)$ is a Tutte split if either $G_1-\{(a,b,i)\}$ or $G_2-\{(a,b,i)\}$ is biconnected.

Let S be a nontrivial candidate list for ear P_i. Two vertices u,v in S are an *adjacent separating pair for* P_i if u and v are not adjacent to each other on P_i and S contains no vertex in the interval (u,v) on P_i. Two vertices a,b in S are an *extremal separating pair for* P_i if $|S| \geq 3$ and S contains no vertex in the interval outside $[a,b]$. An ear split on an adjacent or extremal separating pair is a Tutte split, and the Tutte components of G are obtained by performing an ear split on each adjacent and extremal separating pair [MR87, Ra90].

With each ear split $e(a,b,i)$ corresponding to an adjacent or extremal pair separating P_i, we can associate a unique Tutte component of G as follows [FRT89, Ra90]. Let $e(a,b,i)$ be such a split. Then by definition $T_i(a,b) \cup \{(a,b,i)\}$ is the upper split graph associated with the ear split $e(a,b,i)$. The *triconnected component of the ear split* $e(a,b,i)$, denoted by $TC(a,b,i)$, is $T_i(a,b) \cup \{(a,b,i)\}$ with the following modifications: Call a pair c,d separating an ear P_j in $T_i(a,b)$ a *maximal pair for* $T_i(a,b)$ if there is no e,f in $T_i(a,b)$ such that e,f separates some ear P_k in $T_i(a,b)$ and c and d are in $T_k(e,f)$. In $T_i(a,b) \cup \{(a,b,i)\}$ replace $T_j(c,d)$ together with all two-attachment bridges with attachments at c and d by the edge (c,d,j), for each maximal pair c,d of $T_i(a,b)$, to obtain $TC(a,b,i)$. We denote by $TC(0,0,0)$, the unique triconnected component that contains edge P_0.