A Linear-Time Algorithm for Finding an Ambitus

B. Mishra[1]

Department of Computer Science
Courant Institute of Mathematical Sciences
New York University
719 Broadway
New York, NY 10003

R.E. Tarjan[2]

Department of Computer Science
Princeton University
Princeton, NJ 08544

and

NEC Research Institute
Princeton, NJ 08540

[1]Supported in parts by National Science Foundation Grants DMS-8703458 and CCR-9002819.

[2]Research at Princeton University partially supported by DIMACS, a National Science Foundation Science and Technology Center, Grant No. NSF-STC88-09648, and by National Science Foundation Grant CCR-8929505.

Abstract

We devise a linear-time algorithm for finding an ambitus in an undirected graph. An ambitus is a cycle in a graph containing two distinguished vertices such that certain different groups of bridges (called B^P-, B^Q- and B^{PQ}-bridges) satisfy the property that a bridge in one group does not interlace with any bridge in the other groups. Thus, an ambitus allows the graph to be cut into pieces, where, in each piece, certain graph properties may be investigated independently and recursively, and then the pieces can be pasted together to yield information about these graph properties in the original graph. In order to achieve a good time-complexity for such an algorithm employing the divide-and-conquer paradigm, it is necessary to find an ambitus quickly. We also show that, using ambitus, linear-time algorithms can be devised for abiding-path-finding and nonseparating-induced-cycle-finding problems.

Contents

1 Introduction

The concept of an ambitus was first introduced in [5] and [6], in the process of devising an efficient divide-and-conquer algorithm for the all-bidirectional-edges problem. An ambitus is a cycle in a graph containing two distinguished vertices such that certain different groups of bridges (called B^P-, B^Q- and B^{PQ}-bridges) satisfy the property that a bridge in one group avoids (*i.e.*, does not interlace with) every bridge in the other groups. (See section 2, for a more formal definition of ambitus.) Thus, an ambitus allows the graph to be cut into pieces, where, in each piece, certain graph properties may be investigated independently and recursively, and then the pieces can be pasted together to yield information about these graph properties in the original graph. In order to achieve a good time complexity for such an algorithm employing the divide-and-conquer paradigm, it is necessary to find an ambitus *quickly*, *i.e.*, in $O(|E|+|V|)$ time. Such an algorithm first appears in [6] and yields a time complexity of $O(|E| \cdot |V|)$ for the algorithm to find all bidirectional edges of an undirected graph.

In many respects, this algorithm is a generalization of the planarity testing algorithm, due to Hopcroft and Tarjan [3]. Like their planarity testing algorithm, the ambitus-finding algorithm needs to decompose the graph into a set of internally vertex disjoint subpaths, although the decomposition needed for our purpose has to be somewhat different. Unlike their planarity testing algorithm, the ambitus finding algorithm cannot traverse the subpaths in an order, known *a priori*; rather the order, in which the subpaths are traversed, has to be determined dynamically. For this purpose, we have developed a novel data structure that maintains a set of (integral) intervals and supports a fast FIND-AND-UPDATE operation that detects the interval corresponding to a subpath to be visited next and updates the set of intervals, appropriately. This data structure has a good *amortized* time complexity, and may be of independent interest.

Subsequently, many other researchers have developed new algorithms either for ambitus-finding in special classes of graphs or for certain other closely related concepts in general graphs. Two such related concepts are *abiding paths* and *nonseparating induced cycles*. In section 8, we present linear-time algorithms for abiding paths in nonseparable graphs and nonseparating induced cycles in 3-connected graphs. The abiding paths have to be found in a 'divide' step for an algorithm, due to Ohtsuki [7], for the two-vertex-disjoint-paths problem. Although Ohtsuki first claimed to have a linear-time algorithm for abiding-path-finding problem, without the relevant implementation details, it is unclear if the algorithm he sketched can achieve the claimed time complexity. Nonseparating induced cycles need to be found repeatedly (at most $|V|$ times) in an algorithm, due to Cheriyan and Maheswari [1], that finds three independent spanning trees rooted at a distinguished vertex, in a 3-connected graph.

In [4], Krishnan, Pandu Rangan and Sheshadri have developed a simple linear-time algorithm to find an ambitus in a planar graph. Although their algorithm is simple and elegant, the techniques employed do not generalize to other classes of graphs. In [10], R. Sundar has developed an $O(|E| + |V| \log |V|)$ time algorithm to find an abiding path in a nonseparable graph. This algorithm can be used for the ambitus-finding problem, if the graph is suitably modified. However, such an algorithm has a linear-time behavior, only if the graph is dense. In [1], Cheriyan and Maheswari have developed a linear-time algorithm to find a nonseparating induced cycle in a 3-connected graph. With suitable modifications to certain basic steps in the algorithm, a different linear-time algorithm for the ambitus-finding problem can be devised.

The paper is organized as follows: In section 2, we define some graph theoretic terminology and introduce other key concepts required in the paper. In section 3, we demonstrate that an ambitus always exists in a nonseparable graph, and an ambitus can be found by a naïve algorithm of complexity, $O(|E| \cdot |V|)$. In section 4, we provide a sketch of the main algorithm, and in the subsequent two sections, we provide the implementation details, in order to guarantee a linear-time behavior. In the last section, we give two simple applications of the ambitus-finding algorithm.

2 Preliminaries

In this section, we define some graph theoretic terminology and introduce other key concepts required in the paper. Most important among these are the terms: *bridge*, *residual path*, *cross-cut* and *ambitus*. The definitions are similar to those used in the context of Tutte's Theorem on Hamiltonian circuits in 4-connected planar graphs given in Tutte [13], those in the planarity-testing algorithm of Hopcroft and Tarjan [3] or those in connection with the four-color problem as presented in Ore [8].

2.1 Graph Theoretic Terminology

A *graph* $G = (V, E)$ is a finite set V of *vertices* and a set E of pairs of vertices, called *edges*. Either the edges are ordered pairs $\langle u, v\rangle$ of distinct vertices (the graph is *directed*) or the edges are unordered pairs $[u, v]$ of distinct vertices (the graph is *undirected*). If $[u, v]$ is an undirected edge, u and v are *adjacent*. If $\langle u, v\rangle$ is a directed edge, u is a *predecessor* of v (respectively, v is a *successor* of u), sometimes denoted by, $u \rightarrow v$ (respectively, $v \leftarrow u$). We call u and v, the *ends* of the edge.

A graph $G_1 = (V_1, E_1)$ is a *subgraph* of G, if $V_1 \subseteq V(G)$, $E_1 \subseteq E(G)$, and each edge of G_1 has the same ends in G_1 as in G. If G_1 is a subgraph of G, other than G itself, then G_1 is a *proper* subgraph of G. If $V_1 = V(G)$ then G_1 is said to be a *spanning subgraph* of G. A *vertex of attachment* of G_1 in G is a vertex of G_1 that is incident in G with some edge not belonging to G_1.

If $E_1 \subseteq E(G)$, let $V(E_1)$ be the set of all vertices v of G such that v is incident with a member of E_1, *i.e.*,

$$V(E_1) = \Big\{v \in V(G) : \Big(\exists\, u \in V(G)\Big)\ [u, v] \in E_1\Big\}.$$

Then the subgraph $\langle E_1\rangle = (V(E_1), E_1)$ is the *reduction* of G to E_1. Similarly, if $V_1 \subseteq V(G)$, let $E(V_1)$ be the set of all edges of G having both ends in V_1,*i.e.*,

$$E(V_1) = \Big\{[u, v] \in E(G) : u, v \in V_1\Big\}.$$

Then the subgraph $\langle V_1\rangle = (V_1, E(V_1))$ is the subgraph of G *induced* by V_1.

An undirected graph is *connected* if there is a path connecting every pair of vertices and *disconnected* otherwise. The maximal connected subgraphs of G are its *connected components* or simply, *components*.

A *path* of *length* k from u to v in G is a sequence of vertices $(u =)\ u_0, u_1, \ldots, u_k\ (= v)$ such that $\langle u_i, u_{i+1}\rangle \in E$ for $0 \le i < k$. (Sometimes denoted by $u \xrightarrow{*} v$.) The path *contains* the edges $\langle u_i, u_{i+1}\rangle$ for $0 \le i < k$ as well as vertices u_i for $0 \le i \le k$. The vertices u and v are called the *ends* of the path P. All other vertices of the path (*i.e.*, u_i's for $0 < i < k$) are the *internal vertices* of the path.

If $0 \le i \le j \le k$, then the sequence of vertices, $u_i, u_{i+1}, \ldots, u_j$ is a *subpath* of the path from u to v. If P is a path from u to v, $u = u_0, u_1, \ldots, u_k = v$, and $0 \le i \le j \le k$ then the subpath from u_i to u_j, including both u_i and u_j is represented by $P[u_i; u_j]$; the subpath excluding u_i but including u_j, by $P]u_i; u_j]$; the subpath including u_i but excluding u_j, by $P[u_i; u_j[$ and the subpath excluding both u_i and u_j, by $P]u_i; u_j[$. If $P = u_0, u_1, \ldots, u_{k-1}, u_k$ is a path from u_0 to u_k, then the *reversal* of the path P is $P^{\mathrm{R}} = u_k, u_{k-1}, \ldots, u_1, u_0$. If $P_1 = u_0, u_1, \ldots, u_i$ and $P_2 = u_i, u_{i+1}, \ldots, u_k$ are two paths then the *concatenation* of P_1 and P_2 is $P_1 * P_2 = u_0, u_1, \ldots, u_i, u_{i+1}, \ldots, u_k$.

The path is *simple* if $u_0, \ldots, u_k$ are distinct (except possibly $u_0 = u_k$) and the path is a *cycle* if $u_0 = u_k$. By convention there is a path of no edges from every vertex to itself (*null path*), but a cycle must contain at least two edges. Two simple paths P_1 and P_2 are said to be *vertex disjoint*, if the vertices of P_1 and P_2 are mutually distinct; *internally vertex disjoint*, if the internal vertices of P_1 and P_2 are mutually distinct.

A connected graph is said to have a *separation vertex* v (also called an articulation point) if there exist vertices a and b, $a \ne v$ and $b \ne v$, such that all the paths connecting a and b pass through v. A graph which has a separation vertex is called *separable*, and one which has none is called *nonseparable* (also called biconnected). The maximal nonseparable subgraphs of G are its *nonseparable components* (also called biconnected components).

2.2 Bridge and Ambitus

In this section we introduces the notion of bridges for cycles in general graphs. The term 'bridge' is taken from Tutte [13], which also contains a rather complete survey of bridge theory in both general and planar graphs. The equivalent terms for bridge, in some older literature, are 'component *mod* J', 'J-component' [12] and 'Gespinst'[3] [9].

Definition 2.1 BRIDGES.[Tutte]

Let J be a fixed subgraph of G. A subgraph G_1 of G is said to be J-*detached* in G, if all its vertices of attachment are in J. We define a *bridge* of J in G as any subgraph B that satisfies the following three conditions:

1. *B is not a subgraph of J.*
2. *B is J-detached in G.*
3. *No proper subgraph of B satisfies both* (1) *and* (2).

The set of vertices of attachment of a bridge B of a subgraph J in G is denoted by $W(G, B) = \{v_0, v_1, \ldots, v_{k-1}\}$. □

Definition 2.2 DEGENERATE AND PROPER BRIDGES. NUCLEUS OF A BRIDGE.

An edge $e = [u, v]$ of G not belonging to J but having both ends in J is referred to as a *degenerate* bridge.

[3] Gespinst is the German word for 'cobweb'.

