1

Planarity Testing and Embedding

Maurizio Patrignani
Roma Tre University

Testing the planarity of a graph and possibly drawing it without intersections is one of the most fascinating and intriguing algorithmic problems of the graph drawing and graph theory areas. Although the problem *per se* can be easily stated, and a complete characterization of planar graphs is known since 1930, the first linear-time solution to this problem was found only in the seventies of the last century.

Planar graphs play an important role both in the graph theory and in the graph drawing areas. In fact, planar graphs have several interesting properties: for example they are sparse, four-colorable, allow a number of operations to be performed more efficiently than for general graphs, and their inner structure can be described more succinctly and elegantly (see Section 1.1.2). From the information visualization perspective, instead, as edge crossings turn out to be the main responsible for reducing readability, planar drawings of graphs are considered clear and comprehensible.

In this chapter we review a number of different algorithms from the literature for efficiently testing planarity and computing planar embeddings. Our main thesis is that all known linear-time planarity algorithms fall into two categories: cycle-based algorithms and vertex-addition algorithms. The first family of algorithms is based on the simple observation that in a planar drawing of a graph any cycle necessarily partitions the graph into the inside and outside portion, and this partition can be suitably used to split the embedding problem. Vertex addition algorithms are based on the incremental construction of the final planar drawing starting from planar drawings of smaller graphs. The fact that some algorithms were based on the same paradigm was already envisaged by several researchers [Tho99, HT08]. However, the evidence that all known algorithms boil down to two simple approaches is a relatively new concept.

0-8493-8597-0/01/$0.00+$1.50

The chapter is organized as follows: Section 1.1 introduces basic definitions, properties, and characterizations for planar graphs; Section 1.2 formally defines the planarity testing and embedding problems; Section 1.3 follows an historic perspective to introduce the main algorithms and a conventional classification for them. Some algorithmic techniques are common to more than one algorithm and sometimes to all of them. These are collected in Section 1.4. Finally, the two Sections 1.5 and 1.6 are devoted to the two approaches to the planarity problem, namely the "cycle-based" and the "vertex-addition" approaches, respectively.

1.1 Properties and Characterizations of Planar Graphs

1.1.1 Basic Definitions

A *graph* $G(V, E)$ is an ordered pair consisting of a finite set V of *vertices* and a finite set E of *edges*, that is, pairs (u, v) of vertices. If each edge is an unordered (ordered) pair of vertices, then the graph is *undirected* (*directed*). An edge (u, v) is a *self-loop* if $u = v$. A graph $G(V, E)$ is *simple* if E is not a multiple set and it does not contain self-loops. For the purposes of this chapter we can restrict us to simple graphs.

The sets of edges and vertices of G can be also denoted $E(G)$ and $V(G)$, respectively. If edge $(u, v) \in E$, vertices u and v are said to be *adjacent* and (u, v) is said to be *incident* to u and v. Two edges are *adjacent* if they have a vertex in common.

A (rooted) *tree* T is a connected acyclic graph with one distinguished vertex, called the *root* r. A *spanning tree* of a graph G is a tree T such that $V(T) = V(G)$ and $E(T) \subseteq E(G)$.

Given two graphs $G_1(V_1, E_1)$ and $G_2(V_2, E_2)$, their *union* $G_1 \cup G_2$ is the graph $G(V_1 \cup V_2, E_1 \cup E_2)$. Analogously, their *intersection* $G_1 \cap G_2$ is the graph $G(V_1 \cap V_2, E_1 \cap E_2)$. A graph G_2 is a *subgraph* of G_1 if $G_1 \cup G_2 = G_1$. A *subdivision* of an edge (u, v) consists of the insertion of a new node w and the replacement of (u, v) with edges (u, w) and (w, v). A graph G_2 is a *subdivision* of G_1 if it can be obtained from G_1 through a sequence of edge subdivisions.

A *drawing* Γ of a graph G maps each vertex v to a distinct point $\Gamma(v)$ of the plane and each edge (u, v) to a simple open Jordan curve $\Gamma(u, v)$ with endpoints $\Gamma(u)$ and $\Gamma(v)$. A drawing is *planar* if no two distinct edges intersect except, possibly, at common endpoints. A graph is *planar* if it admits a planar drawing. A planar drawing partitions the plane into connected regions called *faces*. The unbounded face is usually called *external face* or *outer face*. If all the vertices are incident to the outer face the planar drawing is called *outerplanar* and the graph admitting it is an *outerplanar graph*. Given a planar drawing, the (clockwise) circular order of the edges incident to each vertex is fixed. Two planar drawings are *equivalent* if they determine the same circular orderings of the edges incident to each vertex (sometimes called *rotation scheme*). A *(planar) embedding* is an equivalence class of planar drawings and is described by the clockwise circular order of the edges incident to each vertex. A graph together with one of its planar embedding is sometimes referred to as a *plane* graph.

A *path* is a sequence of distinct vertices $v_1, v_2, \ldots, v_k$, with $k \geq 2$, together with the edges $(v_1, v_2), \ldots, (v_{k-1}, v_k)$. The *length* of the path is the number of its edges.

A *cycle* is a sequence of distinct vertices $v_1, v_2, \ldots, v_k$, with $k \geq 2$, together with the edges $(v_1, v_2), \ldots, (v_{k-1}, v_k), (v_k, v_1)$. The *length* of a cycle is the number of its vertices or the number of its edges.

An undirected graph G is *connected* if, for each pair of nodes u and v, G contains a path from u to v. A *k-connected* graph G is such that removing any $k - 1$ vertices leaves G connected; 3-connected, 2-connected, and 1-connected graphs are also called *triconnected*,

biconnected, and *simply connected* graphs, respectively. A *separating k-set* is a set of k vertices whose removal disconnects the graph. Separating 1- and 2-sets are called *cutvertices* and *separation pairs*, respectively. Hence, a connected graph is biconnected if it has no cutvertices and it is triconnected if it has no separation pairs.

If a graph G is not connected, its maximal connected subgraphs are called the *connected components* of G. If G is connected, its maximal biconnected subgraphs are called the *biconnected components* or *blocks* of G. Note that a cutvertex belongs to several blocks and that a biconnected graph has only one block. The graph whose vertices are the blocks and the cutvertices of G and whose edges link cutvertices to the blocks they belong to is a tree and is called the *block-cutvertex tree* (or *BC-tree*) of G.

Given a biconnected graph G, its *triconnected components* are obtained by a complex splitting and merging process. The first linear-time algorithm to compute them was introduced in [HT73], while an implementation of it is described in [GM01]. The computation has two phases: first G is recursively split into its *split components*; second, split components with exactly two vertices and triangle split components are merged together as much as possible to obtain parallel triconnected components and series triconnected components, respectively. The split operation is performed with respect to a separation pair $\{v_1, v_2\}$ of G. Consider one connected component G_1 obtained by removing $\{v_1, v_2\}$. Let G'_1 be the subgraph of G induced by vertices $V(G_1) \cup \{v_1, v_2\}$ and let $G'_2 = (V(G)/V(G_1), E(G)/E(G'_1))$. Observe that vertices v_1 and v_2 belong to both G'_1 and G'_2 and that edge (v_1, v_2), provided that it exists, belongs to G'_1. A *split* operation consists of replacing G with G''_1 and G''_2, where G''_1 and G''_2 are obtained from G'_1 and G'_2 by adding the *virtual edge* (v_1, v_2). The *split components* of a graph G are obtained by recursively splitting G until a separation pair can be found in the obtained graphs. Split components are not unique and, hence, are not suitable for describing the structure of G. Two split components are *adjacent* if the have the same virtual edge (v_1, v_2). Such adjacent split components could be merged by identifying the two copies of v_1 and v_2 and by removing the two copies of virtual edge (v_1, v_2). By merging together all the split components that have only two vertices $\{v_1, v_2\}$ we obtain *parallel triconnected components*. By merging together all adjacent triangle split components we obtain *series triconnected components*. Split components that are not affected by the merging operations described above are called *rigid triconnected components*.

Triconnected components are unique and are used to describe the inner structure of a graph. In fact, a graph G can be succinctly described by its SPQR-tree $\mathcal{T}$, which provides a high-level view of the unique decomposition of the graph into its triconnected components [DT96a, DT96b, GM01]. Namely, each triconnected component corresponds to a node of $\mathcal{T}$. The triconnected component corresponding to a node μ of $\mathcal{T}$ is called the *skeleton of* μ. As there are parallel, series, and rigid triconnected components, their corresponding tree nodes are called P-, S-, and R-nodes, respectively. Triconnected components sharing a virtual edge are adjacent in $\mathcal{T}$. Usually, a fourth type of node, called Q-node, is used to represent an edge (u, v) of G. Q-nodes are the leaves of $\mathcal{T}$ and they don't have skeletons. Tree $\mathcal{T}$ is unrooted, but for some applications it could be thought as rooted at an arbitrary Q-node.

The connectivity properties of a graph have a strict relationship with its embedding properties. Triconnected planar graphs (and triconnected planar components) have a single embedding up to a flip (that is, up to a reversal of all their incidence lists) [Whi32]. The same thing holds for biconnected outerplanar graphs and their unique outerplanar embedding (adding a star on the outer face yields a triconnected plane graph).

A non-connected graph is planar if and only if all its connected components are planar. Thus, in the following, without loss of generality, we only consider the planarity of connected graphs. Also, a planar embedding of a graph implies a planar embedding for each one of its

blocks, while, starting from a planar embedding of the blocks, a planar embedding for the whole graph can be found. Thus, since the blocks can be identified in linear time [Tar72], a common strategy, both to test planarity and to compute a planar embedding, is that of dividing the graph into its blocks, and tackle each block separately.

Finally, a graph is planar if and only if its triconnected components are planar [Mac37b]. More precisely, as parallel and series triconnected components are always planar, a graph is planar if and only if all its rigid triconnected components are planar. However, since dividing a graph into its triconnected components is a linear but rather laborious process [HT73, GM01], usually planarity algorithms do not assume that the input graph is triconnected.

Also, from a planar embedding of the triconnected components of a graph a planar embedding of the whole graph can be obtained. This property can be exploited to explore the planar embeddings of a given graph in search for some embedding that has a specific property (see, for example, [MW99, MW00, BDBD00, GMW01, ADF$^+$10]).

Given a plane (multi)graph G, its *plane dual* (or simply its *dual*) is the multigraph G^* such that G^* has one vertex for each face of G and two vertices of G^* are linked by one edge e^* if the corresponding faces in G share one edge e. Observe that the planar embedding of G induces a planar embedding of its dual and that the dual of the dual of G is G itself. Also, different embeddings of a planar graph G correspond to different dual graphs. Finally, a cycle in G corresponds to a minimal cut in G^* (anytime this property holds G and G^* are called *abstract dual*).

A graph $G(V, E)$ is k *colorable* if its vertices can be partitioned into k sets $V_1, V_2, \ldots, V_k$ in such a way that no edge is incident to two vertices of the same set. A graph $G(V, E)$ is *complete* if each vertex in V is adjacent to each other.

A graph $G(V, E)$ is *bipartite* if it is 2 colorable. A bipartite graph $G(V_1, V_2, E)$ is *complete* if each vertex in V_1 is adjacent to all vertices in V_2.

1.1.2 Properties

Planar graphs have a variety of properties whose exploitation allows to efficiently perform a number of operations on them.

Perhaps the most renown property is the one stated by Euler's Theorem, which shows that planar graphs are sparse. Namely, given a plane graph with n vertices, m edges and f faces, we have $n - m + f = 2$. A simple corollary is that for a maximal planar graph with at least three vertices, where each face is a triangle ($2m = 3f$), we have $m = 3n - 6$, and, therefore, for any planar graph we have $m \leq 3n - 6$. This number reduces to $m = 2n - 3$ for maximal outerplanar graphs with at least three vertices (and $m \leq 2n - 3$ for general outerplanar graphs). Also, if $n \geq 3$ and the graph has no cycle of length 3, then $m \leq 2n-4$. Finally, if the graph is a tree, then $m = n - 1$.

These considerations allow us to replace m with n in any asymptotic calculation involving planar graphs, while for general graph only $m \in O(n^2)$ can be assumed. From a more practical perspective, they allow us to decide the non-planarity of denser graphs without reading all the edges (which would yield a quadratic algorithm).

The Four Color Theorem [AH77, AHK77, RSST97] asserts that any planar graph is four colorable and settles a conjecture that was for more than a century the most famous unsolved problem in graph theory and perhaps in all of mathematics [Har69]. To stress how important this property is, it suffices to observe that, apart from being considered an important property of planar graphs, it has also been mentioned as the most notable property of the number four.

While 3-colorability is NP-hard even on maximum degree four planar graphs [GJS76], every triangle-free planar graph is 3-colorable [Grö59] and such a 3-coloring can be found

in linear-time [DKT09].

Determining if the graph contains a k-clique, i.e., a set of k pairwise adjacent vertices, is polynomial for planar graphs, as no clique can have more than four vertices. This problem is polynomial even in the weighted case, where each vertex is associated with a weight and the sum of the weights of the pairwise adjacent vertices is requested to be at least k. Observe that both these problems are NP-complete on non-planar graphs.

Graph isomorphism is linear for planar graphs [HW74], while it is of unknown complexity for general graphs [GJ79].

The planar separator theorem [LT79] states that every planar graph $G = (V, E)$ admits a partition of its n vertices into three sets A, B, and C, such that the size of C is $O(\sqrt{n})$, the size of A and B is at most $\frac{2}{3}n$, and there is no edge with one endpoint in A and the other endpoint in B. Such a partition can be found in linear time and is the starting point of a hierarchical decomposition of the graph that may lead to efficient approaches to compute properties of the graph.

1.1.3 Characterizations

The first complete characterization of planar graphs is due to Kuratowski [Kur30], and states that a graph is planar if and only if it contains no subgraph that is a subdivision of K_5 or $K_{3,3}$, where K_5 is the complete graph of order 5 and $K_{3,3}$ is the complete bipartite graph with 3 vertices in each of the sets of the partition. An equivalent later result, recasted in terms of graph minors, is Wagner's theorem that states that a graph G is planar if and only if it has no K_5 or $K_{3,3}$ as minor, that is, K_5 or $K_{3,3}$ cannot be obtained from G by contracting some edges, deleting some edges, and deleting some isolated vertices [Wag37a, HT65]. Observe that the two characterizations are different since a graph may admit K_5 as minor without having a subgraph that is a subdivision of K_5 (consider, for example, a graph of maximum degree 3).

Similarly, it can be proved that a graph is outerplanar if and only if it contains no subgraph that is a subdivision of K_4 or $K_{2,3}$. Trivially, a graph is a tree if it does not contain a subdivision (or a minor) of K_3.

If the graph is triconnected a less renown but much simpler characterization can be formulated. Namely, a triconnected graph distinct from K_5 is planar if and only if it contains no subgraph that is a subdivision of $K_{3,3}$ [Wag37b, Hal43, Kel93, Lie01].

Given a graph G with no isolated vertices, the associated height-two *vertex-edge poset* $<_G$ has $V \cup E$ as elements, and $v <_G e$ if and only if $v \in V$, $e \in E$, and v is an endpoint of edge e. The smallest number of total orders the intersection of which yields the poset is called the *dimension* of the poset. Graph G is planar if and only if its corresponding vertex-edge poset has dimension at most three [Sch89]. Unfortunately, checking if a poset has dimension at most t is proved to be NP-complete for $t \geq 3$ and for $t \geq 4$ if the poset has height two [Yan82].

