Incremental Convex Planarity Testing[1]

Giuseppe Di Battista[2]

Dipartimento di Informatica e Automazione, Università degli Studi di Roma Tre, Via della Vasca Navale 79, 00146 Rome, Italy
E-mail: gdb@dia.uniroma3.it

and

Roberto Tamassia and Luca Vismara[3]

Center for Geometric Computing, Department of Computer Science, Brown University, Providence, Rhode Island 02912-1910
E-mail: rt@cs.brown.edu, lv@cs.brown.edu

Received February 17, 1997; final manuscript received September 29, 2000

An important class of planar straight-line drawings of graphs are convex drawings, in which all the faces are drawn as convex polygons. A planar graph is said to be convex planar if it admits a convex drawing. We give a new combinatorial characterization of convex planar graphs based on the decomposition of a biconnected graph into its triconnected components. We then consider the problem of testing convex planarity in an incremental environment, where a biconnected planar graph is subject to on-line insertions of vertices and edges. We present a data structure for the on-line incremental convex planarity testing problem with the following performance, where n denotes the current number of vertices of the graph: (strictly) convex planarity testing takes $O(1)$ worst-case time, insertion of vertices takes $O(\log n)$ worst-case time, insertion of edges takes $O(\log n)$ amortized time, and the space requirement of the data structure is $O(n)$. © 2001 Academic Press

INTRODUCTION

Planar straight-line drawings of planar graphs are especially interesting for their combinatorial and geometric properties. A classical result independently established by Steinitz and Rademacher [45], Wagner [56], Fary [29], and Stein [44] shows that every planar graph has a planar straight-line drawing. A grid drawing is a drawing in which the vertices have integer coordinates. Independently, de Fraysseix *et al.* [12], and Schnyder [40] have shown that every n-vertex planar graph has a planar straight-line grid drawing with $O(n^2)$ area.

An important class of planar straight-line drawings are convex drawings, in which all the faces are drawn as convex polygons (see Figs. 1a and 2a). Convex drawings of planar graphs have been extensively studied in graph theory. A planar graph is said to be convex planar if it admits a convex drawing. Tutte [54, 55] has considered strictly convex drawings, in which faces are strictly convex polygons (i.e., 180° angles are not allowed). He has shown that every triconnected planar graph is strictly convex planar, and that a strictly convex drawing can be constructed by solving a system of linear equations. Tutte [54, 55], Thomassen [52, 53], Chiba *et al.* [6], and Djidjev [24] have presented combinatorial characterizations of convex and strictly convex planar graphs. Chiba *et al.* [6] have presented a linear time algorithm for testing convex planarity, based on their characterization, and a linear time algorithm for constructing

[1] Research supported in part by the National Science Foundation under Grants CCR-9732327 and CDA-9703080, by the U.S. Army Research Office under Grant DAAH04-96-1-0013, by the NATO Scientific Affairs Division under Collaborative Research Grant 911016, by Grant 94.00023.CT07 of the Consiglio Nazionale delle Ricerche, and by the ESPRIT Long Term Research of the European Community under Project 20244 (ALCOM-IT). A preliminary version of this paper was presented at the "20th International Workshop on Graph-Theoretic Concepts in Computer Science (WG '94), Herrsching (München), Germany, 1994."

[2] Research performed in part while this author was with the Dipartimento di Informatica e Sistemistica, Università degli Studi di Roma "La Sapienza" and with the Dipartimento di Ingegneria e Fisica dell'Ambiente, Università degli Studi della Basilicata.

[3] Research performed in part while this author was with the Dipartimento di Informatica e Sistemistica, Università degli Studi di Roma "La Sapienza".