Figure 1: *Bridges of J.*

Let G^- be the graph derived from G by deleting the vertices of J and all their incident edges. Let C be any component of G^-. Let B be the subgraph of G obtained from C by adjoining to it each edge of G having one end in C and one in J, and adjoining also the ends in J of all such edges. The subgraph B satisfies the conditions to be a bridge. Such a bridge is called *proper*. The component C of G^- is the *nucleus* of B. $\square$

Remark 2.3 By a Theorem due to Tutte, if B is any bridge of a subgraph J of G then B is either degenerate or proper. Hence using the above definition of a bridge and the theorem, we can give a linear-time algorithm to find all bridges of a subgraph J of G. $\square$

Example 2.4 In the Figure 1, we give an example of bridges of a cycle J. In this example, bridges B_1, B_2, B_3 and B_6 are proper bridges, and B_4 and B_5 are degenerate bridges.

Definition 2.5 RESIDUAL PATHS.

Let the vertices of attachment of a bridge B of a cycle J in G, be $W(G,B) = \{v_0, v_1, \ldots, v_{k-1}\}$ and let $v_0, v_1, \ldots, v_{k-1}$ be their enumeration in their cyclic order on J. The vertices of attachment dissect J into k subpaths $L_0, L_1, \ldots, L_{k-1}$ such that $L_j = J[v_j; v_{j+1(\mathrm{mod}k)}]$. These subpaths are called the *residual paths* of B in J. $\square$

Definition 2.6 RELATIONS BETWEEN BRIDGES.

Let B_1 and B_2 be two distinct bridges of a cycle J of G.

• We say B_1 *avoids* B_2 if and only if one of the following two conditions is satisfied:

1. $|w(G, B_1)| \leq 1$ *or* $|w(G, B_2)| \leq 1$.
2. *All the vertices of attachment of* B_1 *are contained in a single residual path* L *of* B_2.

• If B_1 and B_2 do not avoid one another we say that they *overlap*.

• If there exist two vertices of attachment x_1 and x_2 of B_1 and two vertices of attachment y_1 and y_2 of B_2, all four distinct, such that x_1 and x_2 separate y_1 and y_2 in the cycle J, then we say that they *interlace*. $\square$

Example 2.7 In the Figure 1, the bridges B_1 and B_2 interlace, where as, the bridge B_1 avoids B_4. Also notice that the bridge B_3 avoids every other bridge.

Definition 2.8 CROSS-CUTS.

Let J be a cycle of the graph G. A path N in G avoiding J but having its two ends x and y in J is called a *cross-cut* of J between x and y. $\square$

Definition 2.9 PATHS P AND Q.

Let G be an undirected graph with two distinguished vertices s and t and let J be a cycle of the graph containing the vertices, s and t. Let the vertices of J be ordered in a clockwise cyclic order starting with the vertex s. A subpath $J[a; b] = (a =)u_0,\ u_1,\ \ldots, u_{k-1},\ u_k(= b)$ denotes the *unique* subpath of J in which u_{i-1} precedes u_i in the clockwise cyclic order (for $1 \leq i \leq k$).

The vertices, s and t dissect the cycle J into two internally vertex disjoint paths: $P[s; t]$, where $P = J[s; t]$ and its complementary subpath in J, $Q[s; t]$, where $Q^{\mathrm{R}} = J[t; s]$. Clearly P and Q are internally vertex disjoint.

The vertices of P are ordered according to the cyclic order, and vertices of Q, according to the reverse cyclic order. A vertex u of P is said to be *to the left* of a vertex v of P, if u precedes v in the cyclic order of J; and u is *strictly to the left* of v, if, in addition, u and v are distinct. On the other hand, a vertex u of Q is said to be *to the left* of a vertex v of Q, if v precedes u in the cyclic order of J; and u is *strictly to the left* of v, if, in addition, u and v are distinct. The relation '*to the right of*' is the inverse of the relation 'to the left of;' and the relation '*strictly to the right of*' is the relation 'to the right of' with additional irreflexivity property. $\square$

Definition 2.10 BRIDGES WITH RESPECT TO THE PATHS.

Let G be an undirected graph with two distinguished vertices s and t with two internally vertex disjoint paths $P[s; t]$ and $Q[s; t]$, which meet each other only in their end vertices, s and t. We consider three different classes of bridges of interest to us:

- B^{PQ}-BRIDGES: *The set of bridges with* at least one *vertex of attachment on* $P]s; t[$ *and* at least one *vertex of attachment on* $Q]s; t[$.

Figure 2: B^{PQ}-, B^{P}- *and* B^{Q}*-bridges of* P *and* Q.

- B^{P}-BRIDGES: *The set of bridges with* at least one *vertex of attachment on* $P]s;t[$ *and* no *vertex of attachment on* $Q]s;t[$.
- B^{Q}-BRIDGES: *The set of bridges with* no *vertex of attachment on* $P]s;t[$ *and* at least one *vertex of attachment on* $Q]s;t[$.

If a bridge B of $J = P \cup Q$ in G is not a B^{PQ}-, B^{P}- or B^{Q}-bridge then it has only s or t as vertices of attachment. □

Example 2.11 In the figure 2, we show B^{PQ}-, B^{P}- and B^{Q}- bridges of the paths P and Q. Bridges B_1, B_2 and B_3 are B^{PQ}-bridges; B_4 is a B^{P}-bridge and B_5, a B^{Q}-bridge.

Definition 2.12 AMBITUS.

Let J, P and Q be as in the previous definition. Then J is called an *ambitus* if every B^{P}- or B^{Q}-bridge avoids every B^{PQ}-bridge. □

Problem 2.1 AMBITUS-FINDING PROBLEM.

Assume that $G = (V, E)$ is a nonseparable graph containing two distinguished vertices s and t. Find two internally vertex disjoint paths $P[s;t]$ and $Q[s;t]$ in G such that the cycle $J = P \cup Q$ is an ambitus. □

Without loss of generality, we assume that initially the graph G is presented to us with two internally vertex disjoint paths connecting s and t, and that the bridges of the cycle composed of these paths in G are B^P-, B^Q- or B^{PQ}-bridges. That is, we assume that the cycle does not have a bridge with all its vertices of attachment only at s or t; such bridges, if they occur, may simply be eliminated without any effect on the solution.

Notation 2.13 Let G, s, t, $P[s;t]$, $Q[s;t]$ and J be as before. If B is a bridge of the cycle J with at least one vertex of attachment on $P]s;t[$, then the left- and the right-most vertices of attachment of B on $P[s;t]$ are referred to by $s_P(B)$ and $t_P(B)$. Similarly, if B is a bridge of the cycle J with at least one vertex of attachment on $Q]s;t[$, then the left- and the right-most vertices of attachment of B on $Q[s;t]$ are referred to by $s_Q(B)$ and $t_Q(B)$. □

3 Existence of an Ambitus

First, we need few definitions.

Definition 3.1 CARRIER OF A B^P- OR A B^Q-BRIDGE.

Let B be a B^P-bridge of $J = P \cup Q$ in G. Let $s_P(B)$ and $t_P(B)$ be its left- and right-most vertices of attachment on P. Then we say $P[s_P(B);t_P(B)]$ is the *P-carrier* (or simply its *carrier*, when B is understood to be a B^P-bridge, from the context). A Q-carrier of a B^Q-bridge is defined in an identical manner. □

Definition 3.2 THE COVERING RELATION.

A B^P-bridge B_1 *covers* another B^P-bridge B_2, if following two conditions are satisfied:

1. *The P-carrier of B_2, $P[s_P(B_2);t_P(B_2)]$, is a subpath of the P-carrier of B_1, $P[s_P(B_1);t_P(B_1)]$.*
2. *B_2 has a vertex of attachment on $P]s_P(B_1);t_P(B_1)[$.*

A B^P-bridge B_1 *covers* a vertex v, if $v \in V(P]s_P(B_1);t_P(B_1)[)$ and *maximally covers*, if, in addition, no B^P-bridge B_2 *covers* B_1. A similar set of definitions holds for B^Q-bridges and the vertices of $Q]s;t[$. □

Definition 3.3 AN OFFENSIVE B^P-BRIDGE.

Let $v \in V(P]s;t[)$ be a vertex of attachment of a B^{PQ}-bridge, B, of $J = P \cup Q$ in G. If a B^P-bridge covers v then it is said to be *offensive* to B. □

Now, we are ready to prove the existence of an ambitus; the proof is constructive and readily provides an $O(|E| \cdot |V|)$ time algorithm.

Theorem 3.1 *Let G be an undirected graph with two distinguished vertices s and t such that it has two internally vertex disjoint paths $P_0[s;t]$ and $Q_0[s;t]$. Then G also has two internally vertex disjoint paths $P[s;t]$ and $Q[s;t]$ such that the cycle $J = P \cup Q$ is an ambitus.*

PROOF.

Let $J_0 = P_0 \cup Q_0$ be the cycle in the graph G. We describe a sequence of cycles, J_0, J_1, ..., J_n, where J_n is an ambitus and $n \leq |V|$. If J_i is not an ambitus then J_{i+1} is obtained first, by finding an offensive B^P- or B^Q-bridge (say, a B^P-bridge B) of J_i, and then by replacing the carrier $P[s_P(B); t_P(B)]$ of B by a cross-cut $R[s_P(B); t_P(B)]$ of B.

Since each such modification adds at least one vertex (a vertex of attachment of a B^{PQ}-bridge) to the nuclei of the B^{PQ}-bridges, the sequence of modifications must terminate in $n \leq |V|$ steps. Since each modification step involves $O(|E|)$ amount of work, the proof suggests an $O(|E| \cdot |V|)$ time algorithm. □

4 Sketch of the Algorithm

For the sake of exposition, we sketch the algorithm FIND-AMBITUS without the implementation details. The algorithm, implemented in a straightforward manner, does not produce a linear-time algorithm. The implementation details, required to achieve the linear time complexity, are discussed in the two subsequent sections.

The ambitus-finding algorithm, we devise, is loosely based on the ideas sketched in the proof of Theorem 3.1. First, we observe that the algorithm may proceed in two phases: In one phase, we deal only with the offensive B^P-bridges and modify the path P until the final modified path has no B^P-bridge left; in the next phase, we deal only with the offensive B^Q-bridges, in an identical manner. As modification of one path does not interfere with the structure of bridges with respect to the other path, in principle, the original algorithm remains unchanged. Thus, we may simply concentrate on the modifications carried out on either of the paths, say P. Let $\mathcal{V}^P$ denote the vertices of attachment of the B^{PQ}-bridges on $P]s;t[$. In order to modify the path P, the algorithm needs to select an offensive B^P-bridge. For this purpose, we select a B^P-bridge (say, B), *maximally covering* some vertex in $\mathcal{V}^P$. The last idea is a judicious selection of a cross-cut $R[s_P(B); t_P(B)]$ of B, which is achieved in the following manner: Let $P' = P[s_P(B); t_P(B)]$ be the carrier of B, and let $G_B = \langle E(B) \cup E(P') \rangle$, i.e. the subgraph induced by the edges of the bridge and the edges of its carrier. If the cycle $J' = P' \cup Q'$ is an ambitus of G_B (which can be computed by a recursive application of the algorithm) then we choose Q' as the required cross-cut R. The algorithm then modifies the path P by replacing the carrier $P[s_P(B); t_P(B)]$ by the cross-cut $R[s_P(B); t_P(B)]$. Note that, in this process, we have not changed the logical structure of the original algorithm. Of course, some of the original B^P-bridges (i.e. the ones interlacing with or covered by B) will merge with some of the original B^{PQ}-bridges; the algorithm marks these B^P-bridges, as they need not be considered any further, and adds new vertices of attachment of B^{PQ}-bridges, thus created, to the set $\mathcal{V}^P$. This operation is called a FIND-AND-UPDATE operation. (See Figure 3.)

