Stop Minding Your P's and Q's: Implementing a Fast and Simple DFS-Based Planarity Testing and Embedding Algorithm*

John M. Boyer[1], Pier Francesco Cortese[2], Maurizio Patrignani[2], and Giuseppe Di Battista[2]

[1] PureEdge Solutions Inc. Victoria, BC Canada
jboyer@acm.org
[2] Università di Roma Tre, Italy
{cortese,patrigna,gdb}@dia.uniroma3.it

Abstract. In this paper we give a new description of the planarity testing and embedding algorithm presented by Boyer and Myrvold [2], providing, in our opinion, new insights on the combinatorial foundations of the algorithm. Especially, we give a detailed illustration of a fundamental phase of the algorithm, called walk-up, which was only succinctly illustrated in [2]. Also, we present an implementation of the algorithm and extensively test its efficiency against the most popular implementations of planarity testing algorithms. Further, as a side effect of the test activity, we propose a general overview of the state of the art (restricted to efficiency issues) of the planarity testing and embedding field.

1 Introduction

Testing the planarity of a graph is one of the most fascinating and intriguing problems of the graph drawing and of the graph algorithms fields.

In 1974 Hopcroft and Tarjan [10] 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 [10], Mehlhorn and Mutzel [13] 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 [12]. This algorithm, also called "vertex addition

* Work partially supported by European Commission - Fet Open project COSIN – COevolution and Self-organisation In dynamical Networks – IST-2001-33555, by "Progetto ALINWEB: Algoritmica per Internet e per il Web", MIUR Programmi di Ricerca Scientifica di Rilevante Interesse Nazionale, and by "The Multichannel Adaptive Information Systems (MAIS) Project", MIUR Fondo per gli Investimenti della Ricerca di Base.

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 [1]. Also in this case, a further contribution by Chiba, Nishizeki, Abe, and Ozawa [5] has been needed for showing how to construct an embedding of a graph that is found planar.

A further interesting algorithm is based on a characterization given by de Fraysseix and Rosenstiehl [6] and, although it has not been fully described in the literature, it has a very efficient implementation in the Pigale software library [7].

However, the story of the planarity testing algorithms enumerates several more recent contributions, aimed at further exploring the relationships between planarity and Depth First Search (DFS) and at devising algorithms that are easier to understand and to implement.

Two recent DFS-based planarity testing algorithms are those presented by Shih and Hsu [15,16,11] and by Boyer and Myrvold [2]. Shih and Hsu algorithm replaces biconnected portions of the graph with single nodes whose embedding is fixed. Boyer and Myrvold algorithm, which is the starting point of this paper, represents embedded biconnected portions of the graph with a data structure that allows the embeddings to be "flipped" in constant time. More recently, in an unpublished manuscript, Boyer and Myrvold [3] presented an algorithm that, although inspired by [2], constitutes, in our opinion, a new and original approach to the planarity testing problem.

The contributions of the present paper can be summarized as follows. In Section 3 we describe the algorithm of [2] in a new way, that is, in our opinion, easily readable and more suitable for an implementation. In particular, in Section 4 we give a detailed description of a fundamental phase of the algorithm, called walk-up, which is essential for a correct and efficient implementation and was only succinctly illustrated in [2]. Finally, in Section 5 we present an experimental analysis which provides a general overview of the state of the art (restricted to efficiency issues) of the planarity testing and embedding field.

2 Background

We assume that the reader is familiar with graph terminology and basic properties of planar graphs. Unless otherwise specified, we only consider graphs without self-loops and multiple edges, as they can be replaced with paths of length two without affecting planarity.

A *cutvertex* of a graph G is such that its removal increases the number of connected components of G. A connected graph is said to be *biconnected* if it has no cutvertices. The *biconnected components* of a connected graph (also called *bicomps*) are its maximal biconnected subgraphs. A bicomp is said to be *elementary* if it is formed by a single edge. Note that a cutvertex belongs to several bicomps, while each edge falls into exactly one bicomp of the graph.