0890-5401/01 $35.00

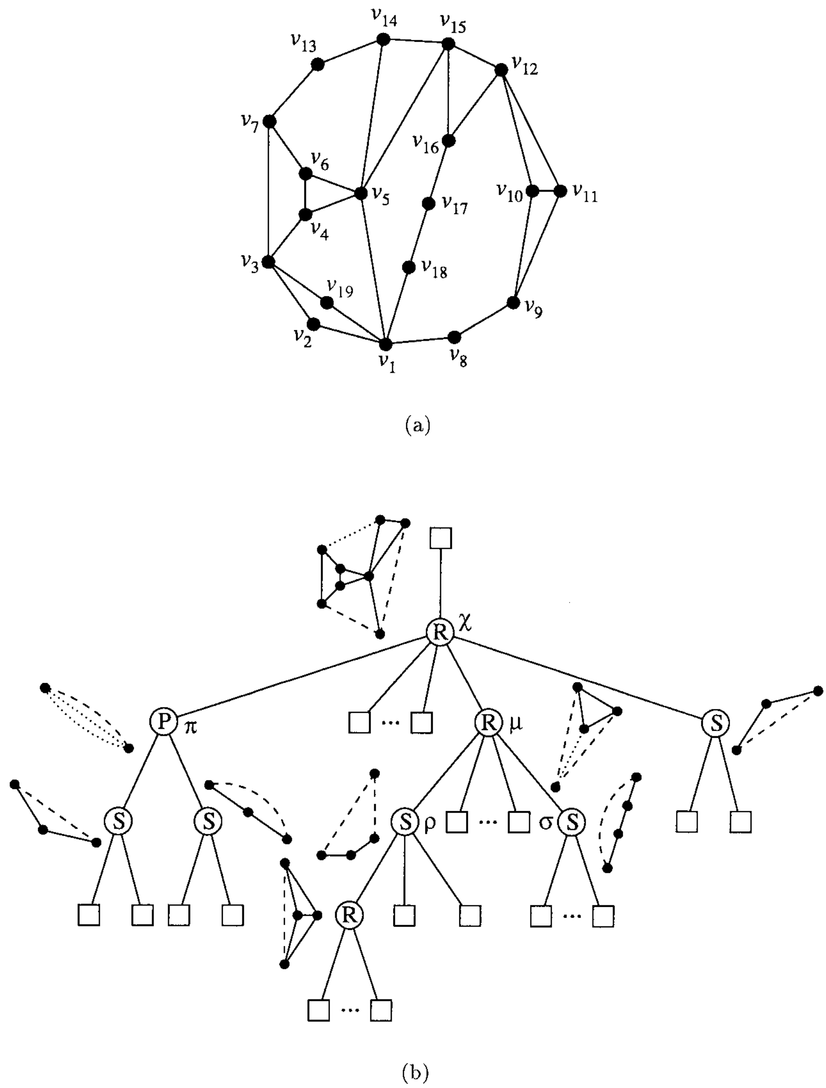
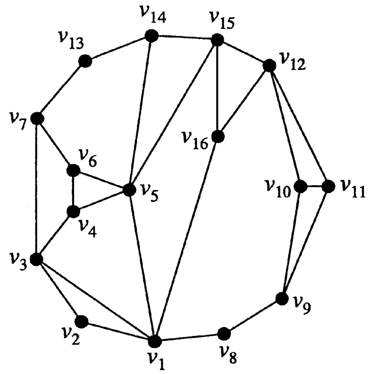


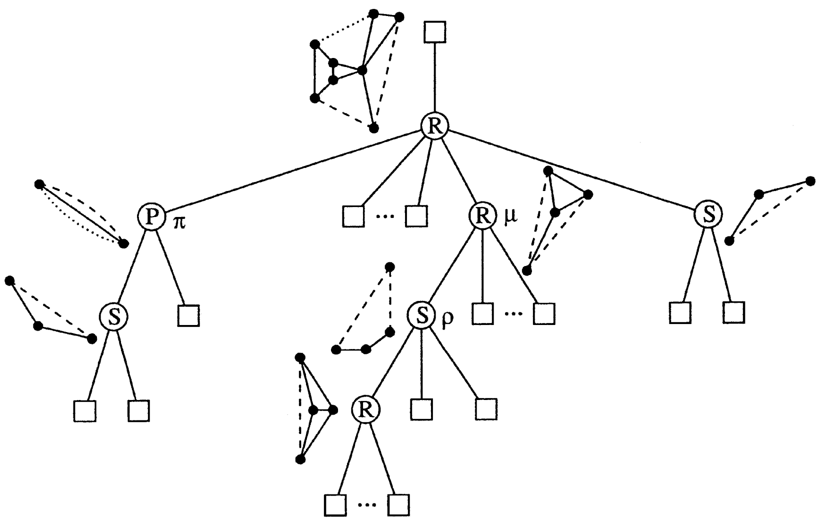
FIG. 1. (a) A convex drawing of a biconnected planar graph G. (b) The SPQR-tree of G with respect to reference edge (v_3, v_7) and the skeletons of its non-Q-nodes.