The advantage of selecting a maximally-covering B^P-bridge and its cross-cut, as explained, is that the modified path has a simple structure, with respect to its set of B^P-bridges. In particular, each of its B^P-bridges has all its vertices of attachment only on $P[s; t_P(B)]$, only on $R[s_P(B); t_P(B)]$ or only on $P[s_P(B); t]$, and further, no B^P-bridge with its vertices of attachment on $R[s_P(B); t_P(B)]$ could be offensive. This allows one to simplify the data structure, supporting the FIND-AND-UPDATE operation, as well as to eliminate the recursion. The efficient implementation of these basic steps are based on the following two ideas:

1. PATH-FINDING: The first idea involves a modification to the depth first search and helps us to quickly find the required paths in a B^P-bridge.
2. AUXILIARY DATA STRUCTURE: The second idea involves an auxiliary data structure with the operation FIND-AND-UPDATE, which, in effect, determines the order in which B^P-bridges are to be examined.

4.1 The Algorithm

A high-level description of the algorithm can be presented in terms of three mutually recursive algorithms: FIND-AMBITUS, ANALYZE-BRIDGES and ANALYZE-BRIDGE. The correctness of the algorithm is proven in the next subsection.

Algorithm FIND-AMBITUS(G, s, t):
begin

Let $\mathcal{B}^P$ be the set of B^P-bridges of J in G,and $\mathcal{V}^P$,
the set of vertices of attachment of the
B^{PQ}-bridges of J in G, on $P]s; t[$;
$P'[s; t] :=$ ANALYZE-BRIDGES$(P[s; t], \mathcal{B}^P, \mathcal{V}^P)$;

Let J' be $P' \cup Q$;

Let $\mathcal{B}^Q$ be the set of B^Q-bridges of J' in G, and $\mathcal{V}^Q$,
the set of vertices of attachment of the
B^{PQ}-bridges of J' in G, on $Q]s; t[$;
$Q'[s; t] :=$ ANALYZE-BRIDGES$(Q[s; t], \mathcal{B}^Q, \mathcal{V}^Q)$;

return $J'' = P' \cup Q'$;
end{**FIND-AMBITUS**} $\square$

```
Algorithm ANALYZE-BRIDGES(L[s;t], 𝓑^L, 𝓥^L):
begin
    L'[s;t] := L[s;t];
    𝓥' := 𝓥^L;
    Unmark bridges of 𝓑^L;

    MainLoop:
    until 𝓥' = ∅ loop

        Find-and-Update:
        Pair := ⟨v, B⟩
            (* v ∈ 𝓥' and B = an unmarked bridge, maximally covering v,
             * if such a bridge exists; B = ⊥, otherwise. *)
        𝓥' := 𝓥' \ {v};

        if B ≠ ⊥ then
            Let 𝓑' ⊆ 𝓑^L be the set of bridges, whose carriers intersect
                with the subpath L]s_L(B); t_L(B)[;
            for every B' ∈ 𝓑' loop
                (* Let W(B') = vertices of attachment of B'. *)
                if B' is unmarked then
                    if B' ≠ B cand
                        B' has a vertex of attachment on L]s_L(B); t_L(B)[
                    then
                        𝓥' := 𝓥' ∪ W(B');
                    end{if }
                    Mark B';
                end{if };
            end{loop };
        end{if };
        end{Find-and-Update};

        if B ≠ ⊥ then
            Let L'' = L[s_L(B); t_L(B)] be the carrier of B;
            Let G_B = ⟨E(B) ∪ E(L'')⟩;
            R[s_L(B); t_L(B)] := ANALYZE-BRIDGE(G_B, s_L(B), t_L(B), L'');
            L'[s;t] := L'[s; s_L(B)] * R[s_L(B); t_L(B)] * L'[t_L(B); t];
        end{if };
    end{MainLoop};

    return L'[s;t];
end{ANALYZE-BRIDGES} □
```

```
Algorithm ANALYZE-BRIDGE(G_B, s, t, Q[s;t]):
begin
      (* G_B is composed of the bridge, B, and its carrier, Q[s;t] *)

      Find a path P[s;t] from s to t in B;
      (* P[s;t] and Q[s;t] are two internally vertex disjoint paths
       * in G_B and J = P ∪ Q is a cycle in G_B. *)

      Let 𝓑^P be the set of B^P-bridges of J in G_B, and 𝓥^P,
            the set of vertices of attachment of the B^{PQ}-bridges of J
            in G_B, on P]s;t[;

      R[s;t] := ANALYZE-BRIDGES(P[s;t], 𝓑^P, 𝓥^P);

      return R[s;t];
end{ANALYZE-BRIDGE} □
```

4.2 The Correctness of the Algorithm Find-Ambitus

For the proof of correctness of the algorithms, we need to define the following notations:

- A sequence of paths: $P_0[s;t]$, $P_1[s;t]$, ..., $P_i[s;t]$, ..., where $P_0[s;t] = P[s;t]$ and $P_i[s;t]$ is the modified path obtained at the end of the i^{th} iteration of the MAINLOOP of the algorithm ANALYZE-BRIDGES.
- A sequence of sets of vertices: $\mathcal{V}_0$, $\mathcal{V}_1$, ..., $\mathcal{V}_i$, ..., where $\mathcal{V}_0 = \mathcal{V}^P$ and $\mathcal{V}_i$ is the modified $\mathcal{V}'$ at the end of the i^{th} iteration of the MAINLOOP of the algorithm ANALYZE-BRIDGES.

Definition 4.1 A path $P_i[s;t]$ is *well defined* with respect to the cycle $J = P \cup Q$, if

1. *It is simple and internally vertex disjoint with* $Q[s;t]$.
2. *There is a sequence of vertices:* v_1, v_1', v_2, v_2', ..., v_p, v_p' *on* $P]s;t[$ *arranged in a left-to-right order such that* $P_i[s;t]$ *can be written as the concatenation of a sequence of alternating common-sections and cross-cuts as follows:*

$$P[s;v_1] * R[v_1;v_1'] * \cdots * R[v_p;v_p'] * P[v_p';t]. \quad \square$$

Let

$$\mathcal{W}_i = V(P) \cap V(P_i) = V(P[s;v_1]) \cup V(P[v_1';v_2]) \cup \cdots \cup V(P[v_p';t]),$$

and

$$\overline{\mathcal{W}_i} = V(P) \setminus V(P_i) = V(P]v_1;v_1'[) \cup V(P]v_2;v_2'[) \cup \cdots \cup V(P]v_p;v_p'[).$$

Let P_i be well defined, with the sequence of vertices v_1, v_1', v_2, v_2', ..., v_p, v_p' dissecting it into internally vertex disjoint subpaths as in the Definition 4.1. Thus $J_i = P_i \cup Q_i$ is a cycle in G. Let $\mathcal{B}_i$, $\mathcal{N}_i$ and $\mathcal{A}_i$ stand, respectively, for the set of B^{PQ}-bridges of J_i in G, the vertices of their nuclei and their vertices of attachment on $P_i]s;t[$. Also, we say that the B^P-bridges of J_i are *partitioned*, if every B^P-bridge of J_i in G has all its vertices of attachment on one of the subpaths $P_i[s;v_1]$, $P_i[v_1;v_1']$, $P_i[v_1';v_2]$, ..., $P_i[v_p;v_p']$ or $P_i[v_p';t]$.

Lemma 4.1 (Main Technical Lemma)
For all $i \geq 0$,

1. $P_i[s;t]$ *is well defined with respect to* J.
2. (a) $\overline{\mathcal{W}_i} \subseteq \mathcal{N}_i$.
   (b) $\mathcal{V}_i \cap \mathcal{W}_i \subseteq \mathcal{A}_i$.
3. (a) *The* B^P*-bridges of* J_i *in* G *are partitioned.*
   (b) *If there is an offensive* B^P*-bridge* B' *of* J_i *in* G*, then* B' *is also a* B^P*-bridge of* J*. Furthermore,* B' *is unmarked and covers a vertex* $w \in \mathcal{V}_i \cap \mathcal{W}_i$ *of* J.

PROOF.
The proof is by a *double induction* on i and the size of the graph. The basis step follows trivially, since $P_0[s;t] = P[s;t]$, $\mathcal{W}_0 = V(P[s;t])$, $\mathcal{V}_0 = \mathcal{A}_0$, and since every B^P-bridge of J is initially unmarked.

Induction Step: Let $\langle v, B\rangle$ be the *Pair* chosen in the i^{th} iteration of MAINLOOP. Hence $v \in \mathcal{V}_{i-1}$. The case, when $B = \perp$, is immediate, since $P_i = P_{i-1}$, $J_i = J_{i-1}$, and

$$\mathcal{V}_i \cap \mathcal{W}_i = \Big(\mathcal{V}_{i-1} \cap \mathcal{W}_{i-1}\Big) \setminus \{v\} \subseteq \mathcal{V}_{i-1} \cap \mathcal{W}_{i-1} \subseteq \mathcal{A}_{i-1} = \mathcal{A}_i.$$

Hence, assume that

> $B \in \mathcal{B}^P$ *is an unmarked bridge, maximally covering* v. Since every bridge covering $w \in \overline{\mathcal{W}_{i-1}}$ is marked, $v \in \mathcal{W}_{i-1}$.

The invariant (1) follows from the facts that P_{i-1} is well defined (by inductive hypothesis) and that the cross-cut

$$R[s_P(B); t_P(B)] = P_i[s_P(B); t_P(B)]$$

is internally vertex disjoint with J_{i-1} and the vertices $s_P(B)$ and $t_P(B)$ lie on a common-section, $P_{i-1}[v'_k; v_{k+1}] = P[v'_k; v_{k+1}]$.

The invariant (2) holds as a result of the followings: $\mathcal{W}_i = \mathcal{W}_{i-1} \setminus V(P]s_P(B); t_P(B)[)$. Since $V(P]s_P(B); t_P(B)[) \subseteq \mathcal{N}_i$, by the inductive hypothesis, $\overline{\mathcal{W}_i} \subseteq \mathcal{N}_i$. Also, for all $x \in \mathcal{W}_i$, if $x \in \mathcal{V}_i$ then $x \in \mathcal{A}_i$. If $x \in \mathcal{V}_i \cap \mathcal{V}_{i-1}$ then by inductive hypothesis $x \in \mathcal{A}_i$. Otherwise, $x \in \mathcal{V}_i \setminus \mathcal{V}_{i-1}$: then x is a vertex of attachment of a B^P-bridge, B', of J in G, where B' has a vertex of attachment, y on $P]s_P(B); t_P(B)[$. But since $y \in \mathcal{N}_i$, $x \in \mathcal{A}_i$.

The invariant (3a) holds by virtue of the inductive hypothesis and the assumption that B is a maximal B^P-bridge covering v, since if B' is a B^P-bridge of J_i in G with a vertex of attachment on $P_i]s_P(B); t_P(B)[$ then all its vertices of attachment lie on $P_i[s_P(B); t_P(B)]$.

The last invariant (3b) can be proven as follows: Let B' be an offensive B^P-bridge of J_i in G, with all its vertices of attachment on $P_i[s_P(B); t_P(B)]$. As a result of the recursive nature of the algorithm, and by the inductive hypothesis, it follows that B' has $s_P(B)$ and $t_P(B)$ as only its vertices of attachment. But, this causes a violation of the assumption that B is a bridge maximally covering v. Thus, we see that every offensive B^P-bridge of J_i, B', has its vertices of attachment on some common-section or cross-cut of $P_i[s;t] \setminus P_i[s_P(B); t_P(B)]$. But then B' is also an offensive B^P-bridge of J_{i-1}, and the rest follows from the inductive hypothesis. □

Figure 3: *The modifications performed by* ANALYZE-BRIDGES *with the Pair* $= \langle v, B\rangle$.