A *Depth First Search* (*DFS* in short) is a technique for visiting all the vertices of a graph. Each vertex v is assigned a *DFS index*, denoted $DFS(v)$, which specifies the order in which it was reached by the DFS visit, starting from the root r which has $DFS(r) = 1$. The edges used by the DFS visit to move from one vertex to the next one are called *tree-edges* and form a spanning tree of G, called the *DFS tree*. The remaining edges are *back-edges*. The *ancestors* of a vertex v are the vertices in the unique chain of tree-edges from v to the root r. If a vertex u is an ancestor of v, then v is a *descendant* of u. A tree-edge links a *parent* vertex to a *child* vertex, the former (latter) being the one with lowest (highest) DFS-index, while a back-edge is thought to be oriented exiting the descendant and entering the ancestor. The *lowpoint* of a vertex v, denoted by $Lowpt(v)$, is the lower DFS index of an ancestor of v reachable through a back-edge from a descendant of v. The set of back-edges entering a vertex v is denoted $B_{in}(v)$, while the set of back-edges exiting v is denoted $B_{out}(v)$. For a back-edge e, we call *support of* e and denote by $S(e)$ the unique chain of tree-edges having the same endpoints as the back-edge e. The support of e and e form a cycle.

A *planar drawing* of a graph is such that no two edges intersect (except at common endpoints). A graph is *planar* if it admits a planar drawing. A planar drawing partitions the plane into topologically connected regions, called *faces*. The unbounded face is called the *external face*. The *boundary* of a face is its delimiting circuit. The *incidence list* of a vertex v is the set of edges incident upon v. A planar drawing determines a circular ordering on the incidence list of each vertex v according to the clockwise sequence of the incident edges around v. Two planar drawings of the same connected graph G are *equivalent* if they determine the same circular orderings of the incidence lists. Two equivalent planar drawings have the same face boundaries. A *planar embedding* or simply *embedding* of G is an equivalence class of planar drawings and is described by circularly sorted incidence lists for each vertex v. In the following, unless otherwise specified, we will consider the embeddings comprehensive of the specification of the external face. In [2] a suitable data structure is introduced which can be used to describe an embedded bicomp. This data structure allows to "flip" the bicomp (that is, to represent the embedding in which all the adjacency lists of the vertices have been reversed) in constant time.

Since a graph is planar iff its bicomps are planar, in the following section we assume that the graph to be processed is biconnected. A simple method for handling graphs that are not biconnected is based on the property, which is easy to prove, that a graph G is planar iff it is planar the graph G' obtained from G by merging two adjacent bicomps b_1 and b_2, sharing a cutvertex v, with the addition of an edge between two vertices adjacent to v, one belonging to b_1 and the other to b_2. Based on this property, all the bicomps of the graph G can be merged with dummy edges without affecting planarity and the dummy edges can be removed after the embedding is computed.

3 The Boyer and Myrvold Planarity Testing and Embedding Algorithm

In this section we give a new description of the Boyer and Myrvold planarity testing algorithm [2]. This algorithm tests the planarity of a biconnected graph in linear time by trying to build a planar embedding of it and consists of three steps: *Preprocessing*, *Embedding Construction*, and *Embedding Reporting*.

In Step *Preprocessing* a DFS is performed on the input graph G. During the visit a DFS tree is constructed. For each vertex v we compute $DFS(v)$, $Lowpt(v)$, the set $B_{in}(v)$, and the list $L(v)$ of the children of v in the DFS tree sorted by their lowpoint values. Also, we compute $H(v)$, that is the lowest $DFS(w)$, with w in $B_{out}(v)$. Roughly, $H(v)$ is the DFS index of the vertex "nearest" to the root, reachable from v with a back-edge. If $B_{out}(v)$ is empty, then $H(v)$ is set to an "infinite" value. Further, each edge of the DFS tree is associated with a data structure representing an (embedded) elementary bicomp.

Step *Embedding Construction* is more complex and is based on visiting one-by-one the vertices of the input graph in inverse DFS-index order. Observe that this corresponds to a post-order traversal performed on the DFS tree.

Let v be the current visited vertex. Two strictly related tasks, called *walk-up* and *walk-down* (both described below), are performed starting from v. Let $G(v)$ be the subgraph of G composed by the edges of the DFS tree augmented with the edges in $B_{in}(w)$, for each w such that $DFS(w) \geq v$. The target of the two tasks is to determine (if it is possible) a planar embedding for the bicomps of $G(v)$ exploiting the planar embeddings already computed for graph $G(u)$, where $DFS(u) = DFS(v) + 1$, which are preserved up to a "flip" operation. See Fig. 1.