convex drawings with real coordinates for the vertices, based on Thomassen's characterization. An alternative linear time algorithm for testing convex planarity has been presented by Djidjev [24]. Chiba *et al.* [5] have extended the results of [6] to construct "quasi-convex" drawings of graphs that are not convex planar. Kant [33] has presented a linear time algorithm for constructing convex drawings of triconnected planar graphs with integer coordinates for the vertices and quadratic area. The constant factors for the area were later reduced by Chrobak and Kant [8]. Chrobak *et al.* [7] have presented algorithms for constructing convex drawings in the plane and in 3D space with integer or rational coordinates for the vertices under various resolution rules.

The study of *dynamic* graph problems has acquired increasing interest in the past decade and is motivated by various important applications in network optimization, VLSI layout, computational geometry, and distributed computing. The existing literature includes work on connected, biconnected, and triconnected components, transitive closure, shortest path, minimum spanning tree, planar embedding, and planarity testing (for a brief survey, see Section 2 of [20]). A dynamic graph problem consists of a sequence of query and update operations on a graph, such that each operation is completed before the next one is processed. If the sequence of operations is not known in advance, the term *on-line* dynamic graph problem is used. Typically, the update operations are insertions and deletions of vertices and edges. If only insertions or deletions are allowed, the graph problem is called *semi-dynamic*; otherwise,



(a)



(b)

FIG. 2. (a) A strictly convex drawing of a biconnected planar graph G. (b) The SPQR-tree of G with respect to reference edge (v_3, v_7) and the skeletons of its non-Q-nodes.

it is called *fully dynamic*. In particular, semi-dynamic graph problems are also referred to as *incremental* graph problems, if only insertions are allowed, and *decremental* graph problems, if only deletions are allowed.

The concept of amortized complexity [1, 10, 51] is often used in the analysis of algorithms and data structures for dynamic graph problems. In an amortized analysis, the time required to perform a sequence of operations is averaged over all the operations performed. Through amortized analysis one can show that the average cost of an operation in the sequence is small, even though a single operation may be expensive. Note that, unlike average-case analysis, probability is not used in amortized analysis.

Two of the most studied dynamic graph problems are the *dynamic embedding* problem and the *dynamic planarity testing* problem. In both cases, the graph is subject to on-line insertions and deletions of vertices and edges. In the dynamic embedding problem, a specific embedding of the graph is maintained; the query is to determine whether there is a face of the current embedding that contains two given vertices. The dynamic planarity testing problem is more general: instead of maintaining a specific embedding of the graph, an implicit representation of all the possible embeddings of the graph is maintained (we recall that a graph may have an exponential number of different embeddings); the query is to determine whether there is an embedding of the current graph such that two given vertices are on the same face. Tamassia [47] has presented a data structure for the incremental embedding problem (and for a restricted version of the fully dynamic embedding problem) with $O(\log n)$ query and update time (amortized for edge insertion). A data structure for the fully dynamic embedding problem with $O(\log^2 n)$ query and update time has been presented by Italiano *et al.* [32]. As for the dynamic planarity testing problem, Di Battista and Tamassia [20] have presented a data structure for the incremental planarity testing problem with $O(\log n)$ query and update time (amortized for edge insertion). This time bound was reduced first by Westbrook [57], who showed that a sequence of k query and update operations can be performed in $O(k\alpha(k, n))$ expected time, and then by La Poutré [35], who showed that the sequence of operations can be performed in $O(k\alpha(k, n))$ deterministic time; $\alpha(k, n)$ is the very slowly growing inverse of Ackermann's function. The best result for the fully dynamic planarity testing problem is that of Eppstein *et al.* [26], who presented a data structure with $O(\sqrt{n})$ amortized query and update time.