Edges traversing a bipartition of the vertices of G are called a *cocycle*. Observe that while a cycle is a collection of edges that covers each vertex an even (possibly zero) number of times, a cocycle is a collection of edges that intersects each cycle in an even number of edges. A *bicycle* is a collection of edges that is both a cycle and a cocycle. Planarity can be characterized in terms of the properties of the vector spaces of cycles [Mac37a], cocycles [APBL95, LS10], and bicycles [APBL95].

A further planarity characterization is expressed via Colin de Verdiére's graph invariant $\mu(G)$, which in turn is based on the maximum multiplicity of the second eigenvalue of certain Schrödinger operators defined by the graph [Col90, Col91], and states that a graph G is planar if and only if $\mu(G) \leq 3$.

Alternative characterizations can be found in the literature based on the existence of an abstract dual graph [Whi32], on the edge poset dimension [dO96], on the relationship among theta-graph minors [AŠ98], on the orientability of circuits [LH77, Che81], on the arrangements of pseudo-lines [TT97], or on DFS traversals of the graph [dR82, dR85, SH93, SH99, BM99, BM04].

1.2 Planarity Problems

The main planarity problem is the decision problem of recognizing planar graphs, that is of deciding the planarity of the input graph. Both with the purpose of exhibiting a planarity certificate and of producing a planar embedding for information-visualization applications, planarity testing algorithms are usually coupled with planar embedding procedures, that sometimes, depending on the algorithmic approach, required themselves a considerable research effort to be devised.

On the opposite, if the graph is not planar, the search for a non-planarity certificate is called Kuratowski subgraph isolation [CMS08], and the research concentrated on planarization algorithms that allow us to produce a planar graph where some degree-four vertices have been added to replace crossings [Lie01]. Since crossing number minimization is NP-complete [GJ79] planarization algorithms use heuristics to introduce a reduced number of dummy vertices.

Dynamic algorithms have also been devised for efficiently determining planarity and computing a planar embedding of graphs where edges and vertices are added or deleted one at a time [DT89, DT96b, GIS99, DBTV01].

Efficient algorithms for planarity testing in parallel have been investigated in [KR88, RR89, RR94].

1.2.1 Constrained Planarity

The problem of determining the planarity of a graph and of computing a possible embedding of it can be combined with additional constraints on the desired drawing that result in restrictions on the set of admissible planar embeddings [Tam98, GKM08]. Typical constraints ask for some vertices to be on the same face (usually the outer face), some vertex to have a specified circular ordering of its incident edges, some path to be drawn along a straight line, etc. In the easier cases, such constraints can be enforced by replacing sets of nodes and edges of the input graph with suitable gadgets, by launching an ordinary planarity algorithm, and by transferring the results back on the original graph. More complex cases require to efficiently explore the possible embeddings of the graph by considering their inner structures described by their BC-trees and SPQR-trees. In [GKM08] embedding constraints that restrict the admissible order of incident edges around a vertex are considered.

A very restrictive constraint is when the input graph G is partially embedded, i.e. when a subgraph H of G is provided with an embedding $\mathcal{H}$. In this case, the problem of determining a planar embedding of the whole graph that extends the embedding $\mathcal{H}$, if one exists, is linear [ADF+10]. Also, if the answer is negative, an obstruction taken from a collection of minimal non-planar instances can be produced in polynomial-time [JKR11].

A constrained planarization is implied anytime an embedding that minimizes some quality measure is desired. As pointed out in [BM90, PT00, Piz05], the quality of a planar embedding can be measured in terms of the maximum distance of its vertices from the external face. Such a distance can be given in terms of different incidence relationships between vertices and faces. For example, if two faces are considered adjacent when they

share a vertex, then the maximum distance to the external face is called *radius* [RS84]. If two vertices are adjacent when they are endpoints of an edge, then the maximum distance to the external face is called *width* [DLT84]. If two vertices are adjacent when they are on the same face and the external face is adjacent to all its vertices, then the maximum distance to the external face is called *outerplanarity* [Bak94]. If two faces are adjacent when they share an edge, then the maximum distance to the external face is called *depth* [BM88]. In [PT00, GM04] algorithm are proposed to minimize the maximum distance of the biconnected components of the graph from the external face, where two biconnected components are adjacent if they share a cut-vertex. This measure, which is also called "depth", is a rougher indicator of the quality of the embedding but can be computed in linear time.

In [BM90], Bienstock and Monma present an algorithm to compute the planar embedding of an n-vertex planar graph with minimum maximum distance to the external face in $O(n^5 \log n)$ time, which is improved to $O(n^4)$ time in [ADP11]. The considered distance is the depth. However, it is possible to compute the radius, the width, and the outerplanarity of a graph by modifying and simplifying the algorithm for the minimum depth, since such distance measures are intrinsically simpler to compute than the depth [BM90]. The complexity bounds for computing such simpler distance measures is improved in [Kam07], where an algorithm that computes the outerplanarity of an n-vertex planar graph in $O(n^2)$ time is described. Simple variations of this algorithm can lead to compute the radius in $O(n^2)$ time and the width in $O(n^3)$ time [Kam07].

1.2.2 Deletion and Partition Problems

Deleting the minimum number of edges in order to obtain a planar graph is called *maximum planar subgraph* and proved to be NP-hard in [GJ79]. Analogously, deleting the minimum number of vertices in order to obtain a planar graph is called *maximum induced planar subgraph* and proved to be NP-hard in [Yan78].

The problem of partitioning the edges of a graph $G = (V, E)$ into k sets $E_1, \ldots, E_k$ in such a way that each graph $G_i = (V, E_i)$, with $i = 1, 2, \ldots k$ is planar is called *graph thickness* and is shown to be NP-hard for $k = 2$ in [Man83].

1.2.3 Upward Planarity

If the input graph G is directed, adding the requirement that the drawing of G is upward, that is that each edge is a curve of increasing y-coordinates, transforms the planarity problem into the upward planarity one, which was shown to be NP-complete in [GT01].

However, Upward Planarity Testing turns out to be polynomial for several families of directed graphs:

1. if the digraph G is outerplanar. This problem was shown to be $O(n^2)$ in [Pap95].
2. If the digraph G is triconnected [BD91, BDLM94].
3. If the digraph G has a fixed embedding. An $O(n^2)$-time algorithm was introduced in [BDLM94]. Linear in the case of embedded outerplanar graphs ([Pap95]).
4. If the digraph G is single-source. The $O(n^2)$-time algorithm described in [HL96] was improved to linear in [BDMT98].

1.2.4 Outerplanarity

Determining if a graph is outerplanar and producing an outerplanar drawing of it is a problem that can be solved independently or by using a planarity algorithm as a subroutine.

In fact, a graph $G = (V, E)$ is outerplanar if and only if it is planar the graph $G'(V', E')$, where $V' = V \cup \{v\}$ and E' is obtained from E by adding an edge (v_i, v) for each vertex $v_i \in V$.

Deleting the minimum number of vertices from a graph in order to make it outerplanar is NP-complete [Yan78].

1.2.5 Simultaneous Planarity

A recent variant of the planarity problem asks for the simultaneous embedding of two graphs on the same set of vertices V. Namely, a *simultaneous embedding* of $G_1 = (V, E_1)$ and $G_2 = (V, E_2)$ consists of two planar drawings Γ_1 and Γ_2 of G_1 and G_2, respectively, such that any vertex $v \in V$ is mapped to the same point in each of the two drawings. When Γ_1 and Γ_2 are required to be straight-line drawings, this problem is called *geometric simultaneous embedding*. When edges common to E_1 and E_2 are required to be represented by the same Jordan curve in Γ_1 and Γ_2 this problem is called *simultaneous embedding with fixed edges* (or SEFE, for short). The above definition can be easily generalized to k graphs $G_i = (V, E_i)$, with $i = 1, 2, \ldots, k$.

Geometric simultaneous embedding turns out to have limited usability, since testing whether two planar graphs admit such an embedding is NP-hard [EBGJ+07] and since a geometric simultaneous embedding does not always exist for two outerplanar graphs [BCD+07], for two trees [GKV09], and even for a tree and a path [AGKN12].

Conversely, for several classes of graphs the computation of a simultaneous embedding with fixed edges, if any, can be performed in polynomial time [EK05, DL07, Fra06, FGJ+08, JS09, ADF+10, HJL10, ADF+11], although the general problem is of unknown complexity.

1.2.6 Clustered Planarity

The user's need of drawing some set of vertices near one to the other naturally leads to the requirement of drawing them inside the same simple closed region of the plane. This target is pursued by clustered planarity where the containment relationship among regions and vertices is described by an arbitrary hierarchy. More formally, a *clustered graph* $C(G, T)$ is a graph G and a rooted tree T whose leaves are the vertices of G. A *c-planar* drawing of $C(G, T)$ is such that G is planarly drawn and each internal node ν of T is a simple closed region $R(\nu)$ such that:

- $R(\nu)$ contains the drawing of the graph $G(\nu)$ induced by the vertices that are leaves of the subtree rooted at ν;
- $R(\nu)$ contains a region $R(\mu)$ if and only if μ is a descendant of ν in T;
- any two regions $R(\nu_1)$ and $R(\nu_2)$ do not intersect if ν_1 is not an ancestor or a descendant of ν_2; and
- an edge e does not cross the boundary of a region $R(\nu)$ more than once.

Restrictions on the c-planarity testing problem that have been considered in the literature include: (i) assuming that each cluster induces a small number of connected components [FCE95b, FCE95a, Dah98, GJL+02, GLS05, CW06, CDF+08, JJKL08]; (ii) considering only *flat* hierarchies, where all clusters different from the root of T are children of the root [CDPP04, DF08]; (iii) focusing on particular families of underlying graphs [CDPP04, CDPP05, JKK+08]; and (iv) fixing the embedding of the underlying graph [DF08, JKK+08].

Although the general problem is of unknown complexity, it has been shown to be polynomial-time solvable in the following cases:

- If the subgraph $G(\nu)$ induced by each cluster ν is connected the clustered graph is called *c-connected*. The algorithm proposed in [FCE95b, FCE95a] is quadratic. Linear-time algorithms are described in [Dah98, CDF$^+$08]. The case when each cluster induces at most two connected components has been investigated in [JJKL08].
- The results [BKM98, Bie98] on "partitioned drawings" of graphs can be interpreted as linear-time c-planarity tests for non-connected flat clustered graphs with exactly two clusters. The same result (flat clustered planarity for non-connected graphs with exactly two clusters) is shown in [HN09] where the problem is modeled as a two-page book embedding.
- Gutwenger et al. presented a polynomial-time algorithm for c-planarity testing for *almost connected* clustered graphs [GJL$^+$02], i.e., graphs for which all nodes corresponding to the non-connected clusters lie on the same path in T starting at the root of T, or graphs in which for each non-connected cluster its parent cluster and all its siblings in T are connected.
- Cortese et al. studied the class of non-connected clustered graphs such that the underlying graph is a cycle and the clusters at the same level of T also form a cycle, where two clusters are considered adjacent if they are incident to the same edge [CDPP04, CDPP05]. The c-planarity testing and embedding problem is linear for this class of graphs.
- Goodrich et al. introduced a polynomial-time algorithm for producing planar drawings of *extrovert* clustered graphs [GLS05], i.e., graphs for which all clusters are connected or extrovert. A cluster μ with parent ν is extrovert if and only if ν is connected and each connected component of μ has a vertex with an edge that is incident to a cluster which is external to ν.
- Jelínková et al. presented a polynomial-time algorithm for testing the c-planarity of "k-rib-Eulerian" graphs [JKK$^+$08]. A graph is *k-rib-Eulerian* if it is Eulerian and it can be obtained from a 3-connected planar graph with k vertices, for some constant k, by replacing some edges with one or more paths in parallel.

1.3 History of Planarity Algorithms

Directly applying Kuratowski's characterization of planar graphs based on subdivisions would yield an exponential-time algorithm while Wagner's characterization based on minors would give a factorial-time algorithm. The first polynomial-time algorithms for planarity are due to Auslander and Parter citeap-igs-61, Goldstein [Gol63], and, independently, Bader [Bad64].

In 1974 Hopcroft and Tarjan [HT74] proposed the first linear-time planarity testing algorithm. This algorithm, also called "path-addition algorithm," starts from a cycle and adds to it one path at a time. However, the algorithm is so complex and difficult to implement that several other contributions followed their breakthrough. For example, about twenty years after [HT74], Mehlhorn and Mutzel [MM96] contributed a paper to clarify how to construct the embedding of a graph that is found to be planar by the original Hopcroft and Tarjan algorithm.

A different approach has its starting point in the algorithm presented by Lempel, Even, and Cederbaum [LEC67]. This algorithm, also called "vertex-addition algorithm," is based on considering the vertices one-by-one, following an *st*-numbering; it has been shown to be implementable in linear time by Booth and Lueker [BL76], while a linear-time algorithm for computing the needed st-numbering was provided in [ET76]. Also in this case, a further

contribution by Chiba, Nishizeki, Abe, and Ozawa [CNAO85] has been needed for showing how to construct an embedding of a graph that is found planar.

A further interesting algorithm [dOR06, de 08, Bra09] is based on a characterization given by de Fraysseix and Rosenstiehl [dR82, dR85] in turn based on intuitions of Liu and Wu [Wu74, Ros80, Liu88, Liu89, Xu89]. For a long time the algorithm has not been fully described in the literature but had a very efficient implementation in the Pigale software library [dO02].

However, although the planarity problem has been carefully studied in the above cited literature, the story of the planarity testing algorithms enumerates several more recent contributions. The motivations behind such relatively new papers are two-fold. On one side, even if the known algorithms are combinatorially elegant, they are quite difficult to understand and to implement. On the other side, the researchers are interested in deepening the relationships between planarity and Depth First Search (DFS). Such relationships are clearly strong but, probably, up to now, not completely understood.

Two recent DFS-based planarity testing algorithms, whose similarities were stressed in [Tho99], are those presented by Shih and Hsu [SH93, SH99, Hsu03] and by Boyer and Myrvold [BM99, BM04].

The Shih-Hsu algorithm replaces biconnected portions of the graph with single nodes, called C-nodes, whose embedding is fixed.

The Boyer and Myrvold algorithm represents embedded biconnected portions of the graph with a data structure that allows the embeddings to be "flipped" in constant time.

1.4 Common Algorithmic Techniques and Tools

In this section we introduce some definitions and common techniques used by the planarity testing algorithms. The most important technique, common to almost all the algorithms, is *Depth First Search*, or DFS in short. DFS is a method for visiting all the vertices of a graph G. It starts from an arbitrarily chosen vertex of G, and continues moving from the current vertex to an adjacent one, as long as unexplored neighbors are found. When the current vertex has no unexplored neighbors, the traversal backtracks to the first vertex with an unexplored adjacent vertex.

The edges by which DFS discovers new vertices of G form a spanning tree T of G, called *Palm Tree*, or *DFS Tree*. The root of T is the vertex at which the traversal started. The edges of T are called *tree edges*, while the remaining edges of G are called *back edges* (or *co-tree edges*).

After performing a DFS traversal, each vertex v of G can be associated with a *DFS index*, $DFS(v)$, that is the order in which v was reached during the DFS visit. The root of T has index one. For a tree edge (u, v), we have that $DFS(u) < DFS(v)$. On the contrary, a back edge is oriented from the end vertex with higher DFS index to the end vertex with lower DFS index. An example DFS is shown in Fig. 1.1.