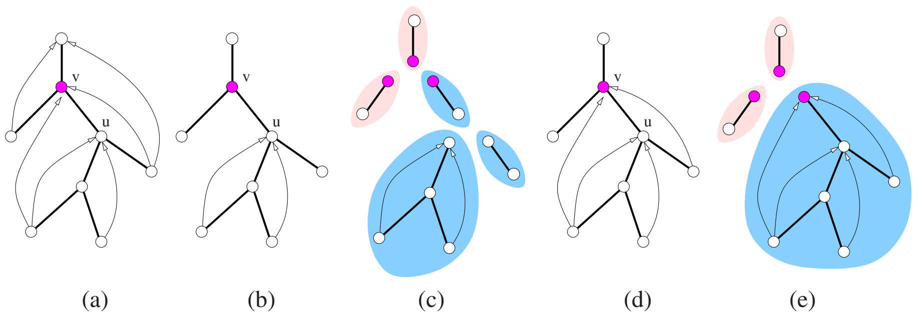

Fig. 1. (a) A graph G to be planarized; the edges of the DFS tree are drawn thick. $DFS(u) = DFS(v) + 1$. (b) Graph $G(u)$. (c) An embedding of the bicomps of $G(u)$. (d) Graph $G(v)$. (e) An embedding of the bicomps of $G(v)$.

In order to make this possible, the embeddings of the bicomps of $G(v)$, computed when vertex v is processed, must have the outer vertices of $G(v)$ on the external face. We call *outer vertices* of $G(v)$ the cutvertices of $G(v)$ and the vertices of $G(v)$ incident to a back-edge that is not in $G(v)$. Observe that: (i) $G(u)$ is a subgraph $G(v)$. (ii) The edges belonging to $G(v)$ that do not belong to $G(u)$ are exactly the back-edges in $B_{in}(v)$. (iii) Since the input graph G is biconnected, when $DFS(v) = 1$, $G(v) = G$ has a single bicomp, and a planar embedding of $G(v)$, if one exists, is a planar embedding for G.

In the *Embedding Reporting* step, the embedding is copied to a target data structure. The rest of the section shows the walk-up and the walk-down phases.

The purpose of the walk-up phase is to verify if there is a way to add the back-edges in $B_{in}(v)$ to the embedded bicomps of $G(u)$ in such a way that the embedded bicomps of $G(v)$ have the outer vertices of $G(v)$ on the external face. The walk-up phase works as follows. For each edge $e = (w, v)$ in $B_{in}(v)$ a path p_e is searched (and marked) from w to v along the bicomps that contain an edge in $S(e)$. The paths must satisfy several constraints. In order to describe these constraints, we need to introduce some definitions. Consider an embedded bicomp b of $G(u)$ containing some edges of $S(e)$. Observe that the edges of $S(e)$ contained in b form a subpath of $S(e)$. This subpath has its endpoints on the external face of b. We call these two vertices *root of* b and *entry point of* b with respect to $S(e)$, the former (latter) being the one nearest to (farthest from) the root of the DFS tree.

Given a specific $S(e)$, two *borders* $\mathcal{A}$ and $\mathcal{B}$ can be identified from $v_{b,e}$ to r_b along the boundary of the external face of b. If b is an elementary bicomp, the two "sides" of the edge $(v_{b,e}, r_b)$ are considered as distinct borders. A vertex w in a bicomp b different from the root r_b is defined to be *externally active for* b if there is a path from w to an ancestor of the currently being processed vertex v, that is only composed by a (possibly empty) sequence of tree edges not belonging to b plus a single back-edge.

Two constraints, called α and β, must be enforced on each path p_e. (α) For each bicomp b that contains edges in $S(e)$ the path p_e must contain either the border $\mathcal{A}$ or $\mathcal{B}$ from the entry point $v_{b,e}$ to the root r_b of b. (β) A path p_e must not contain a border that has an inner vertex (i.e. a vertex other than $v_{b,e}$ and r_b) that is externally active. Three more constraints, γ, δ, and ϵ, are expressed with respect to each pair of paths p_{e_1} and p_{e_2} traversing the same bicomp b. We say that the border of b used by p_{e_1} *intersects* the one used by p_{e_2} if they share an edge (or if they share the same "side" of the sole edge of the elementary bicomp b). Non-intersecting paths can only share entry or exit vertices (or both) in the bicomps they both traverse. (γ) If the border of b used by p_{e_1} does not intersect the border of b used by p_{e_2}, the two paths must use non-intersecting borders in all the bicomps they both traverse. (δ) Paths p_{e_1} and p_{e_2} must not use intersecting borders of b if they traverse two other distinct bicomps that are externally active. We call a bicomp *externally active* if it contains a (non-root) externally active vertex. (ϵ) If the border of b used by p_{e_1} does not intersect the border of b used by p_{e_2} and the root r_b of b is different from v, then r_b must not be an outer vertex of $G(v)$.