In this paper, we present the following results on convex planarity:

- We give a new combinatorial characterization of convex planar graphs and strictly convex planar graphs, alternative to those present in the literature [6, 24, 52–55], which is based on the decomposition of a biconnected graph into its triconnected components [31].
- We consider the problem of testing convex planarity in an incremental environment, where a biconnected planar graph is subject to on-line insertions of vertices and edges. We present a data structure for the on-line incremental convex planarity testing problem with the following performance, where n denotes the number of vertices of the graph: (strictly) convex planarity testing takes $O(1)$ worst-case time, insertion of vertices takes $O(\log n)$ worst-case time, insertion of edges takes $O(\log n)$ amortized time, and the space requirement of the data structure is $O(n)$.

Note that the (strictly) convex planarity property for planar graphs is not monotone. Namely, there exist sequences of insertions of vertices and edges such that the current graph alternates between being (strictly) convex planar and being nonconvex.

Besides their theoretical significance, our results are motivated by the development of advanced graph drawing systems in information visualization applications. Examples include programming environments (e.g., displaying entity-relationship diagrams and subroutine-call graphs), algorithm animation systems (e.g., representing data structures), and project planning systems (e.g., displaying PERT diagrams and organization charts). Several advanced graph drawing systems have been developed (see, for example, [2, 4, 14, 17, 30]); they usually contain a library of graph drawing algorithms, each devised to take into account a specific set of aesthetic requirements. Thus, in these systems, the problem of selecting the algorithm of the library that provides the "best" visualization of a certain graph is of crucial importance. Since advanced graph drawing systems are often used interactively, the above selection problem must be solved under tight performance requirements, especially for large graphs. The problem becomes harder when the graph to be represented is subject to frequent updates. In an ideal scenario, each graph drawing algorithm of the library should be supplemented with a data structure for efficiently testing whether it can be used to represent the current graph. Typically, after each update of the graph, only a certain number of tests will succeed, quickly indicating which of the available drawing algorithms can actually be applied to the current graph. For example, one can use the data structure described in this paper for efficiently testing if, after a certain number of updates, a graph is (strictly) convex planar; if this is the case, one of the existing algorithms for constructing (strictly) convex drawings (e.g., the algorithm presented in [6]) can be used.

On the other hand, the problem of efficiently maintaining the drawing of a graph in a semi-dynamic or fully dynamic environment is a long-standing open problem in graph drawing. Its difficulty arises from the fact that even a single update to the graph may cause a major restructuring of the drawing. A model for dynamic graph drawing and its application to particular classes of planar graphs is presented in [9]. We will further discuss the issue in the open problems section.

The rest of the paper is organized as follows. Preliminary definitions are given in Section 2. In Section 3 we present a combinatorial characterization of (strictly) convex planar graphs. The repertory of query and update operations for the on-line incremental convex planarity testing problem is described in Section 4. In Section 5 we present a data structure that supports this repertory. The implementation of query and update operations is described in Sections 6 and 7, respectively. In Section 8, we analyze the time complexity of the various operations. Open problems are discussed in Section 9.

2. PRELIMINARIES

We assume familiarity with graph terminology and basic properties of planar graphs (see, e.g., [38]). The graphs whose convex planarity we test are assumed to be simple, i.e., without self-loops and multiple edges. We recall some basic definitions on connectivity. A *separating k-set* of a graph is a set of k vertices whose removal disconnects the graph; separating 1-sets and 2-sets are called *cut-vertices* and *separation pairs*, respectively. A graph is *k-connected* if it contains more than k vertices and no separating $(k-1)$-set; 1-connected, 2-connected, and 3-connected graphs are called *connected*, *biconnected*, and *triconnected*, respectively. A *separating edge* of a graph is an edge whose removal disconnects the graph.

The *biconnected components* of a connected graph (also called *blocks*) are its maximal biconnected subgraphs and its separating edges.