For each vertex v of G, we can also define two sets of edges, called $B_{in}(v)$ and $B_{out}(v)$. These sets contain respectively the back edges entering and exiting v. Note that each back edge in $B_{in}(v)$ connects v to a descendant in the DFS tree, while each back edge in $B_{out}(v)$ connects v to an ancestor. Given a tree edge $e = (u, v)$, its *returning edges* are those back edges that from a descendant of v (included v itself) go to an ancestor of u different from u itself. At last, the *lowpoint* of a vertex v, denoted by $lowpt(v)$, is the lowest DFS index of an ancestor of v reachable through a back edge from a descendant of v. Analogously, the *highpoint* of a vertex v, denoted by $highpt(v)$, is the highest DFS index of an ancestor of v reachable through a back edge from a descendant of v.

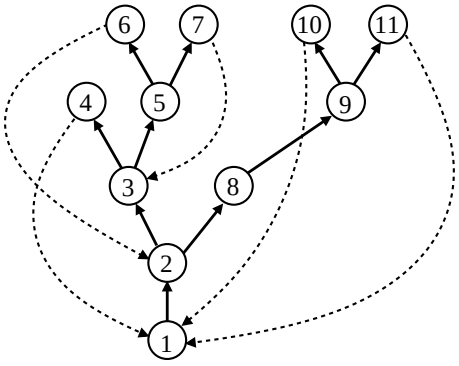

Figure 1.1 A DFS traversal of a graph. Thick lines represent the tree edges, while the back edges are drawn with dashed lines. Each vertex is identified with its DFS index.

1.5 Cycle-Based Algorithms

The shared foundation of all algorithms in this section is an intuitive observation formalized in the Jordan curve theorem: every simple closed curve divides the plane into two connected regions, and hence there is no way to connect two points in both regions without crossing that curve.

Acyclic (undirected) graphs are forests, and therefore planar. If a graph does contain a cycle, that cycle yields a simple closed curve in any planar drawing of it. Consequently, each of the remaining connected parts of the graph needs to be drawn entirely in one of the two connected regions bounded by the cycle. Deciding whether this is possible, and which region to choose, is the essence of planarity testing and embedding, respectively.

It will take three major steps to arrive at simple linear-time algorithms based on this observation. The first step consists in formalizing the approach in a recursive algorithm, the second step yields a linear-time realization of the algorithm, and the third step simplifies the second while adding a corresponding combinatorial characterization.

1.5.1 Adding Segments: The Auslander-Parter Algorithm

Algorithms based on the above cycle criterion were first proposed in [AP61] (see also [Gol63, Bad64, DETT99]).

To introduce the approach formally, consider a simple cycle C in a biconnected graph G. Recall that a graph is planar if and only if its biconnected components are, and that every edge of a biconnected graph is contained in at least one cycle. Each such cycle C yields a collection of connected, edge-induced subgraphs S_i, $i = 1, \ldots, k$ as follows. Either S_i is an edge that connects two vertices of C that are not consecutive (i.e., a *chord*), or S_i is induced by the edges of a connected component of $G \setminus C$ together with the edges connecting that component to C. Each S_i is called a *segment* and, because of biconnectivity, contains at least two vertices of C, referred to as the *attachments* of S_i. Note that vertices of C may be attachments of any number of segments.

A cycle C of G is said to be *separating* if it has at least two segments, while is called *non-separating* otherwise. Of course, if G is a cycle, then C has no pieces and is non-separating. In order to recur on subgraphs, the Auslander-Parter algorithm needs to pick a separating cycle.

LEMMA 1.1 [DETT99] Let G be a biconnected graph, let C be a non-separating cycle of G, and let P be its only segment. If P is not a path, then G has a separating cycle C' consisting of a subpath of C plus a path γ of P between two attachments.

Proof: Let u and v be two attachments of P that are consecutive in the circular ordering of C, let α be a subpath of C between u and v that does not contain any other attachment of P to C, and let β be the subpath of C between u and v different from α. Since P is connected, there is a path γ in P between u and v. Let C' be the cycle obtained from C by replacing α with γ. We have that α is a segment of G with respect to C'. If P is not a path, let e be an edge of P not in γ. There is a segment of C' distinct from α containing e. Therefore, if P is not a path, then C' has at least two segments and is thus a separating cycle of G. □

We have already argued that segments must be drawn entirely in one of the two regions created by the drawing of C. Two segments are said to be *compatible*, if they can be drawn in the same region of C, and *conflicting* otherwise. The following lemma shows that compatibility has a simple characterization.

LEMMA 1.2 Two segments are compatible, if and only if their attachments do not interleave.

The *interlacement graph* of the segments of G with respect to C is the graph whose vertices are the segments of G and whose edges are the pairs of interlacing segments. If there are more than two pairwise incompatible segments, the graph is not planar, because there are only two regions in which they can be drawn. If G is planar, then the interlacement graph is bipartite and two-colorable, each color corresponding to one side of C. We can recursively check the planarity of all subgraphs obtained from the union of a segment S_i and C.

The AUSLANDER-PARTER algorithm is based on the following intuitive recursive characterization of planarity for biconnected graphs.

Theorem 1.1 *[DETT99] A biconnected graph G with a cycle C is planar if and only if the following two conditions hold:*

- *The interlacement graph of the segments of G with respect to C is bipartite.*
- *For each segment P of G with respect to C, the graph obtained by adding P to C is planar.*

Proof: If the graph is planar, it is easy to see that the two conditions hold by considering a planar drawing of it. If the two conditions hold, the proof is by construction and is based on the fact that compatible segments do not interleave (Lemma 1.2) and, hence, can be planarly arranged on the same side of C. □

The algorithm has three cases:

Trivial case. Graph G is a single cycle C. This case can only occur at the beginning of the computation and terminates it.

Base case. Cycle C separates a single segment, which is a path. This terminates the current branch of the computation (there will be no recursion).

Recursive case. A separating cycle C can be found in G. If the interlacement graph

is not bipartite, the algorithm terminates with a non-planarity. Otherwise, recursion is needed on the subgraphs composed by C and each segment.

Here it is not necessary to describe this algorithm in more detail, because, in fact, the subsequent ones are instantiations of this rather generic approach.

It can be shown that the number of recursions is $O(n)$ and that the interlacement graph has size $O(n^2)$, yielding an $O(n^3)$ time algorithm. Also, it is worth mentioning that for a graph that turns out to be planar, the embedding is constructed bottom-up, where planar embeddings may have to be flipped depending on which region they are placed in. There is an interesting alternative approach presented by Demoucron, Malgrange, and Pertuiset [DMP64]. Instead of recursively testing segments for planarity, they start from a fixed embedding of one cycle, and incrementally add only a path connecting two attachments of a segment into a face of the current embedding. This approach requires a careful selection of (facial) cycles and paths and yields a quadratic-time algorithm, but is the only algorithm known to us that does not require alterations of preliminary embeddings.

1.5.2 Adding Paths: The Hopcroft-Tarjan Algorithm

The relative inefficiency of recursively testing augmented segments for planarity is caused by a lack of control over the instances obtained when selecting a cycle.

By exploiting the special structure of DFS trees, Hopcroft and Tarjan [HT74] (see also [Deo76, RND77, Eve79, Wil80]) were able to serialize the combination of trivially planar segments (namely, paths) in a bottom-up fashion.

Let us start from a *spine cycle*, i.e., a fundamental cycle consisting of a path of tree edges starting at the root of the DFS tree together with a single back edge returning to the root. Call the subgraph consisting of only the spine cycle G_0. Next, segments are added recursively one path at a time, which is why the algorithm is often referred to as the *path-addition approach*.

To explain the order in which paths are selected, consider the subgraph G_i consisting of the spine cycle and the first i paths, and an edge e that is incident to but not contained in G_i. Define the segment $S(e)$ of e to be the inclusion-maximal connected subgraph containing e, in which no vertex of G_i has degree larger than one. Moreover, define the vertex with the lowest DFS number in $S(e)$ to be the *lowpoint of the segment*. Since G is biconnected, $S(e)$ contains at least two vertices of G_i, which we call attachments as well. By the order in which paths are inserted, the lowpoint of $S(e)$ will always be an attachment.

Now assume that the DFS tree was pre-built to determine lowpoints and biconnected components. When exploring the tree once again, but this time by traversing edges with lower lowpoints first, we are effectively performing a recursive traversal of segments in which segments with lower lowpoints are traversed first. This order is crucially important for our ability to test efficiently whether segments are conflicting, because it ensures that the attachments of a segment are visited in order of non-decreasing lowpoints. We can therefore place lowpoints on a stack and remove them from the top of the stack during backtracking, thus maintaining in the stack all attachments in the order in which they appear in the lower part of the segment-defining cycle not yet backtracked over. Recall that two segments are compatible if their attachments do not interleave.

Again, we do not go into further details, because the approach is further simplified below. We just note that the algorithm can actually be implemented to run in linear time, but that this is quite difficult and that it took many years until this test was complemented by an embedding phase [MM96] (which is also linear-time).

Part of the difficulty is in the absence of a characterization of planarity that is closely tied to the workings of the algorithm.

1.5.3 Adding Edges: The de Fraysseix-Ossona de Mendez-Rosenstiehl Algorithm

While we have argued that the test of Hopcroft and Tarjan implements that of Auslander and Parter by recursively building up segments one path at a time, it turns out that the original approach can be further simplified by interpreting it on an even more detailed level, adding one edge at a time.

This does not only simplify the algorithm, it also yields a characterization of planarity that provides a less procedural proof of correctness and a straight-forward embedding. Therefore, following the approach of [Bra09], we first recall the characterization and then revisit the algorithm.

Consider a connected undirected graph which needs not to be biconnected, and let $G = (V, T \uplus B)$ be the directed graph obtained from a DFS, where T is the set of tree edges and B the set of back edges. We say that G is a *DFS-orientation* of the original graph. Note that this is not a procedural definition, since such an orientation is characterized by consisting of a rooted spanning tree such that each non-tree edge defines a directed cycle.

DEFINITION 1.1 [dOR06] Let $G = (V, T \uplus B)$ be a DFS-oriented graph. A partition $B = L \uplus R$ of its back edges into two classes, referred to as *left* and *right*, is called *left-right partition*, or *LR partition* for short, if for every vertex v with incoming tree edge e and outgoing edges e_1, e_2

- all return edges of e_1 ending strictly higher than $lowpt(e_2)$ belong to one class and
- all return edges of e_2 ending strictly higher than $lowpt(e_1)$ belong to the other class.

As each back edge returns to an ancestor of its source, it implicitly defines a cycle, which is called *fundamental cycle*. Intuitively, the partition of the back edges into classes L and R corresponds to orienting such fundamental cycles in such a way that those closed by back edges in L are counterclockwise while those closed by back edges in R are clockwise.

Theorem 1.2 *A graph is planar if and only if it admits an LR partition.*

Necessity of the condition of Theorem 1.2 is straightforward: given a DFS tree and a planar embedding of the graph it suffices to assign each back edge to the classes L or R depending on whether the fundamental cycle it closes is counterclockwise or clockwise, respectively. Sufficiency is shown by constructing a planar embedding from a given LR partition. First observe that in an LR partition it can be assumed that all return edges from a tree edge e that return to $lowpt(e)$ are on the same side. Such a LR partition is called *aligned*. If a partition is not aligned an equivalent aligned partition can be found.

In order to obtain a planar embedding the LR partition is extended to cover also outgoing tree edges and, for each vertex v, a linear nesting order is defined on its exiting tree edges. Such an order contains both right and left outgoing edges of v mixed together: restricted to the right outgoing tree edges it gives their clockwise order around v and restricted to the left outgoing tree edges it gives their counterclockwise order around v. The final embedding for each vertex v is obtained by suitably interleaving outgoing tree edges with back edges

entering v.

The extension of the LR partition to tree edges is straightforward. If a tree edge has some return edges (i.e., its source is neither the root nor a cut vertex), it is assigned to the same side as one of its return edges ending at the highest return point. Otherwise, the side is arbitrary.

To determine the linear nesting order for tree edges outgoing v, suppose first that all back edges belong to R and consider a fork consisting of tree edge $e = (u, v)$ and outgoing tree edges e_1 and e_2 exiting v. If both e_1 and e_2 have some return edges, v is a branching point of at least two overlapping fundamental cycles sharing e. Since both cycles are clockwise (all edges belong to R), they must be properly nested in order to avoid edge crossings. As the root of the DFS tree is assumed to be on the outer face, we have to put e_2 clockwise after e_1 (i.e., inside the cycle defined by it) if and only if the lowpoint of e_1 is strictly lower than that of e_2. The same holds if both have the same lowpoint but only e_2 is *chordal*, i.e., has another return point above it. On the contrary, if both L and R are not empty, it can happen that both e_1 and e_2 are chordal. In this case the tie is broken arbitrarily, because in any planar embedding these two edges must be on different sides.

Let $e = (v, w)$ be a tree edge. We denote by $L(e)$ ($R(e)$, respectively) the sequence of incoming back edges entering v from descendants of w ordered in such a way that if $b_1 = (x_1, v)$ and $b_2 = (x_2, v)$ are two such back edges, and if (z, x), (x, y_1), and (x, y_2) is the fork of the two cycles closed by b_1 and b_2, then b_1 comes before b_2 in $L(e)$ ($R(e)$, respectively) if and only if (x, y_1) comes before (x, y_2) ((x, y_1) comes after (x, y_2), respectively) in the adjacency list of x.

DEFINITION 1.2 Given an LR partition and a vertex v, let $e_1^L, \ldots, e_l^L$ be the left outgoing tree edges of v, and $e_1^R, \ldots, e_r^R$ its right outgoing edges. If v is not the root, let u be its parent. The clockwise *left-right ordering*, or *LR ordering* for short, of the edges around v is defined as follows:

$$(u, v), L(e_l^L), e_l^L, R(e_l^L), \ldots, L(e_1^L), e_1^L, R(e_1^L), L(e_1^R), e_1^R, R(e_1^R), \ldots, L(e_r^R), e_r^R, R(e_r^R)$$

where (u, v) is absent if v is the root.

The following lemma shows the sufficiency of the left-right planarity criterion of Theorem 1.2 (the proof by contradiction can be found in [Bra09]).

LEMMA 1.3 Given an LR partition, its LR ordering yields a planar embedding.

Hence, the search for a planar embedding of the input graph boils down to the search for an LR partition of its back edges. Fortunately, from the definition of LR partition directly come two constraints that have to be satisfied by back edges in L and R classes. Let $b_1 = (u_1, v_1)$ and $b_2 = (u_2, v_2)$ be two back edges with overlapping fundamental cycles and let $(u, v), (v, w_1), (v, w_2)$ be their fork.

1. b_1 and b_2 belong to different classes if $lowpt(w_2) < v_1$ and $lowpt(w_1) < v_2$
2. b_1 and b_2 belong to the same class if there is an edge $e' = (x, y)$, with $x \in C(b_1) \cap C(b_2)$ and $y \notin C(b_1) \cap C(b_2)$ such that $lowpt(y) < min\{v_1, v_2\}$

Of course, if a pair of back edges is subject to both the constraints above, no LR partition can exist and hence the graph is non-planar. By exploiting the constraints a quadratic planarity test and embedding algorithm can be immediately found. Namely, build a *constraint*

graph, analogous to the interlacement graph of the AUSLANDER-PARTER algorithm, where each back edge is a vertex and each constraint is an edge, labeled "-1" if the two back edges have to belong to different classes and labeled "+1" if they have to belong to the same class. After contracting "+1" edges, test if the constraint graph is bipartite.

In order to transform this quadratic-time algorithm into a linear one, the constraint graph cannot be explicitly built and the tentative assignment of back edges to the L and R classes may be changed several times during the computation, which is structured as a further traversal of the DFS tree. Details of the linear-time algorithm can be found in [dOR06, de 08, Bra09].