If no set of paths are found satisfying the constraints, then G is not planar [2], otherwise, they are used by the walk-down phase to build the embeddings of the bicomps of $G(v)$. Actually, the information used by the walk-down phase is simpler than the paths themselves. It only consists of the set of the borders of the bicomps used by the paths (that is, the marked borders) and, for each vertex w that was a walk-up entry point for a bicomp, of two sets of vertices called *active roots* and *non-active roots* of w. The set of the active roots of w

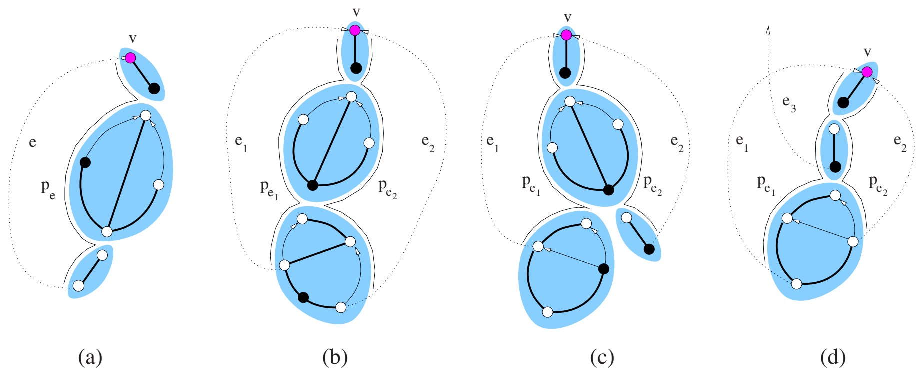

Fig. 2. Four different configurations for illustrating Constraints β, γ, δ, and ϵ. Edges in $B_{in}(v)$ are drawn with dotted lines, while externally active vertices are drawn black. (a) An example of a path that does not satisfy constraint β. (b), (c) Examples of pairs of paths satisfying constraints γ and δ, respectively. (d) An example of a pair of paths that do not satisfy constraint ϵ.

contains the roots (at most two) of the externally active bicomps traversed by the paths immediately before entering w. The set of the non-active roots of w contains the roots of the non externally active bicomps traversed by the paths immediately before entering w.

Consider two back-edges e_1 and e_2 in $B_{in}(v)$. If the intersection of $S(e_1)$ and $S(e_2)$ is not void, we say that e_1 and e_2 are *interleaving*. It is easy to prove that the interleaving relation is an equivalence relation. Roughly, the edges of the same equivalence class have the same child of v as an internal vertex of their supports. For each equivalence class induced by the interleaving relation, the walk-down process starts on a marked border $\mathcal{A}$ of the elementary bicomp containing v by pushing v onto a stack. When the stack becomes void an analogous process is performed on the opposite border $\mathcal{B}$, except if v is the DFS tree root. Let w be the current vertex on top of the stack. The back-edge e from w to v (if any) is added to the embedding of b_B in such a way to close the border $\mathcal{A}$ in the internal part of the cycle formed by e and p_e. Then, the stack is updated by: (i) removing w; (ii) pushing onto the stack the next marked vertex x along the border adjacent to w provided that the set of active roots of w is empty; (iii) pushing onto the stack one of the active roots of w (if any); (iv) pushing onto the stack all the non-active roots of w. When the current vertex on top of the stack is found to be the root vertex of some bicomp b, then b is merged into the bicomp b_B, flipping its embedding if necessary so that a marked border of b attaches to the just processed border of b_B.

4 Selecting Paths: Implementation Issues for the Walk-up

In order to describe the operations performed during the walk-up phase we need to extend to root vertices the definition given in Section 3 for external activity of non-root vertices. We define a root vertex r_b of a bicomp b *externally active* if it exists a back-edge from r_b to the currently being processed vertex v or if it exists a bicomp $b' \neq b$ which is externally active and such that $r_{b'} = r_b$. Also, we call a vertex w on a bicomp b ($w \neq r_b$) *pertinent for* b if it is an entry point for b and if it is not externally active for b.

Due to space reasons, some technicalities needed to detect in constant time the external activity of vertices and bicomps and the pertinence of vertices are left out from this camera ready and can be found in [4].