The *triconnected components* of a biconnected graph G are defined as follows [31]. If G is triconnected, then G itself is the unique triconnected component of G. Otherwise, let $\{u, v\}$ be a separation pair of G. We partition the edges of G into two disjoint subsets E_1 and E_2, $|E_1| \geq 2$, $|E_2| \geq 2$, such that the subgraphs G_1 and G_2 induced by them have only vertices u and v in common. Graphs $G'_1 = G_1 + (u, v)$ and $G'_2 = G_2 + (u, v)$ are called the *split graphs* of G with respect to $\{u, v\}$ (multiple edges are allowed); edge (u, v) in G'_1 and G'_2 is called a *virtual edge*. Dividing G into split graphs G'_1 and G'_2 is called *splitting*. Reassembling split graphs G'_1 and G'_2 into G, is called *merging*. Note that only split graphs that resulted from the same splitting operation can be merged together. We continue the splitting process recursively on G'_1 and G'_2 until no further splitting is possible. The resulting graphs are each either a triconnected simple graph, or a set of three multiple edges (called *triple bond* in [31]), or a cycle of length three (called *triangle* in [31]). The *triconnected components* of G are obtained from these graphs by merging the triple bonds into maximal sets of multiple edges (called *bonds* in [31]), and the triangles into maximal simple cycles (called *polygons in* [31]). When merging triple bonds into bonds and triangles into polygons, virtual edges with both endvertices in common are removed; we will refer to the remaining virtual edges at the end of the merging process as the *virtual edges of the triconnected components*. Note that, although the graphs obtained at the end of the splitting process depend on the order of the splittings, the triconnected components of G are unique. See [31] for further details.

For background on graph drawing, see [3, 11, 13, 15, 16, 18, 22, 23, 34, 39, 46, 48, 49, 58]. A *drawing* of a graph maps each vertex to a distinct point of the plane and each edge (u, v) to a simple Jordan curve with endpoints u and v. A drawing is *planar* if no two edges intersect, except, possibly, at common endpoints. A graph is planar if it has a planar drawing. A *straight-line* drawing is a drawing in which every edge is mapped to a straight-line segment. Two planar drawings of a planar graph G are *equivalent* if, for each vertex v, they have the same clockwise circular sequence of edges incident with v. Hence, the planar drawings of G are partitioned into equivalence classes. Each of those classes is called an *embedding* of G. An *embedded* planar graph (also *plane* graph) is a planar graph with a prescribed embedding. A triconnected planar graph has a unique embedding, up to a reflection. A planar drawing divides the plane into topologically connected regions; cycles of G that bound a topologically connected region are called *faces*. The *external* face is the boundary of the external region; all the other faces are *internal*. Two equivalent planar drawings have the same faces. Hence, one can refer to the faces of an embedding.

A *polygon* is a finite set of segments such that every segment endpoint is shared by exactly two segments and no subset of segments has the same property. A polygon is *simple* if there is no pair of nonconsecutive segments sharing a point. A simple polygon is *convex* if its interior is a convex set. A simple polygon is *strictly convex* if its interior is a strictly convex set; i.e., no 180° angle is allowed. A *convex* drawing of a planar graph G is a planar straight-line drawing of G in which all the faces are drawn as convex polygons. A *strictly convex* drawing of a planar graph G is a planar straight-line drawing of G in which all the faces are drawn as strictly convex polygons. See Figs. 1a and 2a compared to Fig. 3a. A planar graph is said to be (*strictly*) *convex planar* if it admits a (strictly) convex drawing.

LEMMA 1. *A planar graph is* (*strictly*) *convex planar only if it is biconnected.*

Proof. Let G be a planar graph. We prove the claim by contradiction. If G is connected but not biconnected, two cases are possible:

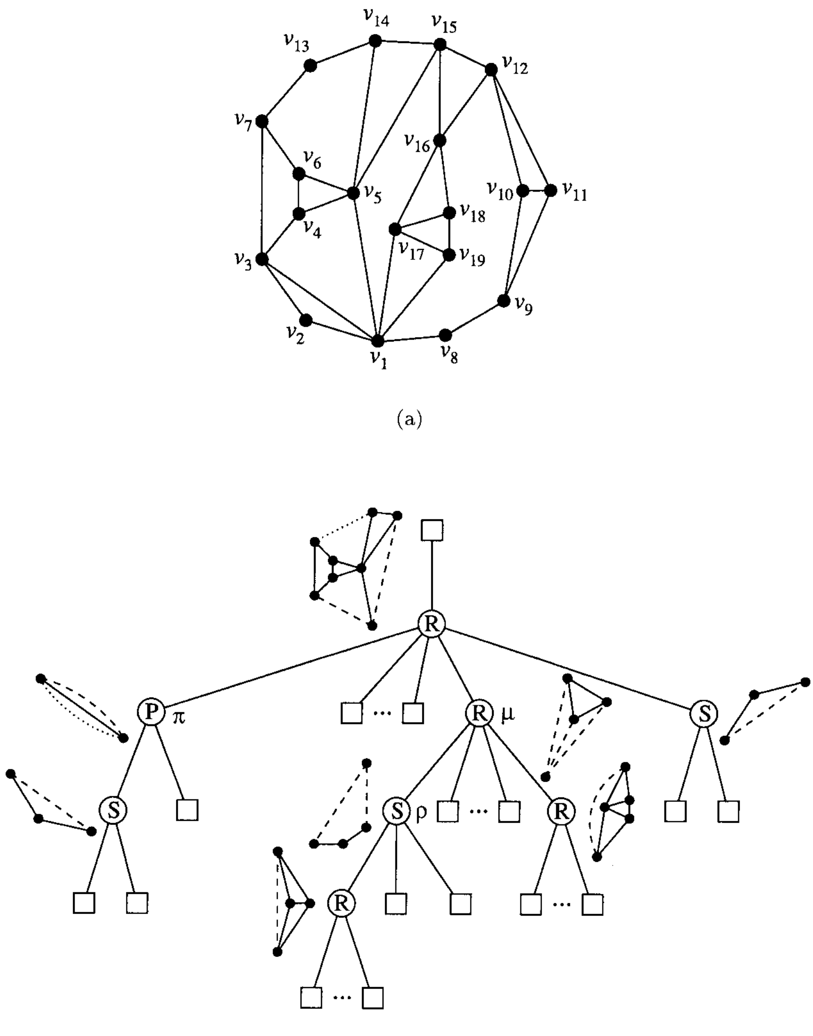


FIG. 3. (a) A nonconvex drawing of a biconnected planar graph G. (b) The SPQR-tree of G with respect to reference edge (v_3, v_7) and the skeletons of its non-Q-nodes.

1. If G is a path, then in any drawing of G the two distinct points representing the first and the last vertex of G are not shared by two segments from the set of segments representing the (only) face f of G. Thus, the set of segments representing f is not a polygon.

2. Otherwise, there exist at least one cut-vertex v of G and one face f of G containing v such that, in any drawing of G, the point representing v is shared by more than two segments from the set of segments representing f. Thus, the set of segments representing f is not a polygon.

If G is not connected, then in any drawing of G there exists at least one face represented by a set of segments that do not satisfy the minimality property in the definition of polygon. ■

In the rest of this section, the *SPQR-tree* presented in [19, 20] is described. Let G be a biconnected graph. A *split pair* of G is either a pair of adjacent vertices or a separation pair (note that the two cases are not disjoint, since the vertices of a separation pair may be adjacent). If the two vertices are adjacent then the split pair is called *trivial*, otherwise it is called *nontrivial*. A *split component* of a split pair $\{u, v\}$ is either an edge (u, v) or a maximal subgraph C of G such that C contains u and v, and $\{u, v\}$ is not a split pair of C. In the former case the split component is called *trivial*, in the latter *nontrivial*. Vertices u and v are called the *poles* of the split component. Note that each vertex of G distinct from u and v belongs to exactly one nontrivial split component of $\{u, v\}$. Let $\{s, t\}$ be a split pair of G. A *maximal* *split pair* $\{u, v\}$ of G with respect to $\{s, t\}$ is a split pair of G distinct from $\{s, t\}$ such that for any other split pair $\{u', v'\}$ of G, there exists a split component of $\{u', v'\}$ containing vertices u, v, s, and t.

In the graph in Fig. 1a, $\{v_1, v_5\}$ is a trivial split pair, $\{v_9, v_{12}\}$ is a nontrivial split pair, edge (v_1, v_5) is a trivial split component, the subgraph induced by v_9, v_{10}, v_{11}, and v_{12} is a nontrivial split component, and split pair $\{v_1, v_{15}\}$ is maximal with respect to $\{v_3, v_7\}$, while split pair $\{v_1, v_{12}\}$ is not maximal with respect to $\{v_3, v_7\}$.