1.6 Vertex Addition Algorithms

Given a planar drawing Γ of a graph $G(V, E)$, we could delete one vertex at a time from Γ to obtain a sequence of smaller planar drawings ending with a single isolated vertex. The intuition that this process could be suitably reversed yields the so-called "vertex addition" algorithms.

We classify in this family the LEMPEL-EVEN-CEDERBAUM, the SHIH-HSU, and the BOYER-MYRVOLD algorithms, although we know that some authors proposed a different classification for their approach. The similarities between the SHIH-HSU and the BOYER-MYRVOLD algorithms were already pointed out in [Tho99], while a common view encompassing all the three algorithms was envisaged by Haeupler and Tarjan in [HT08].

Vertex addition algorithms start from an initial graph G_1 composed by one isolated vertex v_1. At each step $i = 2, \dots n$, a new vertex v_i is added to the graph and the subgraph $G_i(V_i, E_i)$, induced by the current vertices $V_i = \{v_1, \dots, v_i\} \subseteq V$, is considered. Two kinds of operations are performed: first, G_i is checked for planarity; second, some data structures are updated in order to allow analogous checks to be efficiently performed at step $i + 1$.

A key feature, common to this family of algorithms, is that the order in which the vertices are added is not arbitrary. Let $\overline{G}_i(\overline{V}_i, \overline{E}_i)$ be the subgraph of G induced by the vertices $\overline{V}_i = V - V_i$ that have still to be added to the graph. All the algorithms based on vertex addition require that $\overline{G}_i$ is connected for $i = 1, \dots, n$, that is, the vertex addition order is a leaf-to-root order for some spanning tree of G. LEMPEL-EVEN-CEDERBAUM's algorithm, for example, requires that the vertices are added in the order given by an st-numbering; in the SHIH-HSU and in the BOYER-MYRVOLD algorithms the order is that of a reverse DFS traversal of the graph. The importance of this requirement is stated by the following lemma.

LEMMA 1.4 Let $G(V, E)$ be a planar, connected graph and let $\{V_a, V_b\}$ be a bipartition of the vertices in V such that the graph $G_b(V_b, E_b)$ induced by V_b is connected. Consider any planar embedding Γ of G and denote Γ_a the planar embedding Γ restricted to G_a. The following properties hold:

(α) vertices of V_b are on the same face f^* of Γ_a

(β) each face f of Γ_a, with $f \neq f^*$, is also a face of Γ

(γ) if G is biconnected, cut-vertices of G_a are also incident to face f^* of Γ_a

Proof: Property (α) trivially descends from the fact that G_b is connected and Γ is a planar embedding of G. Property (β) is also trivial. Suppose for a contradiction that $f \neq f^*$ is a face of Γ_a but not a face of Γ. Observe that f is a cycle of Γ and, since it is not a face of Γ, it contains at least one edge $e = (u, v)$ of Γ that is not an edge of Γ_a.

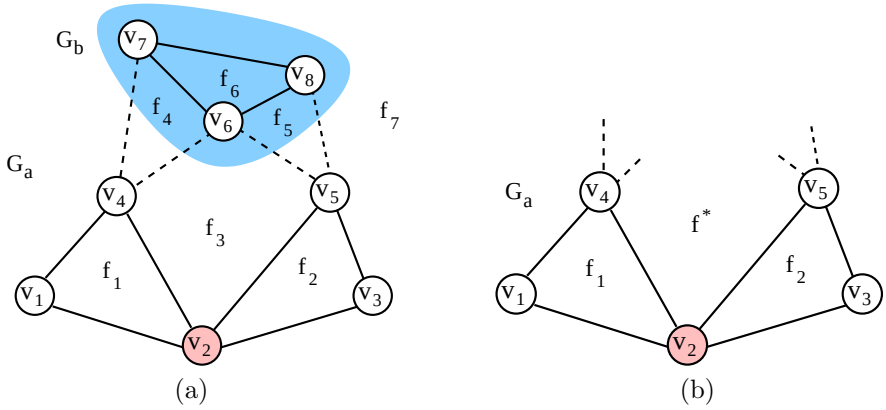

Figure 1.2 Properties of Lemma 1.4. (a) The embedding Γ where the connected subgraph G_b is highlighted. (b) The embedding Γ_a of G_a. By Property 1.4.α, v_6, v_7, and v_8 fall into f^*. By Property 1.4.β, f_1 ad f_2 are also faces of Γ. By Property 1.4.γ, the cut-vertex v_2 is on f^*.

If both u and v belong to V_a, we have a contradiction as e belongs to the graph induced by V_a but it is not in Γ_a. Otherwise, if one among u and v is not in V_a, we have again a contradiction since Property (α) ensures that $f = f^*$. This proves Property (β). Suppose that G is biconnected. If v is a cut-vertex of G_a, then there is a face f of Γ_a that is incident at least two times on v. Since v is not a cut-vertex of Γ, face f is a face of Γ_a but is not a face of Γ, and Property (β) ensures that f^* is the only face of Γ_a that has this property. □

An example showing the three properties of Lemma 1.4 is depicted in Fig. 1.2. Property (α) was also proved in [Eve79, Lemma 8.10] for the special case of connected subgraphs induced by an st-numbering.

Let ψ be a function $\psi : V \to \{1, \ldots, n\}$ that assigns to each vertex of G a different index. We say that ψ is a *proper numbering* of G if for each i we have that the subgraph $\overline{G}_i(\overline{V}_i, \overline{E}_i)$ induced by $\overline{V}_i = \{v | \psi(v) > i\}$ is connected. In order to simplify the notation in the remaining part of this chapter we denote by v_i the vertex for which $\psi(v_i) = i$. Vertex addition algorithms require that vertices are considered in the order imposed by a proper numbering, hence exploiting at each step the properties of Lemma 1.4. Namely, Property (α) guarantees that vertices and edges can be added to a single face f^* of Γ_i, which can be assumed to be the outer face. Property (β) implies that once a vertex or edge is closed inside an internal face of Γ_i it does not need to be considered again (this is a key point to ensure linearity). Finally, Property (γ) justifies the usual assumption, common to most vertex addition algorithms, that G is biconnected.

Properties (α) and (β) lead to the following lemma.

LEMMA 1.5 Let ψ be any proper numbering of a planar, connected graph G. Denote by G_i the subgraph of G induced by vertices in $V_i = \{v | \psi(v) \leq i\}$. There exists a sequence of planar embeddings Γ_i of G_i, with $i = 1, \ldots, n$, such that, for $i = 1, \ldots, n-1$, all internal faces of Γ_i are also internal faces of Γ_{i+1}.

Proof: Let Γ_n be a planar drawing of G with v_n on the external face and let Γ_i, with $i = 1, \ldots, n-1$, be the embeddings of G_i obtained from Γ_n by removing the vertices v_j, with $j = i+1, \ldots, n$. Vertex v_n is on the external face of Γ_n by definition. Since $\overline{G}_i$ is

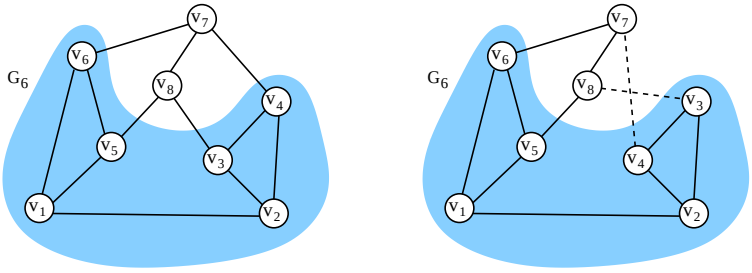

Figure 1.3 A planar graph G with subgraph G_6 highlighted. (a) and (b) show two planar embeddings of G_6, both with the outer vertices of G_6 on the external face. The embedding in (a) is compatible with a planar drawing of G while the embedding in (b) is not.

connected, vertex v_i is also on the external face of Γ_i for any $i = n-1, n-2, \ldots, 1$. Also, $V_a = \{v_1, \ldots, v_{i-1}\}$ and $V_b = \{v_i\}$ is a bipartition of the vertices of G_i of which Γ_i is a planar embedding and $G_b(V_b, \emptyset)$ is trivially connected. Lemma 1.4 applies and by Property (β) we have that all the faces of Γ_{i-1} with the exception of f^* are also faces of Γ_i. Since the external face of Γ_{i-1} is not a face of Γ_i, any other internal face f of Γ_{i-1} is also a face of Γ_i. Finally, as the external face of Γ_i contains v_i, which does not belong to G_{i-1}, face f is an internal face of Γ_i. □

Provided that G is planar, Lemma 1.5 can be exploited for devising an incremental planarity algorithm that, starting from Γ_1, i.e., the trivial embedding of the isolated vertex v_1, computes Γ_i, with $i = 2, \ldots, n$, by adding at each step a vertex v_i on the outer face of Γ_{i-1}, until an embedding Γ_n of the whole graph is produced. Also, Lemma 1.4 provides an indication of what are the properties that these Γ_i should have. Namely, call *outer vertices of* G_i the cut-vertices of G_i and the vertices of G_i adjacent to v_{i+1}, v_{i+2}, ..., v_n. Properties (α) and (γ) of Lemma 1.4 state that if G is biconnected, which can be assumed, each Γ_i necessarily has its outer vertices on the outer face.

Still, computing the sequence of Γ_i, with $i = 1, \ldots, n$, is not an easy task. First, G_i may be not connected. Second, it is easy to see that not any embedding of G_i with its outer vertices on the external face is equivalent to any other. In fact, given a planar graph G, there may exist a planar embedding Γ_i of G_i that has the outer vertices of G_i on the external face but is not obtainable from some planar embedding of G by vertex deletion (Fig. 1.3 provides an example).

Hence, although we know that, starting from any proper numbering ψ of G, the planarity of G implies the existence of a sequence of planar embeddings Γ_i satisfying the conditions of Lemma 1.5, we do not know how to find such a sequence, and choosing a wrong embedding Γ_i along the way would lead to a failure of the whole process even if G is planar. The following lemma comes in help.

LEMMA 1.6 In any planar embedding of a biconnected graph G where vertices $v_1, v_2, \ldots, v_k$ share the same face, they appear in the same circular order up to a reversal.

Proof: The statement is trivial for $k = 2, 3$, since any circular sequence of 2 or 3 labels is equal to any other up to a reversal. Consider two planar embeddings Γ' and Γ'' of G such that vertices $v_1, v_2, \ldots, v_k$, with $k \geq 4$, share the face f' in Γ' and f'' in Γ'' (see Fig. 1.4(a) and 1.4(b) for an example). The proof is based on the trivial observation that a

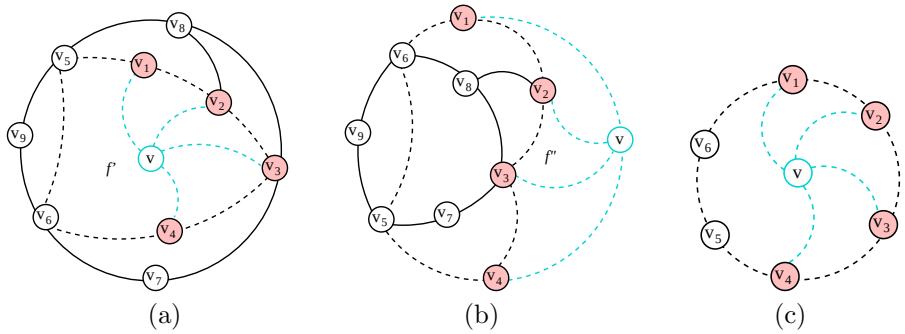

Figure 1.4 (a), (b) Two planar embeddings of a biconnected graph where vertices v_1, v_2, v_3, and v_4 (highlighted in the figure) share the same face. Vertex v is added as in the proof of Lemma 1.6.

dummy vertex v can be inserted into both f' and f'' and planarly connected to $v_1, v_2, \ldots, v_k$. Since G is biconnected, the cycle face f' is simple (see Fig. 1.4(a)). Hence, the subgraph composed by the edges and vertices of f' and v is a wheel (dashed lines in Figs. 1.4(a), 1.4(b), and 1.4(c)) and admits a unique planar embedding up to a reversal. It follows that the circular order of the edges around v is the same in Γ' and in Γ'' up to a reversal. □

Lemma 1.6 applied to each block of G_i is stated in [Eve79, Lemma 8.12] for the special case of subgraphs induced by st-numberings. When iteratively computing a planar embedding for G, the practical use of Lemma 1.6 is that, although in general no definitive choice can be made on the embedding of G_i, something can be said about the embedding of its blocks. Namely, apart from a possible flip, it can be computed an embedding for them that is always compatible with a planar embedding of the whole graph, provided it exists. Surprisingly, this is the only thing that can be safely computed for the embedding of G_i. All the more so, this little amount of information suffices for computing analogous embeddings for the blocks of G_{i+1}, and, since $G_n = G$ is biconnected, at the last step a planar embedding Γ_n of the whole graph is obtained. Finally, the following lemma shows that if the process stops, the graph is not planar.

LEMMA 1.7 Let G be a graph and let ψ be any proper numbering of G. Denote by G_i, with $i = 1, \ldots, n$ the subgraph of G induced by vertices in $V_i = \{v | \psi(v) \leq i\}$ and by $B_i^1, B_i^2, \ldots, B_i^{b_i}$ the b_i blocks of G_i. For a given k, $1 \leq k \leq n-1$, let $\Gamma(B_k^j)$, $1 \leq j \leq b_k$, be arbitrary embeddings of B_k^j with the outer vertices of G_k on their outer faces. If the blocks of G_{k+1} cannot be embedded such that the outer vertices of G_{k+1} are on the outer face and $\Gamma(B_k^j)$, $1 \leq j \leq b_k$, are preserved up to a flip, then G is not planar.

Proof: Suppose for a contradiction that G is planar and that there is no planar embedding for all its blocks B_{k+1}^j, $1 \leq j \leq b_k$, such that the outer vertices of G_{k+1} are on the outer face and the blocks of G_k are embedded, up to a flip, as in $\Gamma(B_k^j)$, $1 \leq j \leq b_k$. Since G is planar, by Lemma 1.5 there is a pair of planar drawings Γ_k^* of G_k and Γ_{k+1}^* of G_{k+1}, both with their outer vertices on the outer face. By Lemma 1.6 the outer vertices of each block of G_k appear in the same order, up to a reversal, both in $\Gamma(B_k^j)$, $1 \leq j \leq b_k$, and in Γ_{k+1}^*. Hence, all embeddings $\Gamma(B_k^j)$ can be inserted into Γ_{k+1}^* yielding a planar embedding for the blocks B_{k+1}^j, $1 \leq j \leq b_k$, such that the outer vertices of G_{k+1} are on the outer face: a

contradiction. □

Lemma 1.7 proves the soundness of the vertex-addition approach. In fact, it shows that iteratively building a planar embedding of the input graph G is not only a sufficient condition for the planarity of G, which is obvious, but also a necessary condition, as G is not planar if one step of the iterative process cannot be accomplished. Usually, in the vertex-addition literature the non-planarity of the input graph in case of failure of the proposed algorithms is proved by a complex case analysis, spread all over the description of the algorithm steps, aimed at identifying a subgraph isomorphic to K_5 or $K_{3,3}$ for each possible cause of failure. Instead, Lemma 1.7 provides a direct proof of the correctness of the approach that avoids the use of Kuratowski's theorem, as claimed in [HT08].