During the walk-up phase, back-edges in $B_{in}(v)$ are considered one-by-one in arbitrary order. The operations performed for each back-edge e in $B_{in}(v)$, called *walk-up*(e), have the purpose of marking the borders used by p_e and of updating, if it is needed, the lists of active roots and non-active roots of the entry point $v_{b,e}$ of each bicomp b traversed by p_e. Exceptionally, walk-up(e) may reverse the decision taken by a previous walk-up(e') about which border $\mathcal{A}$ or $\mathcal{B}$ of a bicomp b is used by $p_{e'}$. Walk-up(e), where $e = (w, v)$, starts at vertex w and selects a path to v that passes through each bicomp b traversed by $S(e)$. It terminates if (i) vertex v is reached, (ii) a vertex u, marked by a previous walk-up(e') is reached and the subpath of p_e from u to v is chosen to be equal to the analogous subpath of $p_{e'}$, (iii) a non-planarity condition has been detected. On each bicomp b traversed by $S(e)$, walk-up(e) proceeds from the entry point $v_{b,e}$ along both borders, $\mathcal{A}$ and $\mathcal{B}$, in parallel, searching for the root r_b of the bicomp or for a marked vertex. Borders blocked by externally active vertices can not be used. When the root r_b is reached, the fully explored border is selected and marked, while the partial exploration of the opposite border is abandoned, and walk-up(e) "ascends" to the next bicomp traversed by $S(e)$, that is, starts a parallel exploration of its borders. When walk-up(e) begins on the first bicomp traversed by $S(e)$, it is said to be *internally active.* It changes to *externally active* when it starts a parallel exploration of the borders of a bicomp after ascending from an externally active one. In order to be able to mark the borders of a bicomp b, we associate with each vertex u on the external face of b two *pathInfo* integers, one for each edge incident to u on the external face of b. During Step Preprocessing, pathInfo integers are assigned a value greater than n, where n is the number of vertices of the input graph. The flags are considered to be *clear* if their absolute value exceeds $DFS(v)$, where v is the currently being processed vertex, while are considered to be *set* if their absolute value is equal to $DFS(v)$. Thus, when the algorithm moves to processing the next vertex, which has lower DFS index, the pathInfo flags are implicitly cleared. While walk-up(e) is internally active, it sets the pathInfo integers to $-V$, where $V = DFS(v)$. Once walk-up(e) becomes externally active, it sets pathInfo integers to $+V$.

The following is a case-by-case statement of the decision points: first we give six rules for an internally active walk-up, then we provide the analogous six rules for an externally active walk-up.

Rule 1. If an internally active walk-up(e) enters $v_{b,e}$ and either pathInfo integer is set, then walk-up(e) terminates (the remaining part of p_e being already marked properly by a prior walk-up).

Rule 2. If an internally active walk-up(e) enters $v_{b,e}$ and both pathInfo integers are not set, then walk-up(e) traverses both borders $\mathcal{A}$ and $\mathcal{B}$ in search of the root r_b. (a) If a vertex w with a pathInfo set to $-V$ is encountered, then set the pathInfo to $-V$ along the border traversed from $v_{b,e}$ to w and terminate walk-up(e). (b) If a border is blocked (i.e. if it has an internal vertex that is externally active), then the opposing border must be taken to reach the root r_b. (c) If the second border is also blocked, then the graph is non-planar.

Rule 3. If an internally active walk-up(e) reaches vertex v then it terminates, and the pathInfo of the border used to reach v are set to $-V$.

Rule 4. If an internally active walk-up(e) traverses border $\mathcal{A}$ to reach the bicomp root r_b, then at least one pathInfo integer of r_b must be clear. If both are clear, then walk-up(e) simply sets the pathInfo to $-V$ along $\mathcal{A}$ and then it ascends to the next bicomp traversed by $S(e)$.

Rule 5. If an internally active walk-up(e) traverses border $\mathcal{A}$ to reach the bicomp root r_b, and the pathInfo of r_b for the opposing border $\mathcal{B}$ is set to $-V$, then: (a) If $\mathcal{B}$ has no internal vertex that is externally active, then walk-up(e) must select $\mathcal{B}$ and set to $-V$ the pathInfo integers along $\mathcal{B}$ (until an edge marked $-V$ is reached). Then, walk-up(e) can terminate. (b) If $\mathcal{B}$ contains only one non-root externally active vertex $w \neq v_{b,e}$, and if $\mathcal{B}$ contains no pertinent vertices between w and r_b (endpoints excluded), then the prior $-V$ marking of the path in $\mathcal{B}$ from w to r_b can be reversed to be a $-V$ path from $(w, \ldots, v_{e,b}, \ldots, r_b)$. The pathInfo of the path in $\mathcal{B}$ from w to r_b can be cleared and walk-up(e) terminated. (c) If $\mathcal{B}$ has an internal vertex w that is externally active, but either $v_{e,b}$ is externally active or $\mathcal{B}$ contains an internal vertex other than w that is pertinent or externally active, then the prior $-V$ pathInfo markings in $\mathcal{B}$ cannot be reversed. The current walk-up must select the border $\mathcal{A}$ and set its pathInfo to $-V$. Since r_b has been approached from both sides, the graph is non-planar if r_b is externally active. Otherwise, walk-up(e) ascends to the next bicomp traversed by $S(e)$ (where it becomes an externally active walk-up).