The correctness of the algorithm follows as an immediate corollary of the preceding lemma.

Corollary 4.2 *The paths $P'[s;t]$ and $Q'[s;t]$, returned by the algorithm* FIND-AMBITUS, *are simple and internally vertex disjoint, and $J'' = P' \cup Q'$ is an ambitus.*

PROOF.

It suffices to observe that if G is an undirected graph, with two internally vertex disjoint paths $P[s;t]$ and $Q[s;t]$, $\mathcal{B}^P$ = the set of its B^P-bridges and $\mathcal{V}^P$ = the set of vertices of attachment of its B^{PQ}-bridges on $P]s;t[$ then $P'[s;t]$, the path returned by the algorithm ANALYZE-BRIDGES$(P[s;t], \mathcal{B}^P, \mathcal{V}^P)$, satisfies the following conditions:

1. $P'[s;t]$ is a simple path internally vertex disjoint with $Q[s;t]$;

2. The number of offensive B^P-bridges of the cycle $P' \cup Q$ is 0;

3. The number of offensive B^Q-bridges of the cycle $P' \cup Q$ is equal to the number of offensive B^Q-bridges of the cycle $P \cup Q$. $\square$

5 Implementation: Path-Finding

In this section, we present an algorithm to obtain a representation of the graph G that allows efficient implementation of many of the basic steps of the algorithm. This section is essentially Hopcroft and Tarjan's Algorithm [2] for planarity-testing, with appropriate modifications. The representation consists of a palm tree $\mathcal{P}$ of the graph G, with its tree edges and back edges appropriately directed together with a partition of the edges of the graph into a set of internally vertex disjoint simple paths.

5.1 Building the Palm Tree

Let $G = (E, V)$ be a nonseparable graph, with two distinguished vertices s and t. Let

$$\begin{aligned} P[s;t] &= v_p(=s), v_{p-1}, \ldots, v_1, v_0(=t), \\ \text{and} \qquad & \\ Q[s;t] &= w_0(=s), w_1, \ldots, w_{q-1}, w_q(=t). \end{aligned}$$

be two internally vertex disjoint paths in G that meet each other only in their end vertices.

Let $\widehat{\Phi}: E(G) \to \mathbf{N} \times \mathbf{N}$ be a function defined on the edges $[u, v]$ of G as follows:

$$\widehat{\Phi}([u,v]) = \begin{cases} \langle 0, 0 \rangle, & \text{if } [u,v] = [w_0, w_1]; \\ \langle 0, |V|+1 \rangle, & \text{if } [u,v] \in E(J) \setminus \{[w_0, w_1]\}; \\ \langle |V|+1, |V|+1 \rangle, & \text{if } [u,v] \in E(G) \setminus E(J). \end{cases}$$

Assume that the graph G is represented by adjacency lists $A(v)$, ordered according to increasing values of $\widehat{\Phi}$, under a lexicographic ordering. Since $\widehat{\Phi}([u,v])$, for each edge in G, can be calculated in $O(|E|)$ time, and the adjacency lists can be ordered in $O(|E|)$ time, using a radix sort, such a representation can be obtained in $O(|E|)$ time.

Next, we systematically explore the graph G using a depth first search (DFS), starting at s. The DFS directs the edges of G, and partitions the directed edges into two classes: *tree edges* and *back edges*, such that

1. The tree edges form a spanning tree T of G;

2. Each back edge connects a vertex with one of its ancestors in T.

We call the resulting directed graph a palm tree $\mathcal{P}$[11]. The DFS also numbers the vertices by their DFS-number in the range $(1..|V|)$; henceforth, we identify the vertices by their DFS-number.

If v is a vertex, then let S_v stand for the set of vertices reachable from a descendant of v by a single back edge. Let

$$\begin{aligned} \mathrm{LOW1}(v) &= \mathrm{MIN}(\{v\} \cup S_v), \\ \text{and} \qquad & \\ \mathrm{LOW2}(v) &= \mathrm{MIN}(\{v\} \cup (S_v \setminus \{\mathrm{LOW1}(v)\})). \end{aligned}$$

Figure 4: *The Graph G.*

LOW1(v) is the (*first*) *lowest* vertex below v reachable from a descendant of v by a single back edge, and LOW2(v) is the *second lowest* vertex below v reachable from a descendant of v by a single back edge. By convention, LOW values of v are equal to v, if they are not defined.

The LOW values of a vertex v depend only on the LOW values of children of v and on the back edges leaving v; thus it is easy to calculate LOW values by appropriately modifying DFS algorithm. (see Path-finder algorithm in Hopcroft and Tarjan [2, 3]).

We make the following observations:

1. The ordering induced by $\widehat{\Phi}$ ensures that the DFS visits the vertices of P and Q before it enters any of the bridges of J, and that the edge $[v_{p-1}, v_p]$ is a back-edge and all other edges of J are tree-edges.

2. After the DFS each of the edges is directed; the tree edges are directed from a smaller vertex to a larger vertex and the back edges from a larger vertex to a smaller vertex. Thus each edge appears once in the adjacency lists of $\mathcal{P}$. If $u \to v$ is an edge of $\mathcal{P}$ then by convention, $\widehat{\Phi}(u \to v) = \widehat{\Phi}([u, v])$.

In the next step, we use a *second* depth first search, carried out in a special order, to divide the graph into a set of simple paths which may be assembled in order to build the ambitus. To generate paths, we sort the adjacency lists of $\mathcal{P}$ according to the LOW values. For this purpose, we define a function $\Phi: E(G) \to \mathbf{N} \times \mathbf{N}$, on the edges $u \to v$ of $\mathcal{P}$ as follows:

$$\Phi'(u \to v) = \begin{cases} \langle v, u \rangle, & \text{if } u \to v \text{ is a back edge;} \\ \langle \text{LOW1}(v), u \rangle, & \text{if } u \to v \text{ is a tree edge and } \text{LOW2}(v) \geq u; \\ \langle \text{LOW1}(v), \text{LOW2}(v) \rangle, & \text{if } u \to v \text{ is a tree edge and } \text{LOW2}(v) < u. \end{cases}$$

and

$$\Phi(u \to v) = \text{MIN}\left(\Phi'(u \to v), \widehat{\Phi}(u \to v)\right),$$

where MIN is taken with respect to the lexicographic ordering.

We calculate $\Phi(u \to v)$ for each edge $u \to v$ of $\mathcal{P}$ and order the adjacency lists according to increasing values of Φ (under a lexicographic ordering), using a radix sort to achieve an $O(|E| + |V|)$ time bound.

Now we generate paths by applying depth first search to $\mathcal{P}$, using the new adjacency lists. Each time we traverse a tree edge we add it to the path being built. Each time we traverse a back edge, the back edge becomes the last edge of the current path. The next stage starts a new path. Thus each path consists of a sequence of tree edges followed by a back edge. Since each path is completely built before the next stage with a new path is started, the order in which the paths are generated induces a linear order among the paths. Furthermore, the edges $E(G)$ of the graph, G, are partitioned into a set of internally vertex disjoint simple paths; this partition of $E(G)$ is represented by associating with each edge $u \to v$ of $\mathcal{P}$, PATH$(u \to v)$, the unique path containing the edge. This algorithm, called PATH-FINDER, is just a depth first search with a few additional operations to construct paths, and can be implemented in $O(|V| + |E|)$ time. Figure 5 shows the paths obtained by the PATH-FINDER algorithm when applied to the graph G of Figure 4.

We mention a few interesting properties about the paths found by the PATH-FINDER algorithm; the proofs can be readily obtained with simple modifications of the proofs in Hopcroft and Tarjan [2].

Lemma 5.1

(A) Let L be the first path found by the algorithm PATH-FINDER. *Then $L = J = P \cup Q$.*

(B) Let $L[f; l]$ be a path found by the algorithm PATH-FINDER. *If L is not the initial path then*

1. *L is a simple path.*

2. *L contains exactly two vertices (f and l) in common with previously generated paths.*

3. *If we consider the back edges not yet used when the first edge of L is traversed, then l is the lowest vertex reachable via such a back edge from any descendant of f.*

4. *If $v \in L]f; l[$ then l is the lowest vertex reachable from any descendant of v via any back edge.* □

Definition 5.1 SEGMENTS. Let J be the cycle in G. When J is removed, G falls into several connected pieces, called *segments* of J in G. Each segment, S, consists either of a single back edge $e = u \to v$, or of a tree edge $e = u \to v$ plus a subtree with root at v plus all back edges leading from the subtree; here, $u \in V(J)$. We say that the segment, S, is associated with the edge e.

Figure 5: *The paths obtained from the palm tree of* G.

Let S' be a segment associated with an edge $e' = u' \to v'$. Let $L = \text{PATH}(u' \to v')$. If S' consists of the single back edge e' then the set of *segments* of L in S' is empty. Otherwise, S' consists of the tree edge e' plus a subtree with root v' plus all back edges leading from the subtree. When L is removed, S' falls into several connected pieces, called the *segments* of L in S'. Each segment, S, consists either of a single back edge $e = u \to v$, or of a tree edge $e = u \to v$ plus a subtree with root at v plus all back edges leading from the subtree; here, $u \in V(L)$. We say that the segment, S, is associated with the edge e.

Note that the definition of a segment is recursive, and is based on the structure of the palm tree. □

5.2 Using the Palm Tree

Next we describe how the palm tree can be used to find paths and bridges of the graphs quickly, in a recursive manner. Also, we classify B^P-bridges into: (i) *Normal* B^P*-bridges* and (ii) *Special*

Figure 6: *The paths found in the Basis Step.*

B^P-bridges, and use this classification in the recursive application of the algorithm, since the bridges of different classes have to be treated somewhat differently.

(1) The Basis Step. (Figure 6.)
By Lemma 5.1 (A), the first path found by the PATH-FINDER is $\text{PATH}(s \to w_1) = J$, the cycle consisting of the tree edges $s \to w_1 \to \cdots \to w_{q-1} \to t \to v_1 \to \cdots \to v_{p-1}$ and the back edge $v_{p-1} \to s$.

Notice that every bridge of J in G is a segment of J in G. Let u be a vertex on $P]s;t[$, and $e = u \to v$ be an edge, not belonging to P. Let B be the bridge of J in G, equal to the segment associated with e.

If e is a back edge and $s < v < t$ then B is a degenerate B^{PQ}-bridge. If e is a tree edge and $s < \text{LOW1}(v) < t$, or $s = \text{LOW1}(v) < \text{LOW2}(v) < t$ then B is a proper B^{PQ}-bridge. Otherwise, B is a B^P-bridge. If a B^P-bridge B has a vertex of attachment at s then it is said to be a *Special B^P-bridge*; otherwise, a *Normal B^P-bridge.*

If B is B^P-bridge then we can also obtain some additional informations about the bridge based on the LOW values of v.

- Let e be a back edge. Then B is degenerate.

If $v \geq t$ then the left- and right-most vertices of attachment of B are u and v, respectively. All the edges of the carrier of B are tree edges, and B is a *Normal B^P-bridge.*

If $v = s$ then the left- and right-most vertices of attachment of B are s and u, respectively. All but the last edge of the carrier of B are tree edges, B is a *Special B^P-bridge.*

Figure 7: *The paths found in the Induction Step: Normal B^P-bridge.*

- Let e be a tree edge. Then B is proper.

If $\text{LOW1}(v) \geq t$ then the left- and right-most vertices of attachment of B are u and $\text{LOW1}(v)$, respectively. All edges of the carrier of B are tree edges, and B is a *Normal B^P-bridge*.