Observe that, since the internal faces of the blocks are preserved in the final embedding of G, at each iterative step of the vertex-addition algorithms the embedded blocks may be flipped and composed together, but they are never inserted one into the other. Hence, all vertex addition algorithms make use of suitable data structures to describe the subgraph G_i that has been explored so far and in particular the embedding of its blocks. These data structures allow for permuting the blocks around the cut-vertices and for flipping the blocks in constant time. In the Lempel-Even-Cederbaum algorithm the data structure is Booth and Lueker's PQ-tree. The Shih-Hsu algorithm uses PC-trees. The Boyer-Myrvold algorithm uses `bicomp` data structure. The purpose of these data structures is analogous: they allow us to flip a portion of the graph (a block) in constant time; they allow us to permute (or to leave undecided) the order of the blocks around a cut-vertex until the blocks are merged together.

1.6.1 The Lempel-Even-Cederbaum Algorithm

The Lempel-Even-Cederbaum algorithm was the first one to exploit the vertex addition paradigm [LEC67] (see also [Eve79, BFNd04]). It is no surprise, therefore, that in order to ease the computation several simplifying assumptions are made. First, but this is usual, the input graph is assumed to be biconnected. Second, the description of the algorithm in [LEC67] only checks the planarity of the input graph, without actually computing a planar embedding if it exists. This gap was closed by Chiba, Nishizeki, Abe, and Ozawa [CNAO85] some decades later. Third, a proper numbering of the vertices of G is required that also ensures that G_i, the graph induced by V_i, is connected. Namely, given any edge (s, t) of a biconnected graph $G(V, E)$ with n vertices an *st-numbering* of G is a function $\psi : V \to \{1, \ldots, n\}$ that assigns to each vertex a different index, such that: (i) $\psi(s) = 1$; (ii) $\psi(t) = n$; and (iii) any vertex except s and t is adjacent both to a lower-numbered and to a higher-numbered vertex. This strong constraint, which implies that both the st-numbering and its reversal are proper numberings, fostered the search for a linear-time algorithm to actually compute an st-numbering of a biconnected graph. Such an algorithm was not known when the approach was introduced (the time complexity of the algorithm used in [LEC67] is $O(nm)$ [ET76]), and was finally found in [ET76].

Working of the algorithm

A *bush* is a single-source connected planar directed graph that admits a planar embedding, called a *bush form*, where all vertices of degree one are on the outer face.

Let G be a biconnected graph G, let ψ be an st-numbering of G, and let G_i be the graph induced by vertices $\{v_1, \ldots, v_i\}$. Graph G can be assumed to be directed, where each edge is oriented from the vertex with the lower value to the vertex with higher value of ψ (see Fig. 1.5(a) for an example). Denote $\mathcal{B}_i$ the graph G_i augmented with the edges of G incident

to the outer vertices of G_i. These edges are called *virtual edges*, while the leaves that they introduce in $\mathcal{B}_i$ are *virtual vertices*. Virtual vertices are labeled with the same indexes they have in G, and multiple instances of the same vertex are kept separate in $\mathcal{B}_i$. Since G_i is determined by an st-numbering, $\mathcal{B}_i$ is connected. Observe that a planar embedding of $\mathcal{B}_i$ with the virtual vertices on the outer face corresponds to a planar embedding of G_i with the outer vertices on the outer face. Hence, if G is planar by Lemma 1.5 $\mathcal{B}_i$ is a bush. See Fig. 1.5 for an example of a graph G_i and the corresponding bush $\mathcal{B}_i$. A bush form $\Gamma_{\mathcal{B}_i}$ is usually represented by drawing all the virtual vertices on the same horizontal line (dashed line of Fig. 1.5(b)).

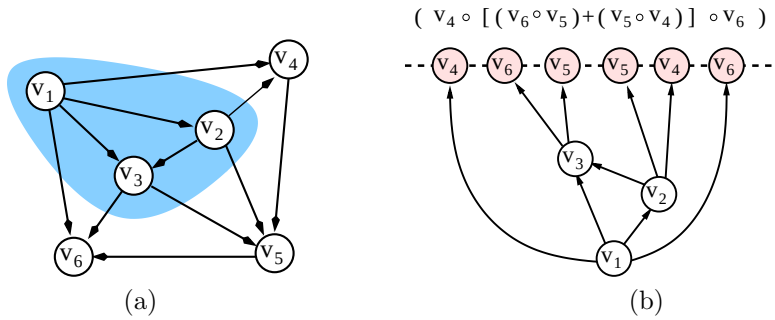

Figure 1.5 (a) A directed planar graph. Labels correspond to an st-numbering of the vertices. The highlighted area is the subgraph G_3 induced by $\{v_1, v_2, v_3\}$. Observe that, due to the st-numbering, both G_3 and $G - G_3$ are connected. (b) The bush form $\mathcal{B}_3$.

Bush form $\Gamma_{\mathcal{B}_i}$ contains a planar embedding of all biconnected components of G_i, and Lemma 1.7 ensures that such embeddings can be kept fixed up to a flip when searching for a planar drawing of G.

The strategy of the algorithm is that of focusing on the virtual vertices of $\mathcal{B}_i$ and of encoding the linear order that they have in $\Gamma_{\mathcal{B}_i}$ into a suitable algebraic expression $\varepsilon(\Gamma_{\mathcal{B}_i})$ that implicitly represents all their permutations compatible with a planar embedding of $\mathcal{B}_i$ with virtual vertices on the outer face.

The definition of $\varepsilon(\Gamma_{\mathcal{B}_i})$ can be inductively provided as follows. Let v be the source of $\Gamma_{\mathcal{B}}$. If $\Gamma_{\mathcal{B}}$ is a trivial bush form consisting of a single directed edge (v, u) then $\varepsilon(\Gamma_{\mathcal{B}}) = u$. Otherwise, if v is a cutvertex splitting $\Gamma_{\mathcal{B}}$ into bush forms $b_1, b_2, \ldots, b_k$. Let $\varepsilon(b_1), \varepsilon(b_2), \ldots, \varepsilon(b_k)$ be the corresponding expressions for $b_1, b_2, \ldots, b_k$. The algebraic expression associated with $\Gamma_{\mathcal{B}}$ is $\varepsilon(\Gamma_{\mathcal{B}}) = (\varepsilon(b_1) \circ \varepsilon(b_2) \circ \ldots \circ \varepsilon(b_k))$. Observe that any permutation of $b_1, b_2, \ldots, b_k$ is compatible with a planar embedding of $\mathcal{B}$. Finally, if v is not a cut vertex of $\Gamma_{\mathcal{B}}$, consider the biconnected component b of $\Gamma_{\mathcal{B}}$ including v and let $u_1, u_2, \ldots, u_k$ be the cut vertices of $\mathcal{B}$ belonging to b. Observe that each subgraph of $\Gamma_{\mathcal{B}}$ routed at u_i, with $i = 1, \ldots, k$, is a bush form b_i. Let $\varepsilon(b_1), \varepsilon(b_2), \ldots, \varepsilon(b_k)$ be the corresponding expressions for $b_1, b_2, \ldots, b_k$. The algebraic expression associated with $\Gamma_{\mathcal{B}}$ is $\varepsilon(\Gamma_{\mathcal{B}}) = [\varepsilon(b_1) + \varepsilon(b_2) + \ldots + \varepsilon(b_k)]$. Observe that flipping the biconnected component b corresponds to flipping the expression $[\varepsilon(b_k) + \varepsilon(b_{k-1}) + \ldots + \varepsilon(b_1)]$.

Figure 1.6 illustrates an example of permutations and flipping in a bush form.

Given a bush form $\Gamma_{\mathcal{B}_i}$, the *reduction* operation changes the embedding of $\mathcal{B}_i$, by permuting bush forms attached to cut vertices and by flipping biconnected components, and produces bush form $\Gamma'_{\mathcal{B}_i}$ where all virtual vertices labeled v_{i+1} are consecutively disposed. If this is not possible, then there is no way of adding vertex v_{i+1} to the embedding while

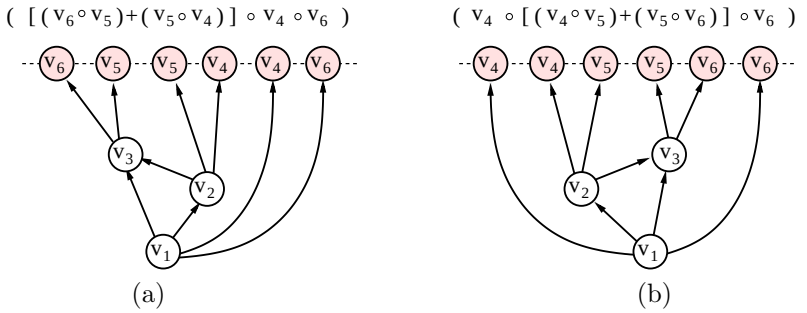

Figure 1.6 (a) A permutation of the bush form of Fig. 1.5(b). (b) A flip of the bush form of Fig. 1.5(b).

keeping all outer vertices of G_{i-1} on the outer face, and by Lemma 1.7 the graph is not planar. If this is possible, then a *substitution* operation is performed on $\Gamma_{\mathcal{B}_i}$, obtaining a drawing $\Gamma_{\mathcal{B}_{i+1}}$. Namely, the virtual vertices labeled v_{i+1} are merged together, and for each edge (v_{i+1}, v_j) exiting v_{i+1} a new virtual vertex v_j is introduced and connected to v_{i+1}.

In the original description of the LEMPEL-EVEN-CEDERBAUM algorithm these operations are not actually performed. Instead, it is shown that the reduction operation on $\Gamma_{\mathcal{B}_i}$ corresponds to an equivalent transformation on $\varepsilon(\Gamma_{\mathcal{B}_i})$ that produces an algebraic expression $\varepsilon(\Gamma'_{\mathcal{B}_i})$ where all the variables v_{i+1} are consecutive. Analogously, the substitution operation corresponds to the removal of the sequence of variables v_{i+1} which are replaced by $(v_{j_1} \circ v_{j_2} \circ \ldots \circ v_{j_k})$, where $v_{j_1}, \ldots, v_{j_k}$ are the vertices directly attached to v_{i+1}.

Data structures

The problem of efficiently identifying the flips and the permutations needed to reduce $\Gamma_{\mathcal{B}_i}$ (or, equivalently, needed to normalize $\varepsilon(\Gamma_{\mathcal{B}_i})$) is solved in [BL76] where the PQ-tree data structure is introduced. Intuitively, a PQ-tree is a data structure corresponding to the syntax tree of the expression $\varepsilon(\Gamma_{\mathcal{B}_i})$. Namely, a *PQ-tree* is a rooted, directed, ordered tree with three types of nodes: P-nodes, Q-nodes, and leaves. For each $\circ$ operation $(\epsilon_1 \circ \epsilon_2 \circ \ldots \circ \epsilon_k)$ in ε the corresponding PQ-tree has a *P-node* with children $PQ(\epsilon_1)$, ... Also, for each $+$ operation $(\epsilon_1 \circ \epsilon_2 \circ \ldots \circ \epsilon_k)$ in ε the corresponding PQ-tree has a *P-node* with children $PQ(\epsilon_1)$, ... The children of a P-node can be arbitrarily permuted, while the order of the children of a Q-node can be reversed. In [BL76] it is shown how a bottom up computation starting from all leaves labeled v_{i+1} is sufficient to compute a sequence of permutations and flips that consecutively disposes all v_{i+1} leaves. Only the smallest subtree containing the v_{i+1} leaves is traversed.

1.6.2 The SHIH-HSU Algorithm

The SHIH-HSU algorithm either constructs a planar embedding of the input graph G or fails and outputs the information that G is not planar [SH93, SH99] (see also [Hsu01, Boy05]). The proper numbering ψ of the vertices of G used by the SHIH-HSU Algorithm is obtained by a DFS traversal of G. Namely, vertices are considered in reverse DFS order, where the root r of the DFS tree has $\psi(r) = n$. Therefore, differently from the LEMPEL-EVEN-CEDERBAUM algorithm, although the graph $\overline{G}_i(\overline{V}_i, \overline{E}_i)$ induced by $\overline{V}_i = \{v | \psi(v) > i\}$ is always connected, the graph $G_i(V_i, E_i)$ induced by vertices in $V_i = \{v | \psi(v) \leq i\}$ is not guaranteed to be connected. At step 1 graph G_1 has vertex v_1 only. At a generic step i, with $i = 2, \ldots, n$, an embedding Γ_{G_i} is obtained from the embedding of $\Gamma_{G_{i-1}}$ by adding

vertex v_i together with all edges connecting it to vertices with lower values of ψ. The strategy used by the SHIH-HSU Algorithm is that of characterizing those configurations that determine a non-planarity, and by giving a recipe to build Γ_{G_i} otherwise.

As G_i is not necessarily connected, at each step i a planar embedding of each connected component of G_i is encoded into a data structure called "PC-tree". A *PC-tree* $\mathcal{T}$ is a rooted, ordered tree with two types of nodes: P-nodes and C-nodes. While the neighbors of a P-node can be arbitrarily permuted, C-nodes come with a cyclic ordering of their adjacency list which can only be reversed. Intuitively P-nodes represent regular nodes of an embedded partial graph, while C-nodes represent biconnected components. Consider a planar embedding of a connected component C of graph G_i such that the outer vertices of C are on its outer face. The PC-tree $\mathcal{T}$ associated with C can be easily obtained from $\mathcal{C}$ by replacing each biconnected component of C with a C-node connected to the outer vertices of it in the same circular order as they appear on the border of the biconnected component. In order to simplify the tree, each C-node representing a trivial biconnected component composed of a single edge connecting two cut-vertices v_j and v_k is replaced with a single edge attached to the two P-nodes corresponding to v_j and v_k. Let r be the *root* of the connected component C, i.e., the vertex of C with higher value of ψ. Observe that r is an outer vertex of C and always corresponds to a P-node of its PC-tree.

The PC-trees associated with G_i represent all the planar embeddings of G_i such that each connected component of G_i has its outer vertices on the outer face. In particular, if G_i is connected, it is apparent the correspondence between its single PC-tree $\mathcal{T}$ and the PQ-tree of G_i used by the LEMPEL-EVEN-CEDERBAUM algorithm, since the former is obtained from the latter by removing leaves and replacing Q-nodes with C-nodes.

Working of the algorithm

At step 1, graph G_1 only has one isolated vertex labeled v_1 and its PQ-tree is a single P-node associated with vertex v_1.

At a generic step i, graph G_{i-1} has been already processed and its PC-trees have been computed. When the new vertex v_i is added, all tree-edges and back edges connecting v_i to G_{i-1} are considered. Suppose that only a tree edge (v_i, u) exits from v_i. In this case only the PC-tree corresponding to the connected component of G_{i-1} needs to be updated. Otherwise, if v_i has more than one child $u_1, u_2, \ldots, u_k$, then v_i is a cut-vertex of G_i; a new P-node is introduced for it and suitably attached to the PC-trees of the connected components $C_1, C_2, \ldots, C_k$ of G_{i-1} containing $u_1, u_2, \ldots, u_k$, respectively, producing a single PC-tree for the new connected component of G_i. Consider a child u of v_i. The PC-tree $\mathcal{T}$ corresponding to the connected component containing u can be attached to v_i in a way that is independent of the PC-trees corresponding to the other children of v_i. Hence, for simplicity of description, we will assume that v_i has a single child u.