Rule 6. If an internally active walk-up(e) traverses border $\mathcal{A}$ to reach the bicomp root r_b, and the pathInfo of r_b for the opposing border $\mathcal{B}$ is set to $+V$, then we must test whether the prior externally active walk-up's path selection must be reversed. (a) Let w denote the vertex with a $+V$ pathInfo setting that is most distant from r_b in $\mathcal{B}$. If $\mathcal{B}$ contains any internal vertex other than w that is pertinent and has a $+V$ pathInfo setting, then there is no need to reverse the prior $+V$ path selection since the back-edge (w, v) that will be added during walk-down will close a pertinent vertex in an interior face of the embedding regardless of which border is selected. (b) If the path $(v_{b,e}, \ldots, w)$ in $\mathcal{B}$ contains an externally active vertex x (other than w), then u blocks the prior $+V$ path selection from being reversed. (c) If the prior $+V$ path cannot be reversed, then the current walk-up simply sets the pathInfo to $-V$ on the border $\mathcal{A}$. If r_b is externally active, then walk-up(e) terminates because the graph is non-planar. Otherwise, walk-up(e) ascends to the next bicomp traversed by $S(e)$ (where it becomes an externally active walk-up). (d) If the prior $+V$ path selection can be reversed, then the $+V$ pathInfo integers in $\mathcal{B}$ are cleared, while the pathInfo integers along the path $(w, \ldots, v_{b,e}, \ldots, r_b)$ are set to $+V$. Then, walk-up(e) can be terminated (the remaining part of p_e being already marked properly by a prior walk-up)

An externally active walk-up(e) must follow the six rules given below.

Rule 7. If the entry point $v_{b,e}$ of an externally active walk-up(e) has both pathInfo integers set to $+V$, then this is the third externally active walk-up to enter $v_{b,e}$ and the graph is non-planar. Note that a single prior walk-up cannot have passed through $v_{b,e}$ setting both of its pathInfo integers to $+V$ because $v_{b,e}$ is externally active for bicomp b.

Rule 8. If the entry point $v_{b,e}$ of an externally active walk-up(e) has one pathInfo integer set to $+V$, then walk-up(e) must select the opposing border $\mathcal{A}$ to the root r_b. (a) If $\mathcal{A}$ is blocked by an externally active vertex, then the graph is non-planar. (b) If $\mathcal{A}$ is not blocked, then walk-up(e) sets its pathInfo integer to $+V$ up to r_b. If r_b is externally active, then the graph is non-planar. Otherwise, walk-up(e) ascends to the next bicomp traversed by $S(e)$.

Rule 9. The case in which an externally active walk-up(e) enters a vertex $v_{b,e}$ that has both its pathInfo integers set to $-V$ can not occur. In fact, prior internally active walk-ups entering $v_{b,e}$ would select the same path, so only one pathInfo would be set to $-V$. Also, an internally active walk-up cannot pass through $v_{b,e}$ because it is externally active.

Rule 10. If an externally active walk-up(e) reaches vertex v then it terminates, and the pathInfo of the border used to reach v are set to $+V$.

Rule 11. If an externally active walk-up(e) enters a vertex $v_{b,e}$ that has one pathInfo integer clear and the other set to $-V$, then the marked border leading to the root r_b must be selected and the $-V$ settings along it changed to $+V$. Then, walk-up(e) ascends to the next bicomp traversed by $S(e)$.

Rule 12. If an externally active walk-up(e) enters a vertex $v_{b,e}$ whose pathInfo integers are both clear, then walk-up(e) traverses $\mathcal{A}$ and $\mathcal{B}$ in parallel to find r_b by the shortest border that is not blocked by an externally active vertex. (a) If both borders to r_b are blocked by externally active vertices, then the graph is non-planar. Otherwise, let $\mathcal{A}$ denote the shorter unblocked border to r_b. (b) If the pathInfo in r_b is set to $-V$ for the opposing border path $\mathcal{B}$ and if $\mathcal{B}$ is not blocked, then walk-up(e) must select $\mathcal{B}$, set its pathInfo to $+V$, and ascend to the next bicomp traversed by $S(e)$. (c) If the pathInfo in r_b is set to $-V$ for the opposing border path $\mathcal{B}$ but $\mathcal{B}$ is blocked, then walk-up(e) must select $\mathcal{A}$ and set its pathInfo to $+V$. If r_b is externally active, then the graph is non-planar. Otherwise, walk-up(e) ascends to the next bicomp traversed by $S(e)$. (d) If the pathInfo in r_b is set to $+V$ for the opposing border path $\mathcal{B}$, then walk-up(e) must select $\mathcal{A}$ and set its pathInfo to $+V$. If r_b is externally active, then the graph is non-planar. Otherwise, walk-up(e) ascends to the next bicomp traversed by $S(e)$. (e) If the pathInfo in r_b is clear for the opposing border $\mathcal{B}$, then walk-up(e) must select $\mathcal{A}$, set its pathInfo to $+V$, and ascend to the next bicomp traversed by $S(e)$.