If $\text{LOW1}(v) = s$ then the left- and right-most vertices of attachment of B are s and $\text{MIN}(u, \text{LOW2}(v))$, respectively. All but the last edge of the carrier of B are tree edges, and B is a *Special B^P-bridge*.

If $\text{LOW2}(v) \geq u$ then B has exactly two vertices of attachment; otherwise, B has three or more vertices of attachment.

From the above observations, it is easy to see that the sets $\mathcal{B}^P$ and $\mathcal{V}^P$ can be computed in linear time. In order to compute the $\mathcal{V}^P$, we simply traverse the edges of the B^{PQ}-bridges.

(2a) The Induction Step: Normal B^P-bridge. (Figure 7.)
Let B' be a normal B^P-bridge, with its left- and right-most vertices of attachment at s' and t', respectively. Let vertices of attachment of B' be x'_0, x'_1, …, x'_n such that $x'_0 < x'_1 < \cdots < x'_n$, where $n \geq 1$.

The carrier of B' can be written as $Q'[x'_0; x'_n] = x'_0 \rightarrow w'_1 \rightarrow \cdots \rightarrow w'_{q-1} \rightarrow x'_n$, where all the edges of Q' are tree edges, and $x'_0 < w'_1 < \cdots < w'_{q-1} < x'_n$. Let $G_{B'} = \langle E(B') \cup E(Q') \rangle$. In this case, either $s' = x'_0$ and $t' = x'_n$, or $s' = x'_n$ and $t' = x'_0$.
- If B' is degenerate then B' is a back edge $x'_1 \rightarrow x'_0$. Let $P'[x'_1; x'_0] = \text{PATH}(x'_1 \rightarrow x'_0) = x'_1 \rightarrow x'_0$.
- If B' is proper then B' consists of the tree edge $x'_n \rightarrow v'_1$ plus a subtree with root v'_1 plus all back edges leading from the subtree. $\text{LOW1}(v'_1) = x'_0$. Let $P'[x'_n; x'_0] = \text{PATH}(x'_n \rightarrow v'_1) = x'_n \rightarrow v'_1 \rightarrow \cdots \rightarrow v'_{p-1} \rightarrow x'_0$, where all but the last edge $v'_{p-1} \rightarrow x'_0$ are tree edges, and $x'_n < v'_1 < \cdots < v'_{p-1}$. By Lemma 5.1(B.4) $\text{LOW1}(v'_1) = \cdots = \text{LOW1}(v'_{p-1}) = x'_0$.

$J' = Q' \cup P'$ is a cycle in $G_{B'}$, consisting of the tree edges $x'_0 \rightarrow w'_1 \rightarrow \cdots \rightarrow w'_{q-1} \rightarrow x'_n \rightarrow v'_1 \rightarrow \cdots \rightarrow v'_{p-1}$ and a back edge $v'_{p-1} \rightarrow x'_0$. Hence the analysis of paths and bridges of $G_{B'}$ can be done in a manner identical to the *basis case*.

(2b) The Induction Step: Special B^P-bridge.
Let B' be a special B^P-bridge, with its left- and right-most vertices of attachment at s' and t',

Figure 8: *The paths found in the Induction Step: Special B^P-bridge with two vertices of attachment.*

respectively.

Case 1:(Figure 8.)
First, assume that B' *has exactly two vertices of attachment*: x'_0 and x'_1 such that $x'_0 < x'_1$.

The carrier of B' can be written as $Q'[x'_1; x'_0] = x'_1 \to w'_1 \to \cdots \to w'_{q-1} \to x'_0$, where all but the last edge $w'_{q-1} \to x'_0$ are tree edges, and $x'_0 < x'_1 < w'_1 < \cdots < w'_{q-1}$. Let $G_{B'} = \langle E(B') \cup E(Q') \rangle$.
• If B' is degenerate then B' is a back edge $x'_1 \to x'_0$. Let $P'[x'_1; x'_0] = \text{PATH}(x'_1 \to x'_0) = x'_1 \to x'_0$.
• If B' is proper then B' consists of the tree edges $x'_1 \to v'_1$ plus a subtree with root v'_1 plus all back edges leading from the subtree. $\text{LOW1}(v'_1) = x'_0$. Let $P'[x'_1; x'_0] = \text{PATH}\ (x'_1 \to v'_1) = x'_1 \to v'_1 \to \cdots \to v'_{p-1} \to x'_0$, where all but the last edge $v'_{p-1} \to x'_0$ are tree edges, and $x'_1 < v'_1 < \cdots < v'_{p-1}$. By Lemma 5.1(B.4) $\text{LOW1}(v'_1) = \ldots = \text{LOW1}(v'_{p-1}) = x'_0$.

$J' = Q' \cup P'$ is a cycle in $G_{B'}$, consisting of the tree edges $x'_1 \to w'_1 \to \cdots \to w'_{q-1}$ and $x'_1 \to v'_1 \to \cdots \to v'_{p-1}$, and back edges $w'_{q-1} \to x'_0$ and $v'_{p-1} \to x'_0$. All the bridges of J' in $G_{B'}$ are B^P-bridges. The analysis of paths and bridges of $G_{B'}$ can be done in a manner similar to the *basis case*.

Case 2:(Figure 9.)
Next, assume that B' *has three or more vertices of attachment*: x'_0, x'_1, ..., x'_n such that $x'_0 < x'_1 < \cdots < x'_n$,. Hence, B' is a proper bridge, with its left- and right-most vertices of attachment at s' and t'.

The carrier of B' can be written as $Q'[x'_1; x'_0] = x'_1 \to w'_1 \to \cdots \to w'_{q-1} \to x'_0$, where all but the last edge $w'_{q-1} \to x'_0$ are tree edges. Let $G_{B'} = \langle E(B') \cup E(Q') \rangle$. Then, either $s' = x'_0$ and $t' = x'_1$, or $s' = x'_1$ and $t' = x'_0$; without loss of generality, assume the former.

Hence B' consists of a tree edge $x'_n \to v'_1$ plus a subtree with root at v'_1 plus all back edges leading from the subtree. $\text{LOW1}(v'_1) = x'_0$ and $\text{LOW2}(v'_1) = x'_1$. Let $L_1 = \text{PATH}(x'_n \to v'_1) = x'_n(= v'_0) \to v'_1 \to \cdots \to v'_{m-1} \to x'_0(= v'_m)$, where all but the last edge $v'_{m-1} \to x'_0$ are tree edges,

Figure 9: *The paths found in the Induction Step: Special B^P-bridge with three or more vertices of attachment.*

and $x'_n < v'_1 < \cdots < v'_{m-1}$. Let

$$y = \text{MAX}\Big(\Big\{v'_i : 1 \leq i \leq m-1 \text{ and } \text{LOW2}(v'_i) = x'_1\Big\}\Big).$$

For all $1 \leq j \leq m-1$, if $v'_j \leq y$ then $\text{LOW2}(v'_j) = x'_1$, and if $v'_j > y$ then $\text{LOW2}(v'_j) > x'_1$.

Let S be the segment of L_1 in B'; there are two possibilities:

- *S consists of a back edge $y \to x'_1$.* Let $L_2 = \text{PATH}(y \to x'_1) = y(= u'_0) \to x'_1(= u'_1)$.
- *S consists of a tree edge $y \to u'_1$ plus a subtree with root at u'_1 plus all back edges leading from the subtree, and $\text{LOW1}(u'_1) = x'_1$.* Let $L_2 = \text{PATH}(y \to u'_1) = y(= u'_0) \to u'_1 \to \cdots \to u'_{l-1} \to x'_1(= u'_l)$, where all but the last edge $u'_{l-1} \to x'_1$ are tree edges, and $y < u'_1 < \cdots < u'_{l-1}$. $\text{LOW1}(u'_1) = \cdots = \text{LOW1}(u'_{l-1}) = x'_1$.

Let $P'[x'_0; x'_1] = (L_1[y; x'_0])^{\mathrm{R}} * L_2[y; x'_1]$. $J' = P' \cup Q'$ is a cycle in $G_{B'}$ and $R' = L_1[x'_n; y]$, a cross-cut of J' between x'_n (an internal vertex of Q') and y (an internal vertex of P'.) Let $\Theta' = J' \cup R'$ be a subgraph of $G_{B'}$.

Following proposition is immediate from the definition of palm tree:

Proposition 5.2 *Let B' be a special bridge with three or more vertices of attachment and L_1, L_2, J' and Θ', as defined earlier. Then*

1. *L_1 and L_2 are well defined.*
2. *If B is a bridge of Θ' in $G_{B'}$ with a vertex of attachment on $P']x'_0; x'_1[$ then all its vertices of attachment on P' lie either on $P'[x'_0; y]$ or on $P'[y; x'_1]$.* □

Thus, every bridge of Θ' in $G_{B'}$ is either a segment of L_1 in B', other than S, or a segment of L_2 in S. We classify the bridges of Θ' as follows:

Definition 5.2 Let $J = P[s;t] \cup Q[s;t]$ be a cycle in G. Let $R[x;y]$ be cross-cut of J between x, an internal vertex of Q, and y, an internal vertex of P. The subgraph, $\Theta = J \cup R$, of G, is called a Θ-*Graph* in G. A bridge B of Θ in G is a

1. B^P-BRIDGE, if B has *at least one* vertex of attachment on $P]s;t[$, but *none* on $Q]s;t[$ or $R]x;y[$.

2. B^Q-BRIDGE and B^R-BRIDGE are defined in a similar manner.

3. B^{PQR}-BRIDGE, if B has *at least one* vertex of attachment on each of the two or more out of the three subpaths: $P]s;t[$, $Q]s;t[$ and $R]x;y[$. □

Let u be a vertex on $P']x'_0;x'_1[$, and $e = u \to v$ be an edge, not belonging to P'. Let B be the bridge of Θ' in $G_{B'}$, equal to the segment associated with e. If e is a back edge and $x'_1 < v < y$ then B is a degenerate B^{PQR}-bridge. If e is a tree edge and $x'_1 < \text{LOW1}(v) < y$, or $x'_1 < \text{LOW2}(v) < y$ then B is a proper B^{PQR}-bridge. Otherwise, B is a B^P-bridge. If a B^P-bridge B has a vertex of attachment at x'_0 or at x'_1 then it is said to be a *Special* B^P*-bridge*; otherwise, a *Normal* B^P*-bridge*.

If B is a B^P-bridge then we can obtain in linear time some additional information, such as whether B is normal or special, and the extremal vertices of attachment of B using the LOW values of v. We omit the details.

6 Implementation: The Auxiliary Data Structure

In this section we present a data structure for the set $\mathcal{B}^L$ and $\mathcal{V}^L$, generated at each recursive call to the algorithm ANALYZE-BRIDGES. The main function of the data structure is to allow fast implementation of the steps FIND-AND-UPDATE of the algorithm ANALYZE-BRIDGES.

6.1 The Data Structure

Assume that we are given the path L, the sets: $\mathcal{B}^L$ and $\mathcal{V}^L$, and for each bridge B of $\mathcal{B}^L$, its left- and right-most vertices of attachment, and whether it has three or more vertices of attachment. We describe a data structure that supports the following operation:

6.1.1 The 'Find-and-Update' Operation Supported by the Data Structure

Choose a *Pair* $\langle v, B\rangle$, where $v \in \mathcal{V}'$ and $B =$ an unmarked bridge, maximally covering v, if such a bridge exists; $B = \perp$, otherwise. Remove v from $\mathcal{V}'$.