Let C be the connected component of G_{i-1} containing u. If no back edge from C attaches to v_i then the P-node introduced for v_i is attached to the P-node representing r in $\mathcal{T}_i$, and step i concludes. Otherwise, suppose some vertex of C has some back edge to v_i. Since the input graph is biconnected, nodes of $\mathcal{T}_{i-1}$ either have $highpt = i$ or have $lowpt > i$, or both. Call *relevant node* each node w of $\mathcal{T}_{i-1}$ such that $highpt(w) = i$ and $lowpt(w) > i$. It is easy to see that the parent of w either is r or is a relevant node in its turn. Hence, relevant nodes form a subtree of the PC-tree rooted at r. By leveraging the relevant nodes subtree it is possible to efficiently check the planarity of G_i and to compute the PC-tree updated with the P-node for u.

Namely, call *terminal nodes* the leaves of the subtree of the PC-tree composed by relevant nodes. We have the following lemma.

LEMMA 1.8 If $\mathcal{T}$ has more than two terminal nodes then G_i (and hence G) is not planar.

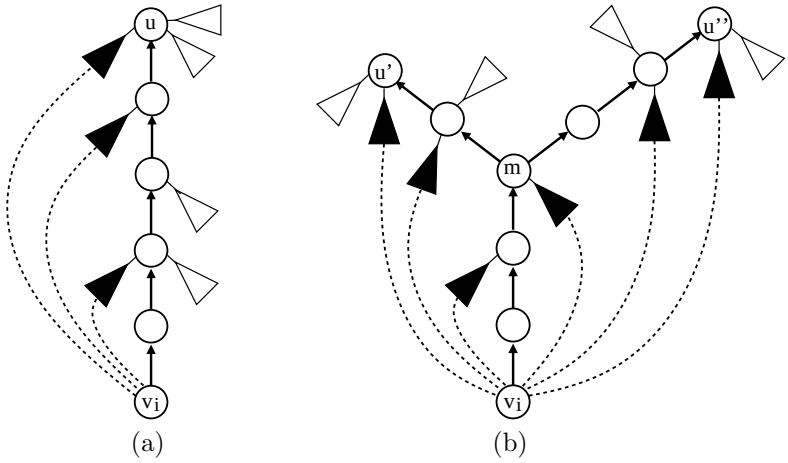

Figure 1.7 (a) Relevant nodes of $\mathcal{T}_{i-1}$ have one terminal. (b) Relevant nodes of $\mathcal{T}_{i-1}$ have two terminals.

Therefore, if G is planar, $\mathcal{T}_{i-1}$ has one or two terminal nodes and the relevant nodes subtree of $\mathcal{T}_{i-1}$ is either a path or a Y-shaped tree, respectively (see Fig. 1.7). Also, observe that an edge exiting a relevant node may be of five different types:

(i) a tree edge to another relevant node;
(ii) a back edge to v_i;
(iii) a tree edge to a subtree whose back edges are all type-(ii) edges; or
(iv) a back edge to a node v_j with $j > i$;
(v) a tree edge to a subtree whose back edges are all type-(iv) edges.

Subtrees attached to edges of type (iii) are called *i-subtrees*, while subtrees attached to edges of type (v) are called *i*-subtrees*. In Fig. 1.7 i-subtrees are represented with black triangles and i*-subtrees with white triangles.

The SHIH-HSU algorithm either identifies a non-planarity or finds a planar arrangement of the back edges to v_i and the i-subtrees to produce a new C-node that represents the block determined by the additions of the back edges to v_i. The algorithm considers four main cases, depending on whether some relevant node is a C-node, and depending on whether $\mathcal{T}_{i-1}$ has one or two terminal nodes.

The easiest case is when $\mathcal{T}_{i-1}$ has exactly one terminal and all the relevant nodes are P-nodes. In fact, in this case all i-subtrees and back edges to i can always be embedded on one side of $\mathcal{T}_{i-1}$, and the embedded part can be replaced with a C-node, as shown in Fig. 1.8(a).

The case when $\mathcal{T}_{i-1}$ has exactly one terminal and some relevant node is a C-node, is analogous, with the difference that the constraints enforced by C-nodes (whose adjacency list can only be flipped) have to be taken into account and may cause a non-planarity whenever, no matter how they are flipped, they force one i-subtree (or back edge to v_i) to be outside the new block or one i*-subtree (or back edge to v_j, with $j > i$) to be inside it.

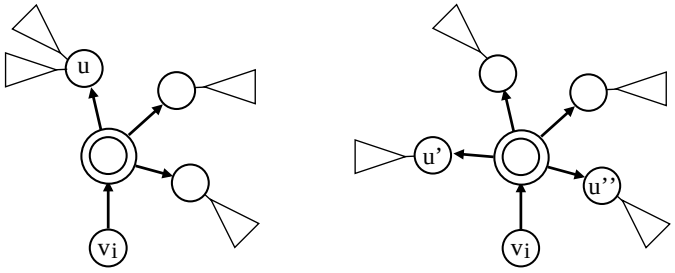

Figure 1.8 (a) The example of Fig. 1.7(a) after contraction. The double border identifies C-nodes. (b) The example of Fig. 1.7(b) after contraction.

The most difficult cases are when $\mathcal{T}_{i-1}$ has two terminal nodes u' and u''. Let m be their common ancestor, P be the unique path in $\mathcal{T}_{i-1}$ from u' to u'', and P' be the path from r to m. If all relevant nodes are P-nodes, then we have the following planarity criterion.

LEMMA 1.9 Graph G_i is planar if and only if any node internal to P' has edges of type (i), (ii), or (iii).

In fact, it is easy to see that if the conditions of Lemma 1.9 are satisfied a new block can be planarly embedded, its border being composed by path P and two paths from the two terminal nodes to v_i and containing all i-subtrees and back edges to v_i. Such a block is replaced by a C-node as shown in Fig. 1.8(b). Again, if some relevant node is a C-node, its constraints on the embedding need to be taken into account and yield a more intricate, although not difficult, case.

Data Structures

A tricky point of the Shih-Hsu algorithm is when a newly identified block has to be replaced with a C-node. To understand why this operation is critical consider that, in order to have a linear-time algorithm, each node of the PC-tree should have a pointer to the parent node. Such a pointer is used, for example, when, starting from the current vertex v_i, its incoming back edges are considered and i-subtrees are traversed moving from child to parent. This operation is needed to identify the relevant node subtree and its terminals. Observe that i*-subtrees cannot be traversed without losing linearity. Also, even identifying them by browsing the adjacency list of a relevant node would have the same result. If the block were naively replaced by a C-node structure as shown in Fig. 1.8 the pointers to the parent of a possibly linear number of children would have to be updated.

Perhaps the easiest way to address this problem is that of encoding the neighborhood of C-nodes with a strategy analogous to that used in [BM99, BM04] which allows us to efficiently traverse in parallel the boundary of a block and it [Hsu01, Hsu03, BFNd04]. A second approach, inspired by the analogous operation on Q-nodes of PQ-trees [LEC67], is that of borrowing the parent pointer from sibling to sibling. The two approaches turn out to be similar, since browsing siblings of a C-node in search for the parent pointer is equivalent to traversing the corresponding block border [Hsu01, Hsu03].

1.6.3 The Boyer-Myrvold Algorithm

The Boyer-Myrvold algorithm [BM99, BM04] (see also [Tho99, BCPD04, HT08]) has several features in common with the Shih-Hsu one, so much that the two have been sometimes identified [Tho99, HT08]. The proper numbering ψ of the vertices of G used by the Boyer-Myrvold algorithm is again a reverse DFS order. The general strategy is that of explicitly maintaining a "flexible" planar embedding of each connected component of G_i with the outer vertices on the outer face. This embedding is "flexible" in the sense that each block can be flipped in constant time, whatever is its size, while the permutation of the blocks around cut-vertices is left undecided. In order to achieve this, each block of G_i is maintained separately from the others in a special structure, and the cut-vertex that has higher value of ψ in one block B, called the *root* of B, has a pointer to the corresponding cut-vertex in the parent block.

Working of the algorithm

The algorithm described in [BM99] was simplified in [BM04]. First, we describe the primitive version in [BM99], which, in our opinion, is more intuitive. Second, we sketch the differences with [BM04].

The computation starts with an initial set of blocks corresponding to the tree edges of the DFS tree of G (see Fig. 1.9). Hence, it could be argued that this is not a vertex-addition algorithm, since all vertices are in place from the first iteration. Actually, a vertex v_j with index higher than the current iteration i is ignored until iteration j is reached. Vertices are considered in reverse DFS order, starting from v_1 and ending with the root v_n of the DFS tree (see Fig. 1.9(a)). If vertex v_i has no incoming back edges, no operation is needed at iteration i.

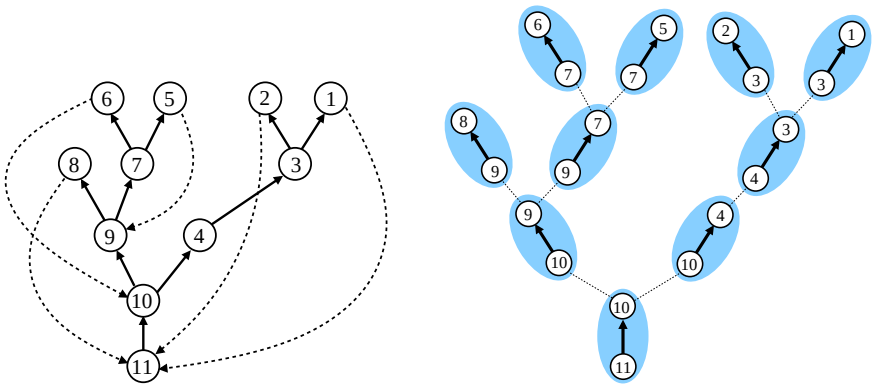

Figure 1.9 The same DFS tree of Fig. 1.1 where vertices are labeled with their reverse DFS index. (b) The Boyer-Myrvold algorithm starts by creating a block for each edge of the tree.

So, for example, a running of the algorithm on the example of Fig. 1.9(b) would not perform any operation at steps $1, 2, \ldots, 8$, as vertices $v_1, v_2, \ldots, v_8$ don't have incoming back edges. Otherwise, if v_i has some incoming back edges, the strategy of the algorithm is that of deciding how to embed them by exploring the borders of the current blocks of G_{i-1}. To give an intuitive example Fig. 1.10(a) represents G_9. At iteration 10 vertex v_{10}

is considered by the algorithm and the back edge (v_6, v_{10}) needs to be embedded. In the embedding choice shown in Fig. 1.10(b) the red path inside the closed face of the new block can be identified in Fig. 1.10(a) as the red path going from v_6 to v_{10} along the borders of the blocks. The approach of BOYER-MYRVOLD algorithm is that of first choosing suitable paths for the back edges returning to v_i and then using such paths to close a new block and update the data structures. Hence, each iteration has two phases: *Path searching* and *Block embedding* (in [BM99, BM04] these phases are called *Walkup* and *Walkdown*, respectively, but the tree is drawn upside down with respect to the convention used here).

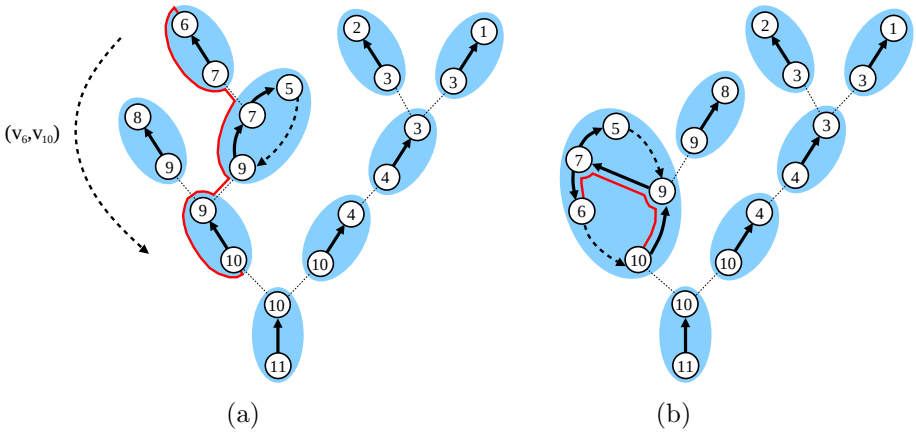

Figure 1.10 The BOYER-MYRVOLD algorithm on the DFS tree of Fig. 1.9(a). (a) When vertex v_{10} is considered the back edge (v_6, v_{10}) needs to be embedded. (b) The embedding of back edge (v_6, v_{10}) corresponding to the choice of the red path from v_6 to v_{10} along the borders of blocks shown in (a).

Let's start from the Path searching phase. Suppose that some back edges enter v_i from vertices $u_1, u_2, \ldots u_k$. For each j, with $j = 1, \ldots, k$, the algorithm searches for a path p_j from u_j to v_i with the following properties:

1. Vertices and edges of p_j are on the boundary of the blocks of G_{i-1}
2. Each vertex of p_j that is the root of a block is followed by the corresponding cut-vertex in the parent block, until v_i is reached
3. Each vertex of p_j which is different from the entry point and the root of the current block is not an outer vertex of G_i (see Fig. 1.11(a))

Also, two paths (i.e. two embedding choices) may be incompatible one with the other. Namely, let p_l and p_m be two paths to v_i and let b be a block b traversed by them. The following compatibility properties are enforced:

1. If p_l and p_m don't share edges of b, they do not share edges in any other block (see Fig. 1.12(a)).
2. Paths p_l and p_m do not share edges of b if they traverse two other distinct outer blocks, where an *outer* block is one containing an outer vertex of G_i (see Fig. 1.12(b)).
3. If p_l and p_m don't share edges of b and the root r_b of b is different from v_i, then r_b is not an outer vertex of G_i. (see Fig. 1.12(c)).

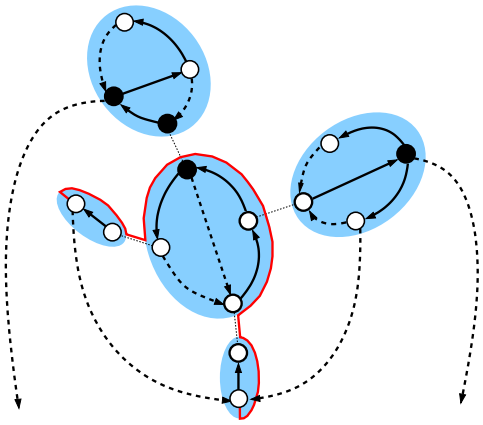

Figure 1.11 Properties of the admissible path to v_i. The lower vertex is the currently processed one while outer vertices of G_i are drawn black. The red path is not admissible, as it traverses an outer vertex of G_i (a cut vertex of G_i in this example).

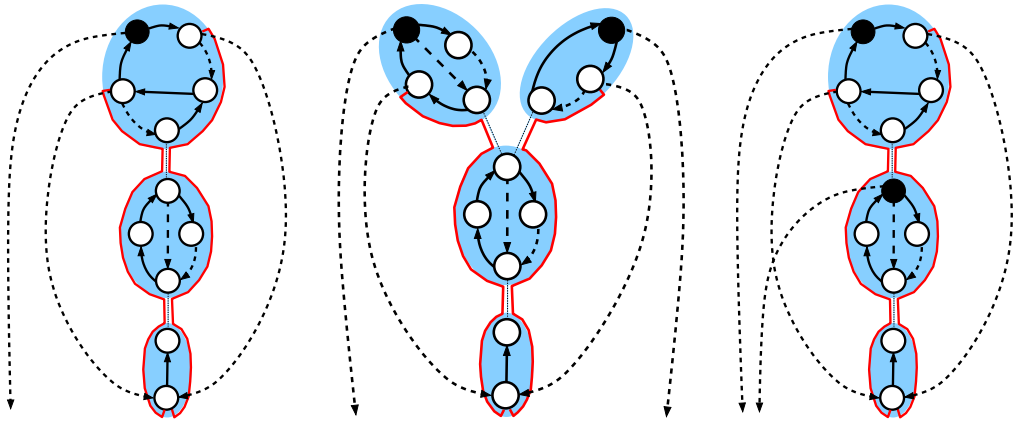

Figure 1.12 Compatibility properties of the paths to v_i. The lower vertex is the currently processed one while outer vertices of G_i are drawn black. (a) Two compatible paths not sharing edges in any block they traverse. (b) Two admissible paths coming from two distinct outer blocks. (c) Two non-admissible paths.