Therefore, the following Theorem can be stated.

Theorem 1 *Let G be a graph with n vertices, and let v_i be the vertex with $DFS(v_i) = i$, for $i = 1, \dots, n$. Graph G is planar iff the walk-up phase for vertex v_i, with $i = n, \dots, 1$, selects a path from w_e to v_i for each edge $e = (w_e, v_i)$ in $B_{in}(v_i)$.*

5 Experimental Analysis

In this section we compare our implementation (in the following called **BM-GDT**) with a selection of other planarity testing and embedding algorithm implementations, providing an assessment of the state of the art in this field.

All the tests have been performed on a 2.4GHz Pentium IV personal computer with 1GB of RAM.

We generated the test suites of graphs with Leda [14] and Pigale [7] generators, which in addition to be well known and widely used, are both fast and rigorously built. For each test suite we generated graphs ranging from 10,000 to 100,000 vertices, increasing each time by 5,000 vertices, 10 graphs for each type, with the exception of the test suite labeled `Planar-Conn-P`, for which we generated graphs ranging from 20,000 to 200,000 edges, increasing each time by 10,000 edges, 10 graphs for each type.

`Planar-L`: planar graphs generated by LEDA. The graph are not necessarily connected. The number of edges is twice the number of vertices. `Max-Planar-L`: maximal planar graphs generated by LEDA, hence connected and with $3n - 6$ edges, where n is the number of vertices. `Non-Planar-L`: the same graphs as `Max-Planar-L` in which one edge was added between two non-adjacent vertices. `Random-L`: random graphs generated by LEDA. The number of edges is chosen to be two times the number of vertices. The graph are not necessarily connected and may be planar. `Planar-Conn-P`: planar connected graphs generated by Pigale. After the generation, the graphs were processed in order to eliminate multiple edges and self-loops. `Random-P`: random graphs generated by Pigale. The number of edges is chosen to be two times the number of vertices. The graph are not necessarily connected but may be planar.

We tested the performance of the implementation of **BM-GDT** against: **LEC-L** Lempel, Even, and Cederbaum algorithm [12], LEDA implementation; **LEC-GTL** Lempel, Even, and Cederbaum algorithm [12], GTL implementation [9]; **HT-L** Hopcroft and Tarjan algorithm [10], LEDA implementation. **FR-P** de Fraysseix and Rosenstiehl algorithm (unpublished, see [6] for basic principles), Pigale implementation; **BM2** Boyer-Myrvold (new algorithm, unpublished see [3]), implementation by the first author. All the implementations are in C++ but for **BM2**, which is implemented in C. The implementation of **BM-GDT** uses the data structures for representing graphs of the GDToolkit library [8], based, in turn, on Leda. Each algorithm implementation extracts the graph from a file and constructs its own data structure. The timer starts after the file has been completely read and stops when the test fails (non-planar) or when the data structure with the embedding has been constructed (planar).

All the implementations gave the same results as for the planarity or non planarity of the graphs. The experiments (see Fig. 3) reveal that most of the algorithms have amazingly short computation times. Even the slowest implementations tested for planarity and, in case, constructed an embedding of the graphs in less than 100 μsec. per vertex