Let $\mathcal{B}' \subseteq \mathcal{B}^L$ be the set of bridges, whose carriers intersect with the subpath $L]s_L(B);t_L(B)[$. An unmarked bridge in $\mathcal{B}' \setminus B$ has a vertex of attachment on $L]s_L(B);t_L(B)[$; add its vertices of attachment to $\mathcal{V}'$. Mark all the unmarked bridges of $\mathcal{B}'$.

Initially, the data structure contains the sets: $L[s;t]$, $\mathcal{V}^L$ and $\mathcal{B}^L$, with all the bridges of $\mathcal{B}^L$ *unmarked*. □

We define the *one-dimensional closed (integral) interval* from x to y,

$$[x; y] = \Big\{ i \in \mathbf{N} : x \leq i \leq y \Big\}, \quad \text{where } x, y \in \mathbf{N} \text{ and } x \leq y,$$

as the set of all integers between the integer x and the integer y, including x and y. The *half-open intervals* $]x; y]$ and $[x; y[$ and the *open intervals* $]x; y[$ from x to y are defined in a similar manner. An interval $[x_i; y_i]$ is a *subinterval* of the interval $[x_j; y_j]$, *i.e.*, $[x_i; y_i] \subseteq [x_j; y_j]$, if $x_i \geq x_j$ and $y_i \leq y_j$, and a *strict subinterval*, *i.e.*, $[x_i; y_i] \subset [x_j; y_j]$, if either of the inequalities is strict. The subinterval relation is extended to half-open or open intervals in a natural way.

Hence, any closed interval $[x; y] \subseteq [1; k]$ can be visualized as a grid point (x, y) in a $k \times k$ square, $[1; k] \times [1; k]$. Because $x \leq y$, this *representative point* must lie on or above and to the left of the diagonal line $x = y$. A set of subintervals of $[1; k]$ can thus be visualized as a set of points in the upper left half triangular region, $\{(x, y) \mid 1 \leq x \leq y \leq k\}$, of the $k \times k$ square.

Let $p_i = (x_i, y_i)$ and $p_j = (x_j, y_j)$ be two distinct points in the upper left half region excluding the diagonal line. Then

$$p_i \succ p_j \ \ (\text{or } p_j \prec p_i), \qquad \text{if } x_i > x_j, \text{ or } x_i = x_j \text{ and } y_i < y_j.$$

The relation $\succ$ defines a linear order. Let $P = \Big\{ p_1 = (x_1, y_1),\ p_2 = (x_2, y_2),\ \ldots,\ p_n = (x_n, y_n) \Big\}$ be a sequence of n distinct points in the upper left half region excluding the diagonal line.

1. The point $p_i \in P$ covers a point (u, u) on the diagonal line, if $x_i < u$ and $y_i > u$; and p_i *immediately covers* (u, u), if, in addition, no $p_l \in P$ such that $p_l \succ p_i$, covers (u, u).

2. The point $p_i \in P$ *covers* a point $p_j \in P$, if $x_i \leq x_j$ and $y_i \geq y_j$; and p_i *immediately covers* p_j, if, in addition, no $p_l \in P$ such that $p_l \succ p_i$, covers p_j.

3. The point $p_i \in P$ *l-interlaces* (*i.e.*, interlaces from left) with the point $p_j \in P$, if $x_i < x_j$ and $x_j < y_i < y_j$.

The *immediate cover* relation defines a forest structure among the points of P and the points of the diagonal line and $\text{FATHER}(q) = p$ in the trees, if p immediately covers q. (See Figure 10.)

6.1.2 The Schematic Representation of the Data Structure

Let $v_1 (= s)$, v_2, ..., $v_k (= t)$ be the sequence of vertices of the path $L[v_1; v_k]$, ordered from left to right. From now on, we assume that each vertex, v_u is identified by its index u. Hence the vertex u' is *to the left of* the vertex u'', if $u' \leq u''$; and u', *strictly to the left of* u'', if the inequality is strict. Hence, the path L can be represented by the interval $[1; k]$, and a subpath $L[x; y]$ $(1 \leq x \leq y \leq k)$ of L, by the interval $[x; y]$.

- **$L[s; t]$:** Each vertex u of L has its *representative point* (u, u) on the diagonal line.

- **$\mathcal{B}^L$:** Let the set of bridges $\mathcal{B}^L$ be partitioned into the classes, $\mathbf{B}_1$, $\mathbf{B}_2$, ..., $\mathbf{B}_n$ where each class contains the bridges with same carrier on L. Each such set of bridges $\mathbf{B}_i \subseteq \mathcal{B}^L$, with the same carrier $L[x_i; y_i]$, $(x_i < y_i)$, has its *representative point* $p_i = (x_i, y_i)$ above and to the left of the diagonal line. We also say, $x(p_i) = x_i$, $y(p_i) = y_i$ and $\mathbf{B}(p_i) = \mathbf{B}_i$. Let

Figure 10: *A pictorial representation of the* FOREST *data structure.*

$P = \{p_1, p_2, \ldots, p_n\}$ be the set of representative points of $\mathbf{B}_1, \mathbf{B}_2, \ldots, \mathbf{B}_n$, ordered by the $\succ$ relation. With each point p_i, there is a flag MARK, which is TRUE, if *all* the bridges of $\mathbf{B}(p_i)$ are marked; otherwise FALSE. Initially, all the flags have the truth value FALSE. Since we know the left- and right-most vertices of attachment of every bridge $B \in \mathcal{B}^L$, this representation can be obtained in time linear in $|L[s;t]| + \left|\mathcal{B}^L\right|$.

1. The forest F is the structure induced by the immediate cover relation over the points on the diagonal line and the points of P.
2. Associated with each point $p \in P$, there is a sequence, $\mathit{left}(p) = [p_1, p_2, \ldots, p_m]$ of points, where
$$p_m = \text{MAX}_{\succ}\Big\{p' : p' \text{ l-interlaces with } p\Big\},$$
$p_i = \text{FATHER}(p_{i+1})$, $(1 \le i < m)$ and $\text{FATHER}(p_1) = \text{FATHER}(p)$.
3. Associated with each point (u, u), there is a set, $\mathit{above}((u, u)) = \{p_1, p_2, \ldots, p_m\}$ of points, such that $u = x(p_1) = x(p_2) = \cdots = x(p_m)$.

The sequences *left* and *above* are implemented by *endogenous single linear lists*, and hence allow *insertion in the front* in $O(1)$ time and *scanning* of the elements in order in $O(1)$ time per element scanned. The representation given in (1), (2) and (3) can be built in time linear in $\Big(|L[s;t]| + |P|\Big) = O\Big(|L[s;t]| + \left|\mathcal{B}^L\right|\Big)$.

- $\mathcal{V}'$**:** The set of vertices in $\mathcal{V}'$ is represented by a *stack* $\mathcal{S}$ and a *bag* $\mathcal{T}$ such that $\mathcal{S} \cup \mathcal{T} = \mathcal{V}'$, $\mathcal{S} \cap \mathcal{T} = \emptyset$. Initially, the stack $\mathcal{S}$ is empty, and the bag $\mathcal{T}$ contains exactly the vertices of initial $\mathcal{V}'$. For the *bag* we use an *endogenous single linear list*, which allows both *insertion* and *deletion* in $O(1)$ time. The time taken to obtain this representation is linear in $\left|\mathcal{V}^L\right| \leq |L[s;t]|$.

Hence, the FOREST data structure can be built in time linear in $O\Big(|L| + \left|\mathcal{V}^L\right| + \left|\mathcal{B}^L\right|\Big)$. $\square$

6.2 Building the Forest Structure

The following algorithm BUILD-FOREST builds the forest structure described in the last subsection and is based on the scan line paradigm.

```
Algorithm BUILD-FOREST(k, P):
begin
      for u := 1 to k loop  FATHER((u, u)) := nil ;
      for i := 1 to n loop
            FATHER(p_i) := nil ;
            left(p_i) := [ ];
      end{loop };

      Stack := [ ];
      for i := 1 to n loop
            low := x(p_i) + 1;

            while Stack ≠ [ ] cand y(p_i) ≥ y(TOP(Stack)) loop
                  p := TOP(Stack);
                  high := x(p);
                  for u := low to high loop  FATHER((u, u)) := p_i;
                  FATHER(p) := p_i;
                  low := y(p);
                  POP(Stack);
            end{loop };

            if Stack = [ ] cor y(p_i) ≤ x(TOP(Stack)) then
                  high := y(p_i) − 1;
            else
                  high := x(TOP(Stack));
                  left(p) := [p_i] & left(p);
            end{if };

            for u := low to high loop  FATHER((u, u)) := p_i;
            PUSH(p_i, Stack);
      end{loop };
end{BUILD-FOREST}  □
```

Lemma 6.1 *The algorithm* BUILD-FOREST *correctly builds the forest structure in time* $O\Big(|L[s;t]| + |P|\Big)$.

PROOF.

Let $Stack^0 = [\,]$.

If $Stack^{i-1} = [q_m, \ldots, q_{j+1}, q_j, q_{j-1}, \ldots, q_1]$ then $Stack^i = [p_i, q_j, \ldots, q_1]$, where $y(q_m), \ldots, y(q_{j+1}) \leq y(p_i)$ and $y(q_j) > y(p_i)$.

By induction on i, it is seen that the following invariants on *Stack* hold:

1. STAIR CASE PROPERTY:
   $x(p_i) < x(q_j) < x(q_{j-1}) < \cdots < x(q_1);$ and $y(p_i) < y(q_j) < y(q_{j-1}) < \cdots < y(q_1)$.

2. $p_i = \text{FATHER}(q_m) = \cdots = \text{FATHER}(q_{j+1})$, and if p_i l-interlaces with q_j then $p_i \in \mathit{left}(q_j)$.

From the way the FATHER and *left* links are created in the algorithm, and by virtue of the above invariants, it is immediate that BUILD-FOREST correctly builds the forest structure. The linearity of the algorithm follows from the observation that the total number of operations is same as the number of FATHER fields in the forest and the sum of the length of all the *left* lists. □

Let $p \in P$ be a point; with p, we associate a rectangular region, $R(p)$ as follows:

$$R(p) = \begin{cases} [x(q); k] \times [x(q); k], & \text{if } q = \text{FATHER}(p); \\ [1; k] \times [1; k], & \text{Otherwise.} \end{cases}$$

Let $a \geq x(p)$. The following algorithm LIST-LEFT(p, a) lists all the points of the set $\mathcal{L} = \left\{q' \in R(p) : q' \text{ l-interlaces with } p \text{ and } y(q') > a\right\}$, ordered by the $\succ$ relation. The structure *List* in the algorithm is implemented by an *endogenous single circular list* and allows *insertion in the front* and *concatenation* in $O(1)$ time, since we allow concatenation to destroy its inputs.

```
Algorithm LIST-LEFT(p, a):
begin
      List := [ ];
      p' := FIRST(left(p));
      while p' ≠ undefined cand y(p') > a loop
            TempList := LIST-LEFT(p', a);
            TempList := [p'] & TempList;
            List := TempList & List;
            p' := NEXT(p', left(p));
      end{loop };
      return list;
end{LIST-LEFT}   □
```

Lemma 6.2 *Let p be a point and let $a \geq x(p)$. Then the algorithm* LIST-LEFT*(p, a) lists all the points of the set*

$$\mathcal{L} = \left\{q' \in R(p) : q' \text{ l-interlaces with } p \text{ and } y(q') > a\right\},$$

ordered by the $\succ$ relation. The algorithm is linear in the number of points listed.