The above properties guarantee that when the new block is closed, no outer vertex of G_i falls inside a face of the block.

In order to be linear, the algorithm does not explicitly compute all the paths p_j, for $j = 1, \ldots, k$. In fact, if two paths share one edge, the second path can follow the same route towards v_1 used by the first one without the need of checking the above properties. Also, whenever a path enters a block b, it searches both the sides of b in parallel, searching for the root r_b of b. In this way the shorter admissible path to r_b is found by exploring at most twice the number of its edges. Since the edges used by the paths will be closed inside some face of the new block, they are never explored again in a subsequent iteration, and the total number of steps required by the algorithm for the computation of such paths is linear.

If the Path searching phase does not detect a non-planarity, the Block embedding phase starts. This is a simpler phase in which, starting from v_i and moving along the boundary

of G_{i-1}, the blocks traversed by the paths are merged together and the back edges are embedded based on their corresponding paths to produce a planar embedding for G_i.

The simplified version of the algorithm described in [BM04] is based on the same two phases, Path searching and Block embedding. However, the check for the paths' compatibility, which in the primitive version were demanded to the first phase are moved to the second phase, which may, therefore, also detect a non-planarity.

Data Structures

The tricky point of the Boyer-Myrvold algorithm is when two blocks, traversed by a path, are merged together. It may happen that a path traverses the child block clockwise and the parent block counterclockwise, or vice-versa. Fortunately, it can be shown that the properties of the paths guarantee that if one path does so all other paths comply with this embedding choice. However, in order to merge the two blocks, one of them needs to be flipped and reversing the adjacency lists of all the vertices of the block may result in a linear-time operation that would yield a quadratic planarity algorithm. In order to solve this problem the authors introduced a suitable data structure, called `bicomp`, that allows us to flip a block in $O(1)$-time, whatever is its size. Such a data structure is based on circular lists that do not have a predefined orientation.

Namely, suppose that the list items of a circular list instead of having the usual `next` and `prev` pointers have two generic pointers `ref1` and `ref2` which could be used arbitrarily to store a reference to the next or previous list item. Suppose, also, that you maintain a reference to the last element encountered while traversing the list. If you want a reference to the next element you compare this reference with `ref1` and `ref2` and choose the one that is different from it. Hence, the circular list is traversed in the direction that is decided by the first step. If the circular order of the list has to be reversed it suffices to begin the traversal in the opposite way.

Of course, if the clockwise direction of the adjacency list of each vertex of a block is independently chosen, this would not necessarily produce a planar embedding. However, it is not difficult to devise some convention to transfer the orientation of the adjacency list of one vertex to the adjacency lists of the neighboring vertices. For example, it may be convened that if in the adjacency list of vertex v_i the list item of vertex v_j uses `ref1` as `next` and `ref2` as `prev`, the same choice is made for the list item of v_i in the adjacency list of v_j (in [BM99, BM04] a less intuitive, but more practical, convention is adopted).

Hence, when two blocks are merged and their common cutvertex is identified the two adjacency lists of the cutvertex can be suitably joined in such a way to implicitly reverse all the adjacency lists of one of the two blocks.

Bibliography

[ADF+10] P. Angelini, G. Di Battista, F. Frati, V. Jelinek, J. Kratochvil, M. Patrignani, and I. Rutter. Testing planarity of partially embedded graphs. In M. Charikar, editor, *Symposium On Discrete Algorithms (SODA '10)*, pages 202–221, 2010.

[ADF+11] P. Angelini, G. Di Battista, F. Frati, M. Patrignani, and I. Rutter. Testing the simultaneous embeddability of two graphs whose intersection is a biconnected graph or a tree. In *Workshop on Combinatorial Algorithms (IWOCA '10)*, volume 6460 of *LNCS*, pages 212–225, 2011.

[ADP11] P. Angelini, G. Di Battista, and M. Patrignani. Finding a minimum-depth embedding of a planar graph in $o(n^4)$ time. *Algorithmica*, 60(4):890–937, 2011.

[AGKN12] P. Angelini, M. Geyer, M. Kaufmann, and D. Neuwirth. On a tree and a path with no geometric simultaneous embedding. *Journal of Graph Algorithms and Applications*, 16(1):37–83, 2012. Special Issue on Selected Papers from GD '10.

[AH77] K. Appel and W. Haken. Every planar map is four colourable, part I: discharging. *Illinois J. Math.*, 21:429–490, 1977.

[AHK77] K. Appel, W. Haken, and J. Koch. Every planar map is four colourable, part II: Reducibility. *Illinois Journal of Mathematics*, 21:491–567, 1977.

[AP61] L. Auslander and S. V. Parter. On imbedding graphs in the sphere. *Journal of Mathematics and Mechanics*, 10(3):517–523, 1961.

[APBL95] D. Archdeacon, C. Paul Bonnington, and C. H. C. Little. An algebraic characterization of planar graphs. *Journal of Graph Theory*, 19(2):237–250, 1995.

[AŠ98] D. Archdeacon and J. Šráň. Characterizing planarity using theta graphs. *Journal of Graph Theory*, 27(1):17–20, 1998.

[Bad64] W. Bader. Das topologische Problem der gedruckten Schaltung und seine Lösung. *Electrical Engineering (Archiv für Elektrotechnik)*, 49(1):2–12, 1964.

[Bak94] B. S. Baker. Approximation algorithms for NP-complete problems on planar graphs. *J. ACM*, 41:153–180, 1994.

[BCD+07] P. Braß, E. Cenek, C. A. Duncan, A. Efrat, C. Erten, D. Ismailescu, S. G. Kobourov, A. Lubiw, and J. S. B. Mitchell. On simultaneous planar graph embeddings. *Computational Geometry*, 36(2):117–130, 2007.

[BCPD04] J. M. Boyer, P. F. Cortese, M. Patrignani, and G. Di Battista. Stop minding your P's and Q's: Implementing a fast and simple DFS-based planarity testing and embedding algorithm. In Giuseppe Liotta, editor, *Graph Drawing (Proc. GD '03)*, volume 2912 of *LNCS*, pages 25–36, 2004.

[BD91] P. Bertolazzi and G. Di Battista. On upward drawing testing of triconnected digraphs. In *Proc. 7th Annu. ACM Sympos. Comput. Geom.*, pages 272–280, 1991.

[BDBD00] P. Bertolazzi, G. Di Battista, and W. Didimo. Computing orthogonal drawings with the minimum number of bends. *IEEE Transaction on Computers*, 49:826–840, August 2000.

[BDLM94] P. Bertolazzi, G. Di Battista, G. Liotta, and C. Mannino. Upward drawings of triconnected digraphs. *Algorithmica*, 6(12):476–497, 1994.

[BDMT98] P. Bertolazzi, G. Di Battista, C. Mannino, and R. Tamassia. Optimal upward

0-8493-8597-0/01/$0.00+$1.50

planarity testing of single-source digraphs. *SIAM J. Comput.*, 27(1):132–169, 1998.

[BFNd04] J. Boyer, C. Fernandes, A. Noma, and J. de Pina. Lempel, Even, and Cederbaum planarity method. In Celso Ribeiro and Simone Martins, editors, *Experimental and Efficient Algorithms*, volume 3059 of *LNCS*, pages 129–144. Springer, 2004.

[Bie98] T. C. Biedl. Drawing planar partitions III: Two constrained embedding problems. Technical Report RRR 13-98, RUTCOR Rutgen University, 1998.

[BKM98] T. C. Biedl, M. Kaufmann, and P. Mutzel. Drawing planar partitions II: HH-Drawings. In J. Hromkovic and O. Sýkora, editors, *Workshop on Graph-Theoretic Concepts in Computer Science (WG '98)*, volume 1517, pages 124–136. Springer, 1998.

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

[BM88] D. Bienstock and C. L. Monma. On the complexity of covering vertices by faces in a planar graph. *SIAM Journal on Computing*, 17:53–76, 1988.

[BM90] D. Bienstock and C. L. Monma. On the complexity of embedding planar graphs to minimize certain distance measures. *Algorithmica*, 5(1):93–109, 1990.

[BM99] J. Boyer and W. Myrvold. Stop minding your P's and Q's: A simplified O(n) planar embedding algorithm. In *10th Annual ACM-SIAM Symposium on Discrete Algorithms*, volume 1027 of *LNCS*, pages 140–146. Springer-Verlag, 1999.

[BM04] J. Boyer and W. Myrvold. On the cutting edge: Simplified O(n) planarity by edge addition. *Journal of Graph Algorithms and Applications*, 8(3):241–273, 2004.

[Boy05] J. Boyer. Additional PC-tree planarity conditions. In J. Pach, editor, *Graph Drawing*, volume 3383 of *LNCS*, pages 82–88. Springer, 2005.

[Bra09] U. Brandes. The left-right planarity test. Manuscript submitted for publication, 2009.

[CDF+08] P. F. Cortese, G. Di Battista, F. Frati, M. Patrignani, and M. Pizzonia. C-planarity of c-connected clustered graphs. *Journal of Graph Algorithms and Applications*, 12(2):225–262, Nov 2008.

[CDPP04] P. F. Cortese, G. Di Battista, M. Patrignani, and M. Pizzonia. Clustering cycles into cycles of clusters. In János Pach, editor, *Proc. Graph Drawing 2004 (GD '04)*, volume 3383 of *LNCS*, pages 100–110. Springer, 2004.

[CDPP05] P. F. Cortese, G. Di Battista, M. Patrignani, and M. Pizzonia. Clustering cycles into cycles of clusters. *Journal of Graph Algorithms and Applications, Special Issue on the 2004 Symposium on Graph Drawing, GD '04*, 9(3):391–413, 2005.

[Che81] C.C. Chen. On a characterization of planar graphs. *Bulletin of the Australian Mathematical Society*, 24:289–294, 1981.

[CMS08] M. Chimani, P. Mutzel, and J. M. Schmidt. Efficient extraction of multiple kuratowski subdivisions. In Seok-Hee Hong, Takao Nishizeki, and Wu Quan, editors, *Graph Drawing (GD 2007)*, volume 4875 of *LNCS*, pages 159–170. Springer, 2008.

[CNAO85] N. Chiba, T. Nishizeki, S. Abe, and T. Ozawa. A linear algorithm for embedding planar graphs using PQ-trees. *J. Comput. Syst. Sci.*, 30(1):54–76, 1985.

[Col90] Y. Colin de Verdière. Sur un nouvel invariant des graphes et un critère de

planarité. *Journal of Combinatorial Theory, Series B*, 50(1):11–21, 1990.

[Col91] Y. Colin de Verdière. On a new graph invariant and a criterion for planarity. In Neil Robertson and Paul D. Seymour, editors, *Graph Structure Theory*, volume 147 of *Contemporary Mathematics*, pages 137–148. American Mathematical Society, 1991.

[CW06] S. Cornelsen and D. Wagner. Completely connected clustered graphs. *Journal of Discrete Algorithms*, 4(2):313–323, 2006.

[Dah98] E. Dahlhaus. Linear time algorithm to recognize clustered planar graphs and its parallelization. In Claudio L. Lucchesi and Arnaldo V. Moura, editors, *Proc. Latin American Theoretical INformatics (LATIN '98)*, volume 1380 of *LNCS*, pages 239–248. Springer, 1998.

[DBTV01] G. Di Battista, R. Tamassia, and L. Vismara. Incremental convex planarity testing. *Information Computation*, 169:94–126, August 2001.

[de 08] H. de Fraysseix. Trémaux trees and planarity. *Electronic Notes in Discrete Mathematics*, 31:169–180, 2008.

[Deo76] N. Deo. Note on Hopcroft and Tarjan planarity algorithm. *Journal of the Association for Computing Machinery*, 23:74–75, 1976.

[DETT99] G. Di Battista, P. Eades, R. Tamassia, and I. G. Tollis. *Graph Drawing*. Prentice Hall, Upper Saddle River, NJ, 1999.

[DF08] G. Di Battista and F. Frati. Efficient c-planarity testing for embedded flat clustered graphs with small faces. In Seok-Hee Hong, Takao Nishizeki, and Wu Quan, editors, *Proc. Graph Drawing 2007 (GD '07)*, volume 4875 of *LNCS*, pages 291–302. Springer, 2008.

[DKT09] Z. Dvorak, K. Kawarabayashi, and R. Thomas. Three-coloring triangle-free planar graphs in linear time. In Claire Mathieu, editor, *SODA*, pages 1176–1182. SIAM, 2009.

[DL07] E. Di Giacomo and G. Liotta. Simultaneous embedding of outerplanar graphs, paths, and cycles. *Int. J. Computational Geometry and Applications*, 17(2):139–160, 2007.

[DLT84] D. Dolev, F. T. Leighton, and H. Trickey. Planar embedding of planar graphs. In Franco P. Preparata, editor, *VLSI Theory*, volume 2 of *Adv. Comput. Res.*, pages 147–161. JAI Press, Greenwich, Conn., 1984.

[DMP64] G. Demoucron, Y. Malgrange, and R. Pertuiset. Graphes planaires: Reconnaissance et construction des représentations planaires topologiques. *Revue Fran cais de Recherche Opérationelle*, 8:33–47, 1964.

[dO96] H. de Fraysseix and P. Ossona de Mendez. Planarity and edge poset dimension. *European Journal of Combinatorics*, 17(8):731–740, 1996.

[dO02] H. de Fraysseix and P. Ossona de Mendez. P.I.G.A.L.E - Public Implementation of a Graph Algorithm Library and Editor, 2002. SourceForge project page `http://pigale.sourceforge.net/` (GPL License).

[dOR06] H. de Fraysseix, P. Ossona de Mendez, and P. Rosenstiehl. Trémaux trees and planarity. *International Journal of Foundations of Computer Science*, 17(5):1017–1029, 2006.

[dR82] H. de Fraysseix and P. Rosenstiehl. A depth-first characterization of planarity. *Annals of Discrete Mathematics*, 13:75–80, 1982.

[dR85] H. de Fraysseix and P. Rosenstiehl. A characterization of planar graphs by Trémaux orders. *Combinatorica*, 5(2):127–135, 1985.

[DT89] G. Di Battista and R. Tamassia. Incremental planarity testing. In *Proc. 30th Annu. IEEE Sympos. Found. Comput. Sci.*, pages 436–441, 1989.

[DT96a] G. Di Battista and R. Tamassia. On-line maintenance of triconnected compo-

nents with SPQR-trees. *Algorithmica*, 15:302–318, 1996.

[DT96b] G. Di Battista and R. Tamassia. On-line planarity testing. *SIAM J. Comput.*, 25:956–997, 1996.

[EBGJ+07] A. Estrella-Balderrama, E. Gassner, M. Jünger, M. Percan, M. Schaefer, and M. Schulz. Simultaneous geometric graph embeddings. In S. H. Hong, T. Nishizeki, and W. Quan, editors, *Graph Drawing (GD '07)*, volume 4875 of *LNCS*, pages 280–290, 2007.