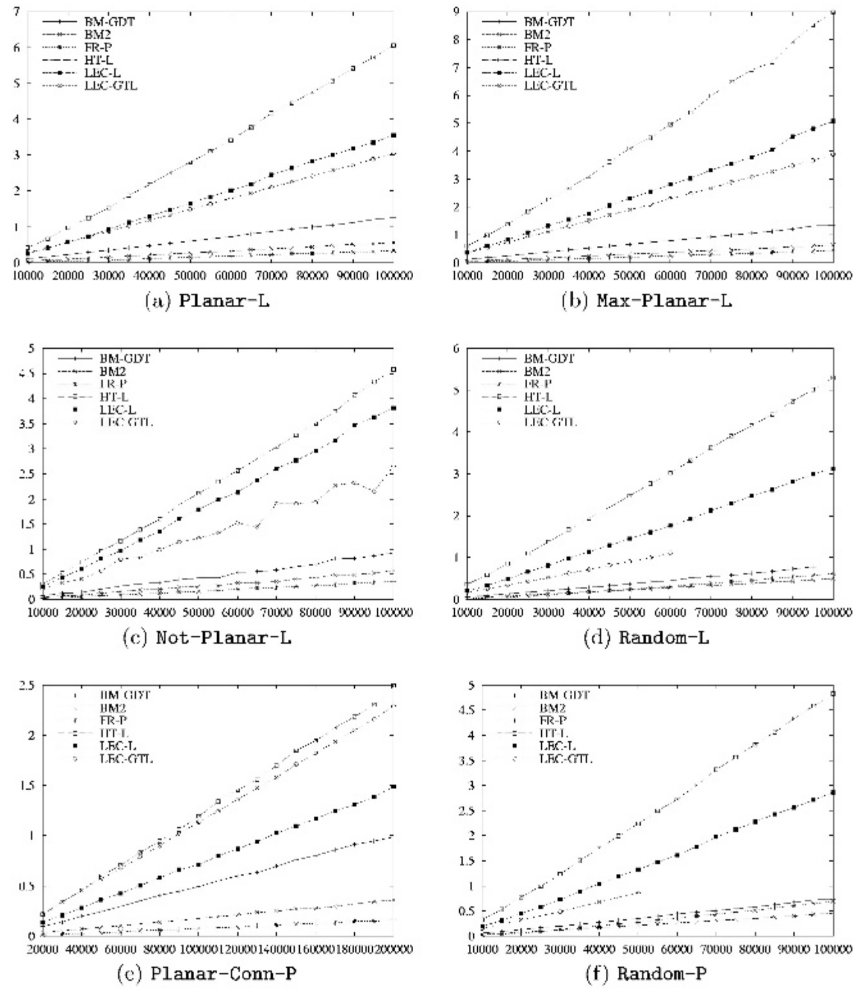

Fig. 3. The running times (seconds) of the tested algorithms against the test suites generated by Leda and Pigale.

FR-P, **BM-GDT** and **BM2** had similar performance profiles. Much slower are **LEC-L**, **LEC-GTL**, and **HT-L**. Also, we were unable to produce an output for algorithm **LEC-GTL** when very big non-planar graphs were involved, because of premature terminations of the program. In addition to connected planar graphs (Fig 3.e), we also tested the algorithms against planar biconnected and triconnected graphs generated by Pigale with analogous parameters and ranges, obtaining similar results.

We have to remark that some of the implementations rely on data structures for graphs representations that are "general purpose" and hence are not specifically tailored for quick planarity testing, especially when very large data sets are involved.

References

1. 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, 1976.
2. 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*, pages 140–146, 1999.
3. J. Boyer and W. Myrvold. Stop minding your P's and Q's: Simplified planarity by edge addition. 2003. Submitted. Preprint at `http://www.pacificcoast.net/~lightning/planarity.ps`.
4. J. M. Boyer, P. 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. Tech. Report RT-DIA-83-2003, Dept. of Computer Sci., Univ. di Roma Tre, 2003.
5. 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.
6. H. de Fraisseix and P. Rosenstiehl. A characterization of planar graphs by Trémaux orders. *Combinatorica*, 5(2):127–135, 1985.
7. H. de Fraysseix and P. Ossona de Mendez. P.I.G.A.L.E - Public Implementation of a Graph Algorithm Library and Editor. SourceForge project page `http://sourceforge.net/projects/pigale`.
8. GDToolkit. An object-oriented library for handling and drawing graphs. Third University of Rome, `http://www.dia.uniroma3.it/~gdt`.
9. GTL. Graph template library. University of Passau - FMI - Theor. Comp. Science `http://infosun.fmi.uni-passau.de/GTL/`.
10. J. Hopcroft and R. E. Tarjan. Efficient planarity testing. *J. ACM*, 21(4), 1974.
11. W.-L. Hsu. An efficient implementation fo the PC-Tree algorithm of Shih and Hsu's planarity test. Tech. Report, Inst. of Inf. Science, Academia Sinica, 2003.
12. 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.
13. K. Mehlhorn and P. Mutzel. On the embedding phase of the Hopcroft and Tarjan planarity testing algorithm. *Algorithmica*, 16:233–242, 1996.
14. K. Mehlhorn and S. Näher. *LEDA: A Platform for Combinatorial and Geometric Computing*. Cambridge University Press, New York, 1998.
15. 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.
16. W.-K. Shih and W.-L. Hsu. A new planarity test. *Theor. Comp. Sci.*, 223, 1999.