PROOF.
The linearity of the algorithm follows from the representation of the lists and the fact that the total number of insertion and concatenation operations is proportional to the number of points listed. The correctness can be shown by a simple inductive argument. □

6.3 Using the Data Structure

```
Algorithm Find-and-Update:
begin
    if S ≠ [ ] then v := TOP(S); POP(S);
    else Choose any element v from T;   T := T \ {v};
    end{if };
    p' := FATHER(v);
    if p' = nil cor p' = marked then Pair := ⟨v, ⊥⟩;
    else
        p := p';   p' := FATHER(p);
        while p' ≠ nil cand p' = unmarked loop
            p := p';   p' := FATHER(p);
        end{loop };
        Let B ∈ B(p) be a bridge with exactly two vertices of attachment,
            if such a bridge exists; otherwise, B ∈ B(p), chosen arbitrarily;
        Pair := ⟨v, B⟩;
    end{if };
    if p ≠⊥ then
        TempList := LIST-LEFT(p, s_L(B));
        p' := FIRST(TempList);
        while p' ≠ undefined loop
            if p' is unmarked then
                for every B' ∈ B(p') loop
                    (* Let W(B') = vertices of attachment of B'. *)
                    PUSH(s_L(B'), S);
                    T := T ∪ (W(B') \ {s_L(B')});
                end{loop };
                Mark p';
            end{if };
            p' := NEXT(p');
        end{loop };
        for x' := x(p) to y(p) − 1 loop
            for every p' ∈ above((x', x')) loop
                if p' = p then
                    for every B' ∈ B(p') loop
                    (* Let W(B') = vertices of attachment of B'. *)
                    if B' ≠ B cand |W(B')| ≥ 3 then T := T ∪ W(B');
                    end{loop };
                    Mark p';
                elsif p' is unmarked then
                    for every B' ∈ B(p') loop
                        (* Let W(B') = vertices of attachment of B' *)
                        T := T ∪ W(B');
                    end{loop };
                    Mark p';
                end{if };
            end{loop };
        end{loop };
end{Find-and-Update}   □
```

Figure 11: *The point p_3 maximally covers the point (v, v). All the points in the shaded region are marked at the end of the* FIND-AND-UPDATE *operation.*

At a first glance, the algorithm appears to be complicated in how it handles the bridges interlacing from left as compared to the bridges interlacing from right. This asymmetry is a result of the order imposed on the points associated with the bridges, and the manner in which it is reflected in the *immediate cover* (i.e. FATHER) relation. $\Big($ Consider three points p_1, p_2 and p_3 with $x(p_1) < x(p_2) < x(p_3)$ and $y(p_3) < y(p_1) < y(p_2)$. Note that p_1 covers p_3, but this relation cannot be derived as a transitive closure of *immediate cover*. In this example, if p_2 l-interlaces some other point and gets marked by a FIND-AND-UPDATE operation then $x(p_2)$ should be handled prior to a $v \in]x(p_3); y(p_3)[$. $\Big)$ Thus, we need to handle the bridges interlacing from left in an ordered way, while the bridges interlacing from right can be handled in an arbitrary order. The *left* structure, LIST-LEFT algorithm, in conjunction with the stack $\mathcal{S}$ in the algorithm FIND-AND-UPDATE, provide efficient mechanisms to scan the l-interlacing bridges in the desired order. Following lemma clarifies the intuition.

Lemma 6.3 *The algorithm* FIND-AND-UPDATE *correctly implements the operation* FIND-AND-UPDATE *on the data structure.*

PROOF.

First we claim that the stack $\mathcal{S}$ and the points of the data structure satisfy the following two invariants, respectively:

1. The elments of the stack $\mathcal{S}$ are in a nondecreasing order, the TOP element being a smallest.

2. (a) If r is a point marked then all points covering r are also marked.

(b) If, in addition, $x(r) \notin \mathcal{S}$ then all points that l-interlace with r are marked.

Clearly, the invariant is satisfied initially.

Case 1: The *Pair* $= \langle v, \perp \rangle$. Assume FATHER$((v,v)) = q \neq$ **nil**, since otherwise there is nothing to prove. Hence q must be marked. Since $x(q) < v$, $x(q) \notin \mathcal{S}$. From the invariant on the data structure, it follows that every $p_l \prec q$ covering (v,v) must be marked. But since q immediately covers (v,v) no $p_l \succ q$ covers (v,v). Hence there is no unmarked bridge covering v.

Next we show that, after this operation on the data structure, the invariants are satisfied. The stack obviously satisfies the condition. Hence, assume that the second invariant does not hold. Let r be a point violating the invariant; r is marked. Then it must be the case that $x(r) = v$ and $v \in \mathcal{S}$ just before the FIND-AND-UPDATE operation, and there is an unmarked point l-interlacing with r. But since such a point also covers (v,v), this provides a contradiction.

Case 2: The *Pair* $= \langle v, p \rangle$, where $p = (x, y)$. Assume FATHER$(p) = q \neq$ **nil**, since otherwise there is nothing to prove. Hence q must be marked. Since $x(q) < v$, $x(q) \notin \mathcal{S}$. From the invariant on the data structure, it follows that every $p_l \prec q$ covering p must be marked. But since q immediately covers p no $p_l \succ q$ covers p. Hence there is no unmarked point covering p. Since all the bridges of $\mathbf{B}(p)$ are unmarked, it is easy to see that the bridge $B \in \mathbf{B}(p)$ chosen by the algorithm is in fact the maximal unmarked bridge covering v.

Let us consider a bridge $B' \in \mathcal{B}'$ whose carrier intersects with $L]s_L(B); t_L(B)[$, *i.e.*, the interval $]x; y[$. Hence the representative point, $p' = (x', y')$, of B' has an interval $[x'; y']$ that intersects with $]x; y[$. If $p' \notin R(p)$ then p' either covers or l-interlaces with $q =$ FATHER(p). Thus, p', and hence the bridges of $\mathbf{B}(p')$ must be marked. This implies that we only need to consider such $p' \in R(p)$. Since p is not covered by any unmarked point, we may further exclude from consideration the p's covering p. It is trivial to see that at the end of the operation all such points (hence there bridges) are marked. The correctness of the operation follows from the following observations.

1. $x' < x$. Then if p' is an unmarked point, it is in LIST-LEFT$(p, x(p))$, and p' l-interlaces with p. Hence, every $B' \in \mathbf{B}(p')$ is unmarked and has a vertex of attachment on $L]s_L(B); t_L(B)[$. Its vertices of attachment are added to $\mathcal{V}' = \mathcal{S} \cup \mathcal{T}$.

2. $x' = x$.

   - $p' = p$. Hence every $B' \in (\mathbf{B}(p) \setminus B)$, with three or more vertices of attachment, is unmarked and has a vertex of attachment on $L]s_L(B); t_L(B)[$. Its vertices of attachment are added to $\mathcal{V}' = \mathcal{S} \cup \mathcal{T}$.
   - $p' \in (above((x,x)) \setminus \{p\})$. Then if p' is unmarked, $y' < y$. Hence, every $B' \in \mathbf{B}(p')$ is unmarked and has a vertex of attachment on $L]s_L(B); t_L(B)[$. Its vertices of attachment are added to $\mathcal{V}' = \mathcal{S} \cup \mathcal{T}$.

3. $x < x' < y$. $p' \in above((x', x'))$. If p' is unmarked, every $B' \in \mathbf{B}(p')$ is unmarked and has a vertex of attachment on $L]s_L(B); t_L(B)[$. Its vertices of attachment are added to $\mathcal{V}' = \mathcal{S} \cup \mathcal{T}$.

Next we show that, after this operation on the data structure, the invariants are satisfied. Since LIST-LEFT lists the points such that they are ordered by the $\succ$ relation, $x(q'_r) \geq x(q'_{r-1}) \geq \cdots \geq x(q'_1)$, and thus the stack obviously satisfies the condition. Hence, assume that the second invariant does not hold. Let r be a point violating the invariant; r is marked. We may assume that $r \in R(p)$ and $x(r) < y$. If $x \leq r < y$ then every point covering r or l-interlacing with it is marked at the end of the operation, $x(r) < x < v$. Hence, using the invariant and the fact that $x(r)$ was not in the stack $\mathcal{S}$ before the operation, we see that such an r must have been unmarked before the operation. Hence $r \in$ LIST-LEFT(p); but since at the end of the operation, every point covering r is marked, and $x(r) \in \mathcal{S}$, this provides a contradiction. □

Lemma 6.4 *Let L, $\mathcal{V}^L$ and $\mathcal{B}^L$ be as described in the beginning of this section. Let $\mathcal{O}_1$, $\mathcal{O}_2$, ..., $\mathcal{O}_{m'}$ be a sequence of* FIND-AND-UPDATE *operations applied to $\langle L, \mathcal{V}^L, \mathcal{B}^L\rangle$ such that at the end of the m'^{th} operation the set $\mathcal{V}' = \emptyset$.*

Let $\mathcal{Q}' = \langle v_1', B_1'\rangle$, $\langle v_2', B_2'\rangle$, ..., $\langle v_{m'}', B_{m'}'\rangle$ be the sequence of values of the Pair*'s, where $\langle v_i, B_i\rangle$ is the* Pair *returned by the operation $\mathcal{O}_i$. Let $\mathcal{Q} = \langle v_1, B_1\rangle$, $\langle v_2, B_2\rangle$, ..., $\langle v_m, B_m\rangle$ be the subsequence of $\mathcal{Q}'$ consisting of all* Pair*'s $\langle v_i', B_i'\rangle$ such that $B_i' \neq \perp$. Let $\mathcal{B}_1^L = \{B_1, B_2, \ldots, B_m\}$, and $\mathcal{B}_2^L = \mathcal{B}^L \setminus \mathcal{B}_1^L$.*

Then the total time taken to build the initial structure plus the time taken by the operations $\mathcal{O}_1$, $\mathcal{O}_2$, ..., $\mathcal{O}_{m'}$ is

$$T\Big(L, \mathcal{V}^L, \mathcal{B}^L\Big) = O\Big(|L| + \left|\mathcal{V}^L\right| + \sum_{B \in \mathcal{B}_2^L} |E(B)|\Big).$$

PROOF.
First observe that if B_i and B_j are two distinct bridges selected by $\mathcal{Q}$ then their carriers are internally vertex disjoint. Thus $\left|\mathcal{B}_1^L\right| \leq |L|$. Also, $\left|\mathcal{B}_2^L\right| \leq \sum_{B \in \mathcal{B}_2^L} |E(B)|$. Hence,

$$\left|\mathcal{B}^L\right| = O\Big(|L| + \sum_{B \in \mathcal{B}_2^L} |E(B)|\Big).$$

Now, from the earlier discussion, it follows that the time taken to build the data structure is

$$O\Big(|L| + \left|\mathcal{V}^L\right| + \left|\mathcal{B}^L\right|\Big) = O\Big(|L| + \left|\mathcal{V}^L\right| + \sum_{B \in \mathcal{B}_2^L} |E(B)|\Big).$$

Clearly, the total number of operations performed is bounded as follows:

$$m' \leq \left|\mathcal{V}^L\right| + \sum_{B \in \mathcal{B}_2^L} |W(B)| = O\Big(\left|\mathcal{V}^L\right| + \sum_{B \in \mathcal{B}_2^L} |E(B)|\Big).$$

It remains to show that the total amount of time spent by the sequence of operations is appropriately bounded: Let the *Pair* selected, in the i^{th} operation $\mathcal{O}_i$, be $< v_i', B_i' >$. If $B_i' = \perp$ then the amount of time spent by the FIND-AND-UPDATE operation is $O(1)$. Otherwise, the amount of time spent by the FIND-AND-UPDATE operation is bounded by the sum of the followings:

1. Total number of points in the region

$$\Big([1; t_L(B_i)[\times]s_L(B_i); k]\Big) \setminus \Big([1; s_L(B_i)[\times]t_L(B_i); k]\Big).$$

2. The time taken to compute the vertices of attachment of those unmarked bridges of $\mathcal{B}' \setminus B_i$ that have at least one vertex of attachment on $L]s_L(B_i); t_L(B_i)[$. If B is such a bridge then $W(B)$ can be computed in time $O\Big(|E(B)| + |V(B)|\Big) = O(|E(B)|)$ time, using the palm tree of the previous section.