[EK05] C. Erten and S. G. Kobourov. Simultaneous embedding of planar graphs with few bends. *Journal of Graph Algorithms and Applications*, 9(3):347–364, 2005.

[ET76] S. Even and R. E. Tarjan. Computing an st-numbering. *Theoret. Comput. Sci.*, 2:339–344, 1976.

[Eve79] S. Even. *Graph Algorithms*. Computer Science Press, Potomac, Maryland, 1979.

[FCE95a] Q. W. Feng, R. F. Cohen, and P. Eades. How to draw a planar clustered graph. In Ding-Zhu Du and Ming Li, editors, *Proc. Computing and Combinatorics (COCOON '95)*, volume 959 of *LNCS*, pages 21–30. Springer, 1995.

[FCE95b] Q. W. Feng, R. F. Cohen, and P. Eades. Planarity for clustered graphs. In *Proc. European Symposium on Algorithms (ESA '95)*, volume 979 of *LNCS*, pages 213–226. Springer, 1995.

[FGJ+08] J. J. Fowler, C. Gutwenger, M. Jünger, P. Mutzel, and M. Schulz. An SPQR-tree approach to decide special cases of simultaneous embedding with fixed edges. In I. G. Tollis and M. Patrignani, editors, *Graph Drawing (GD '08)*, volume 5417 of *LNCS*, pages 157–168, 2008.

[Fra06] F. Frati. Embedding graphs simultaneously with fixed edges. In M. Kaufmann and D. Wagner, editors, *Graph Drawing (GD '06)*, volume 4372 of *LNCS*, pages 108–113, 2006.

[GIS99] Z. Galil, G. F. Italiano, and N. Sarnak. Fully dynamic planarity testing with applications. *Journal of the Association for Computing Machinery*, 46:28–91, January 1999.

[GJ79] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, New York, NY, 1979.

[GJL+02] C. Gutwenger, M. Jünger, S. Leipert, P. Mutzel, M. Percan, and R. Weiskircher. Advances in C-planarity testing of clustered graphs. In Stephen G. Kobourov and Michael T. Goodrich, editors, *Proc. Graph Drawing 2002 (GD '02)*, volume 2528 of *LNCS*, pages 220–235. Springer, 2002.

[GJS76] M. R. Garey, D. S. Johnson, and L. Stockmeyer. Some simplified NP-complete graph problems. *Theoretical Computer Science*, 1(3):237–267, 1976.

[GKM08] C. Gutwenger, K. Klein, and P. Mutzel. Planarity testing and optimal edge insertion with embedding constraints. *Journal of Graph Algorithms and Applications*, 12(1):73–95, 2008.

[GKV09] M. Geyer, M. Kaufmann, and I. Vrt'o. Two trees which are self-intersecting when drawn simultaneously. *Discrete Mathematics*, 307(4):1909–1916, 2009.

[GLS05] M. T. Goodrich, G. S. Lueker, and J. Z. Sun. C-planarity of extrovert clustered graphs. In P. Healy and N.S. Nikolov, editors, *Proc. Graph Drawing 2005 (GD '05)*, volume 3843 of *LNCS*, pages 211–222. Springer, 2005.

[GM01] C. Gutwenger and P. Mutzel. A linear time implementation of SPQR-trees. In Joe Marks, editor, *Graph Drawing (GD 2000)*, volume 1984 of *LNCS*, pages 77–90. Springer, 2001.

[GM04] C. Gutwenger and P. Mutzel. Graph embedding with minimum depth and

maximum external face. In Giuseppe Liotta, editor, *Graph Drawing*, volume 2912 of *LNCS*, pages 259–272. Springer, 2004.

[GMW01] C. Gutwenger, P. Mutzel, and R. Weiskircher. Inserting an edge into a planar graph. In *Proceedings of the twelfth annual ACM-SIAM symposium on Discrete algorithms*, SODA '01, pages 246–255, Philadelphia, PA, USA, 2001. Society for Industrial and Applied Mathematics.

[Gol63] A. J. Goldstein. An efficient and constructive algorithm for testing whether a graph can be embedded in the plane. In Jr. Edmonds, John R., editor, *Graphs and Combinatorics Conference*, Technical Report, page 2 unn. pp. Princeton University, 1963.

[Grö59] H. Grötzsch. Ein dreifarbensatz fü dreikreisfreie netze auf der kugel. Wiss. Z. Martin-Luther-Univ. Halle-Wittenberg Math.-Natur. Reihe 8, 1959.

[GT01] A. Garg and R. Tamassia. On the computational complexity of upward and rectilinear planarity testing. *SIAM J. Comput.*, 31(2):601–625, 2001.

[Hal43] D. W. Hall. A note on primitive skew curves. *Bulletin of the American Mathematical Society*, 49(2):935–936, 1943.

[Har69] F. Harary. *Graph Theory*. Addison-Wesley, Reading, Mass., 1969.

[HJL10] B. Haeupler, K. R. Jampani, and A. Lubiw. Testing simultaneous planarity when the common graph is 2-connected. In *Proceedings of the 21st Symposium on Algorithms and Computation (ISAAC'10)*, volume 6507 of *LNCS*, pages 410–421. Springer Heidelberg/Berlin, 2010.

[HL96] M. D. Hutton and A. Lubiw. Upward planar drawing of single-source acyclic digraphs. *SIAM J. Comput.*, 25(2):291–311, 1996.

[HN09] S. H. Hong and H. Nagamochi. Two-page book embedding and clustered graph planarity. Technical Report 2009-004, Department of Applied Mathematics & Physics, Kyoto University, 2009.

[Hsu01] W. L. Hsu. PC-trees vs. PQ-trees. In *Proceedings of the 7th Annual International Conference on Computing and Combinatorics*, COCOON '01, pages 207–217, London, UK, 2001. Springer-Verlag.

[Hsu03] W. L. Hsu. An efficient implementation fo the PC-Tree algorithm of Shih and Hsu's planarity test. Technical Report TR-IIS-03-015, Inst. of Inf. Science, Academia Sinica, 2003.

[HT65] F. Haray and W. T. Tutte. A dual form of Kuratowski's theorem. *Canad. Math. Bull.*, 8:17–20, 1965.

[HT73] J. Hopcroft and R. E. Tarjan. Dividing a graph into triconnected components. *SIAM J. Comput.*, 2(3):135–158, 1973.

[HT74] J. Hopcroft and R. E. Tarjan. Efficient planarity testing. *J. ACM*, 21(4):549–568, 1974.

[HT08] B. Haeupler and R. E. Tarjan. Planarity algorithms via pq-trees (extended abstract). *Electronic Notes in Discrete Mathematics*, 31:143–149, 2008.

[HW74] J. E. Hopcroft and J. K. Wong. Linear time algorithm for isomorphism of planar graphs (preliminary report). In *Proceedings of the sixth annual ACM symposium on Theory of computing*, STOC '74, pages 172–184, New York, NY, USA, 1974. ACM.

[JJKL08] V. Jelinek, E. Jelinkova, J. Kratochvil, and B. Lidicky. Clustered planarity: Embedded clustered graphs with two-component clusters. In *GD '08*, volume 5417 of *LNCS*, pages 121–132, 2008.

[JKK+08] E. Jelínková, J. Kára, J. Kratochvíl, M. Pergel, O. Suchý, and T. Vyskocil. Clustered planarity: Small clusters in Eulerian graphs. In Seok-Hee Hong, Takao Nishizeki, and Wu Quan, editors, *Proc. Graph Drawing 2007 (GD*

'07), volume 4875 of *LNCS*, pages 303–314. Springer, 2008.

[JKR11] V. Jelínek, J. Kratochvíl, and I. Rutter. A kuratowski-type theorem for planarity of partially embedded graphs. In *Proceedings of the 27th annual ACM symposium on Computational geometry*, SoCG '11, pages 107–116, New York, NY, USA, 2011. ACM.

[JS09] M. Jünger and M. Schulz. Intersection graphs in simultaneous embedding with fixed edges. *Journal of Graph Algorithms and Applications*, 13(2):205–218, 2009.

[Kam07] F. Kammer. Determining the smallest k such that g is k -outerplanar. In L. Arge, M. Hoffmann, and E. Welzl, editors, *ESA '07*, volume 4698 of *LNCS*, pages 359–370, 2007.

[Kel93] A. K. Kelmans. Graph planarity and related topics. In Neil Robertson and Paul Seymour, editors, *Graph Structure Theory, Proceedings of the AMS-IMS-SIAM Joint Summer Research Conference on Graph Minors, 1991*, volume 147 of *Contemporary Mathematics*, pages 635–667, 1993.

[KR88] P. N. Klein and J. H. Reif. An efficient parallel algorithm for planarity. *J. Comput. Syst. Sci.*, 37(2):190–246, 1988.

[Kur30] K. Kuratowski. Sur le problème des courbes gauches en topologie. *Fund. Math.*, 15:271–283, 1930.

[LEC67] A. Lempel, S. Even, and I. Cederbaum. An algorithm for planarity testing of graphs. In *Theory of Graphs: Internat. Symposium (Rome 1966)*, pages 215–232, New York, 1967. Gordon and Breach.

[LH77] C. H. C. Little and D. A. Holton. A new characterization of planar graphs. *Bulletin of the American Mathematical Society*, 83(1):137–138, 1977.

[Lie01] A. Liebers. Planarizing graphs – a survey and annotated bibliography. *Journal of Graph Algorithms and Applications*, 5(1):1–74, 2001.

[Liu88] Y. Liu. A new approach to the linearity of testing planarity of graphs. *Acta Mathematicae Applicatae Sinica (English Series)*, 4(3):257–265, 1988.

[Liu89] Y. Liu. Boolean approach to planar embeddings of a graph. *Acta Mathematica Sinica (New Series)*, 5(1):64–79, 1989.

[LS10] C. H. C. Little and G. Sanjith. Another characterisation of planar graphs. *The Electronic Journal of Combinatorics*, 17(15), 2010.

[LT79] R. J. Lipton and R. E. Tarjan. A separator theorem for planar graphs. *SIAM J. Appl. Math.*, 36:177–189, 1979.

[Mac37a] S. MacLane. A combinatorial condition for planar graphs. *Fundamenta Mathematicae*, 28:22–32, 1937.

[Mac37b] S. MacLane. A structural characterization of planar combinatorial graphs. *Duke Mathematical Journal*, 3:466–472, 1937.

[Man83] A. Mansfield. Determining the thickness of graphs is NP-hard. *Proc. Math. Cambridge Philos. Soc.*, 93:9–23, 1983.

[MM96] K. Mehlhorn and P. Mutzel. On the embedding phase of the Hopcroft and Tarjan planarity testing algorithm. *Algorithmica*, 16:233–242, 1996.

[MW99] P. Mutzel and R. Weiskircher. Optimizing over all combinatorial embeddings of a planar graph. In *Proceedings of the 7th International IPCO Conference on Integer Programming and Combinatorial Optimization*, pages 361–376, London, UK, 1999. Springer-Verlag.

[MW00] P. Mutzel and R. Weiskircher. Computing optimal embeddings for planar graphs. In *Proceedings of the 6th Annual International Conference on Computing and Combinatorics*, COCOON '00, pages 95–104, London, UK, 2000. Springer-Verlag.

[Pap95] A. Papakostas. Upward planarity testing of outerplanar dags. In R. Tamassia and I. G. Tollis, editors, *Graph Drawing (Proc. GD '94)*, volume 894 of *Lecture Notes Comput. Sci.*, pages 298–306. Springer-Verlag, 1995.

[Piz05] M. Pizzonia. Minimum depth graph embeddings and quality of the drawings: An experimental analysis. In P. Healy and N.S. Nikolov, editors, *Graph Drawing '05*, volume 3843 of *LNCS*, pages 397–408, 2005.

[PT00] M. Pizzonia and R. Tamassia. Minimum depth graph embedding. In M. Paterson, editor, *ESA '00*, volume 1879 of *LNCS*, pages 356–367, 2000.

[RND77] E. M. Reingold, J. Nievergelt, and N. Deo. *Combinatorial Algorithms: Theory and Practice*. Prentice Hall, Englewood Cliffs, NJ, 1977.

[Ros80] P. Rosenstiehl. Preuve algébrique du critère de planarité du Wu-Liu. *Annals of Discrete Mathematics*, 9:67–78, 1980.

[RR89] V. Ramachandran and J. H. Reif. An optimal parallel algorithm for graph planarity. In *Proc. 30th Annu. IEEE Sympos. Found. Comput. Sci.*, pages 282–293, 1989.

[RR94] V. Ramachandran and J. Reif. Planarity testing in parallel. *Journal of Computer and System Sciences*, 49:517–561, December 1994.

[RS84] N. Robertson and P. D. Seymour. Graph minors. III. Planar tree-width. *Journal on Combinatorial Theory, Series B*, 36(1):49–64, 1984.

[RSST97] N. Robertson, D.P. Sanders, P.D. Seymour, and R. Thomas. The four color theorem. *J. Combin. Theory Ser. B*, 70:2–4, 1997.

[Sch89] W. Schnyder. Planar graphs and poset dimension. *Order*, 5:323–343, 1989.

[SH93] W.K. Shih and W.L. Hsu. A simple test for planar graphs. In *Int. Workshop on Discrete Math. and Algorithms*, pages 110–122, 1993.

[SH99] W.K. Shih and W.L. Hsu. A new planarity test. *Theor. Comp. Sci.*, 223, 1999.

[Tam98] R. Tamassia. Constraints in graph drawing algorithms. *Constraints*, 3:87–120, April 1998.

[Tar72] R. E. Tarjan. Depth-first search and linear graph algorithms. *SIAM J. Comput.*, 1(2):146–160, 1972.

[Tho99] R. Thomas. Graph planarity and related topics. In Jan Kratochvíl, editor, *Graph Drawing (Proc. GD '99)*, volume 1731 of *LNCS*, pages 137–144. Springer-Verlag, 1999.

[TT97] Hisao Tamaki and Takeshi Tokuyama. A characterization of planar graphs by pseudo-line arrangements. In *Proc. 8th Annu. Internat. Sympos. Algorithms Comput.*, volume 1350 of *Lecture Notes Comput. Sci.*, pages 123–132. Springer-Verlag, 1997.

[Wag37a] K. Wagner. Über eine Eigenschaft der ebenen Komplexe. *Mathematische Annalen*, 114:570–590, 1937.

[Wag37b] K. Wagner. Über eine Erweiterung eines Satzes von Kuratowski. *Deutsche Mathematik*, 2:280–285, 1937.

[Whi32] H. Whitney. Non-separable and planar graphs. *Transactions of the American Mathematical Society*, 34:339–362, 1932.

[Wil80] S. G. Williamson. Embedding graphs in the plane – algorithmic aspects. *Annals of Discrete Mathematics*, 6:349–384, 1980.

[Wu74] W. Wu. Planar embedding of linear graphs. *Kexue Tongbao*, 2:226–282, 1974. (In Chinese).

[Xu89] W. Xu. Improved algorithm for planarity testing based on Wu-Liu's citerion. *Annals of the New York Academy of Science*, 576:641–652, 1989.

[Yan78] M. Yannakakis. Node-and edge-deletion NP-complete problems. In *Proceed-*

ings of the tenth annual ACM symposium on Theory of computing, STOC '78, pages 253–264, New York, NY, USA, 1978. ACM.

[Yan82] M. Yannakakis. The complexity of the partial order dimension problem. *SIAM J. Algebraic Discrete Methods*, 3(3):351–358, 1982.

Index