Hence, a bound for the total time spent by the sequence of operations can be obtained by deducing bound for the contributions due to (1) and (2). Since any point belongs to at most two different regions of (1), the total contribution due to (1) is

$$O\Big(\Big|\mathcal{B}^L\Big|\Big) = O\Big(|L| + \sum_{B \in \mathcal{B}_2^L} |E(B)|\Big).$$

The total contribution due to (2) is bounded by $O\Big(\sum_{B \in \mathcal{B}_2^L} |E(B)|\Big)$. Summing up, we obtain the bound claimed:

$$T\Big(L, \mathcal{V}^L, \mathcal{B}^L\Big) = O\Big(|L| + \Big|\mathcal{V}^L\Big| + \sum_{B \in \mathcal{B}_2^L} |E(B)|\Big). \quad \square$$

7 The Timing Analysis of the Algorithm Find-Ambitus

Theorem 7.1 *The algorithm* FIND-AMBITUS *requires $O(|E| + |V|)$ time to find the ambitus in a nonseparable graph, $G = (E, V)$.*

PROOF.

If L is a path then its subpath containing all the vertices of L except its end vertices, is denoted by $L] \cdot \cdot [$. If v is a vertex then $d(v)$ stands for the local degree of v.

Let $P[s;t]$ and $Q[s;t]$ be the paths in the graph. If T_P and T_Q bound, respectively, the time required to modify $P[s;t]$ and $Q[s;t]$ such that G has no offensive B^P- or B^Q-bridges then the time complexity of the algorithm is bounded by $O(|E| + |V|) + T_P + T_Q$. It suffices to show that T_P and T_Q are each $O(|E| + |V|)$.

Let $\mathcal{P}$ be the palm tree of the graph G. The first path in G is J. Let $\mathcal{S}$ be the segments of J in $\mathcal{P}$, and $\mathcal{S}^P$ be the segments corresponding to B^P-bridges. Let $\mathcal{S}_1^P \subseteq \mathcal{S}^P$ be the set of segments corresponding to the bridges in the sequence of *Pairs*, $\mathcal{Q}$, found by a sequence of FIND-AND-UPDATE operations, and $\mathcal{S}_2^P = \mathcal{S}^P \setminus \mathcal{S}_1^P$.

Hence, T_P is bounded by the sum of (i) the time taken to build the palm tree, (ii) the time taken to find the path J, together with its segments, (iii) the time taken to build the data structure plus that taken by the operations on the data structure, and (iv) the time spent in each recursive call on the segments of $\mathcal{S}_1^P$.

$$\begin{aligned} T_P \;=\; & \sum_{S \in \mathcal{S}_1^P} T(S) + O\Big(|E| + |V|\Big) \\ & + O\Big(|J| + \sum_{v \in V(P]\cdot\cdot[)} d(v) + \sum_{S \in \mathcal{S} \setminus \mathcal{S}^P} |E(S)|\Big) + O\Big(|P| + \Big|\mathcal{V}^P\Big| + \sum_{S \in \mathcal{S}_2^P} |E(S)|\Big). \end{aligned}$$

Let $G_{B'} = \langle E(B') \cup E(Q') \rangle$ be composed of a bridge B' and its carrier Q'; let s' and t' be the two distinguished vertices in $G_{B'}$. Let the segment S' in $\mathcal{P}$ be the one corresponding to B'.

Case 1: If (i) B' is a normal bridge, or (ii) B' is a special bridge with exactly two vertices of attachment then the first path L found in S' is $P'[s';t']$. Let $\mathcal{S}$ be the segments of L in S', and $\mathcal{S}^L$, the segments corresponding to $B^{P'}$-bridges of $P' \cup Q'$. Let $\mathcal{S}_1^L \subseteq \mathcal{S}^L$ be the set of segments corresponding to the bridges in the sequence of *Pairs*, $\mathcal{Q}$, found by a sequence of FIND-AND-UPDATE operations, and $\mathcal{S}_2^L = \mathcal{S}^L \setminus \mathcal{S}_1^L$. Then

$$\begin{aligned} T(S') &= \sum_{S \in \mathcal{S}_1^L} T(S) + O\Big(|L| + \sum_{v \in V(L]\cdot\cdot[)} d(v) + \sum_{S \in \mathcal{S} \setminus \mathcal{S}^L} |E(S)|\Big) \\ &\quad + O\Big(|L| + \left|\mathcal{V}^L\right| + \sum_{S \in \mathcal{S}_2^L} |E(S)|\Big). \end{aligned}$$

Case 2: If B' is a special bridge with three or more vertices of attachment then the paths found in S' are L_1 and L_2, and the path $L = P'[s';t']$ is obtained from L_1 and L_2. Notice that $|L| \leq |L_1| + |L_2|$. Let $\mathcal{S}$ be the segments of L_1 and L_2 in S', and $\mathcal{S}^L$, the segments corresponding to $B^{P'}$-bridges of $P' \cup Q'$. Let $\mathcal{S}_1^L \subseteq \mathcal{S}^L$ be the set of segments corresponding to the bridges in the sequence of *Pairs*, $\mathcal{Q}$, found by a sequence of FIND-AND-UPDATE operations, and $\mathcal{S}_2^L = \mathcal{S}^L \setminus \mathcal{S}_1^L$.Then

$$\begin{aligned} T(S') &= \sum_{S \in \mathcal{S}_1^L} T(S) \\ &\quad + O\Big(|L_1| + |L_2| + \sum_{v \in V(L_1]\cdot\cdot[)} d(v) + \sum_{v \in V(L_2]\cdot\cdot[)} d(v) + \sum_{S \in \mathcal{S} \setminus \mathcal{S}^L} |E(S)|\Big) \\ &\quad + O\Big(|L| + \left|\mathcal{V}^L\right| + \sum_{S \in \mathcal{S}_2^L} |E(S)|\Big). \end{aligned}$$

Solving the recurrence relations, we get

$$T(S) = O\Big(|E(S)| + \sum_{L \in \mathcal{L}(S)} \Big(|L| + \sum_{v \in V(L]\cdot\cdot[)} d(v)\Big)\Big).$$

where $\mathcal{L}(S)$ is the set of internally vertex disjoint paths in the segment S. Hence it follows that

$$T_P = O\Big(|E| + |V| + \sum_{L \in \mathcal{L}(\mathcal{P})} \Big(|L| + \sum_{v \in V(L]\cdot\cdot[)} d(v)\Big)\Big).$$

Since the paths of the palm tree $\mathcal{P}$ are internally vertex disjoint,

$$\sum_{L \in \mathcal{L}(\mathcal{P})} \Big(|L| + \sum_{v \in V(L]\cdot\cdot[)} d(v)\Big) \leq 2 \cdot |E| + \sum_{v \in V} d(v) \leq 4 \cdot |E|,$$

and $T_P = O(|E| + |V|)$. $\square$

8 Abiding Path and Nonseparating Induced Cycle

In this section, we study two graph theoretic notions, closely related to the concept of an ambitus.

Definition 8.1 ABIDING PATH.

Let G be a graph with two distinct vertices s and t, a path $P[s;t]$ and a set of distinguished vertices $W \subseteq V$. A bridge B of $P[s;t]$ is said to be a W^+-*bridge*, if $W \cap N(B) \neq \emptyset$; otherwise, a W^--*bridge*.

Let a (W,P)-*projection* be the set

$$\Pi = \Big(W \cap V(P)\Big) \cup \bigcup_{B=W^+\text{-bridge}} W(G,B),$$

i.e., the set of vertices on $P[s;t]$, either in W or a vertex of attachment of some W^+-bridge.

A path $P[s;t]$ is said to be an *abiding path* with respect to W, if no W^--bridge covers a vertex of the (W,P)-projection, Π. $\square$

Let $\widehat{G}$ be the graph obtained from G by including additional vertices $\{s', w', t'\}$ and additional edges

$$\Big\{[s',s],[t,t'],[s',w'],[w',t']\Big\} \cup \Big\{[w,w'] : w \in W\Big\}.$$

Let $J = P[s';t'] * [t',w'] * [w',s']$ be an ambitus of the graph $\widehat{G}$, containing s' and t'. It is easily seen that the subpath $P[s;t]$ of $P[s';t']$ is an abiding path in G with respect to W.

Definition 8.2 NONSEPARATING INDUCED CYCLE.

A cycle J in a graph G is said to be a *nonseparating induced cycle* of G, if there is at most one proper bridge B of J in G. Given a 3-connected graph G, containing a distinguished vertex u and a distinguished edge $[s,t]$, of particular interest is a nonseparating induced cycle J of G, containing $[s,t]$, but not u. $\square$

Let $\widehat{G}$ be the (nonseparable) graph obtained from G by deleting the edge $[s,t]$, and $P[s;t]$, an abiding path in $\widehat{G}$ with respect to the set $\{u\}$. It is easily seen that $J = P[s;t] * [t,s]$ is a nonseparating induced cycle of G such that $[s,t] \in E(J)$ and $u \notin V(J)$.

It is trivial to see that both these problems can be solved in linear time, using the linear-time ambitus-finding algorithm.

9 Acknowledgement

We wish to acknowledge with gratitude the considerable help and the advice we have received from Profs. E.M. Clarke, R. Kannan and R. Statman, all of Carnegie Mellon University, and the constructive criticisms of two anonymous referees.

References

[1] J. Cheriyan and S.N. Maheswari. Finding Nonseparating Induced Cycles and Independent Spanning Trees in 3-Connected graphs. *Journal of Algorithms*, 9:507–537, 1988.

[2] J. Hopcroft and R. Tarjan. Dividing a Graph into Triconnected Components. *SIAM Journal of Computing*, 2, September 1973.

[3] J. Hopcroft and R. Tarjan. Efficient Planarity Testing. *Journal of Association for Computing Machinery*, 21, October 1974.

[4] S.V. Krishnan, C. Pandu Rangan, and S. Seshadri. A Simple Linear Algorithm for Finding the Ambitus in a Planar Graph. Technical report, Department of Computer Science, Indian Institute of Technology, Madras, India, March 1988.

[5] B. Mishra. An Efficient Algorithm to find All 'Bidirectional' Edges of an Undirected Graph. In *25th Annual Symposium on Foundations of Computer Science*, pages 207–216, 1984.

[6] B. Mishra. *Some Graph Theoretic Issues in VLSI Design.* PhD thesis, Carnegie-Mellon University, September 1985.

[7] T. Ohtsuki. *The Two Disjoint Path Problem and Wire Routing Design.* Number 108 in Graph Theory and Algorithms (Eds. N. Saito, T. Nishizeki). Springer, October 1980.

[8] O. Ore. *The Four-Color Problem.* Academic Press, New York, London, 1967.

[9] H. Sachs. *Einführung in die Theorie der endlichen Graphen.* Teil H.B.G. Teubner, Leipzig, 1972.

[10] R. Sundar. Finding the Abiding Path and its Applications to Graph Algorithms. Technical report, Department of Computer Science and Engineering, Indian Institute of Technology, Madras, India, June 1987.

[11] Robert Tarjan. Depth-First Search and Linear Graph Algorithms. *SIAM Journal of Computing*, 1(2), June 1972.

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

[13] W.T. Tutte. Bridges and Hamiltonian Circuits in Planar Graphs. *Aequationes Mathematicae*, 15, 1977.