Let $e = (s, t)$ be an edge of G, called the *reference edge*. The SPQR-tree T of G with respect to e describes a recursive decomposition of G induced by its split pairs. Tree T is a rooted ordered tree whose nodes are of four types: S, P, Q, and R. Each node μ of T has an associated biconnected multigraph, called the *skeleton* of μ and denoted *skeleton*(μ). Also, each node μ of T (except the root) is associated with an edge of the skeleton of the parent ν of μ, called the *virtual edge* of μ in *skeleton*(ν); at the same time, ν is associated with a virtual edge in *skeleton*(μ). Tree T is recursively defined as follows.

Trivial case: If G consists of exactly two multiple edges between s and t, then T consists of a single Q-node whose skeleton is G itself.

Parallel case: If the split pair $\{s, t\}$ has at least three split components $G_0 = e, G_1, \ldots, G_k, k \geq 2$, then the root of T is a P-node μ. Graph *skeleton*(μ) consists of $k + 1$ multiple edges between s and t, denoted $e_{\mu 0}, e_{\mu 1}, \ldots, e_{\mu k}$ where $e_{\mu 0} = e$.

Series case: If the split pair $\{s, t\}$ has exactly two split components and one of them has at least one cut-vertex, then the root of T is an S-node μ. One of the split components of $\{s, t\}$ is the reference edge e. Let $c_1, \ldots, c_{k-1}, k \geq 2$, be the cut-vertices that partition $G - e$ into its blocks $G_1, \ldots, G_k$, in this order from s to t. Graph *skeleton* (μ) is the cycle $e_{\mu 0}, e_{\mu 1}, \ldots, e_{\mu k}$, where $e_{\mu 0} = e$, $c_0 = s$, $c_k = t$, and e_{μ_i} connects c_{i-1} with c_i, $i = 1, \ldots, k$. Note that in this case $G_1, \ldots, G_k$ are not split components of $\{s, t\}$.

Rigid case: If none of the cases above applies, then the root of T is an R-node μ. Let $\{s_1, t_1\}, \ldots, \{s_k, t_k\}$, $k \geq 1$, be the maximal split pairs of G with respect to $\{s, t\}$, and, for $i = 1, \ldots, k$, let G_i be the union of all the split components of $\{s_i, t_i\}$ except that containing the reference edge e. Graph *skeleton*(μ) is obtained from G by replacing each subgraph G_i with the edge $e_{\mu_i} = (s_i, t_i)$. Note that in this case $G_1, \ldots, G_k$ are not split components of $\{s, t\}$.

For each split component $G_i, i = 1, \ldots, k$, defined in the above cases, let e_μ be an additional edge between the poles of G_i. Except for the trivial case, μ has children $\mu_1, \ldots, \mu_k$ in this order, such that μ_i is the root of the SPQR-tree of graph $G_i \cup e_\mu$, $i = 1, \ldots, k$, with respect to reference edge e_μ. The tree so obtained has a Q-node associated with each edge of G, except the reference edge e. We complete the SPQR-tree by replacing the reference edge e in *skeleton*(μ) with a virtual edge, by adding another Q-node, representing e, and by making it the parent of μ so that it becomes the root. Note that, from the above definition, it follows that two P-nodes or two S-nodes cannot be adjacent in T. Examples of SPQR-trees are shown in Figs. 1b, 2b, and 3b; the Q-nodes are represented by squares and the skeletons of the Q-nodes are not shown.

The virtual edge of node μ_i is edge e_{μ_i} of *skeleton*(μ), while edge e_μ of *skeleton*(μ_i) is the virtual edge of node μ. A virtual edge e_{μ_i} is said to be *trivial* if the corresponding node μ_i is a Q-node, *nontrivial* otherwise. The endvertices s_i and t_i of e_{μ_i} are called the *poles* of μ_i. In Figs. 1b, 2b, and 3b, the nontrivial virtual edges are represented by dashed or dotted lines and the trivial virtual edges are represented by solid lines.

Letting μ be a node of T, we have the following:

- if μ is an R-node, then *skeleton*(μ) is a triconnected simple graph;
- if μ is an S-node, then *skeleton*(μ) is a cycle;
- if μ is a P-node, then *skeleton*(μ) is a multigraph consisting of a bundle of multiple edges;
- if μ is a Q-node, then *skeleton*(μ) is a multigraph consisting of two multiple edges.

The skeletons of the nodes of T are homeomorphic to subgraphs of G. Also, the union of the sets of split pairs of the skeletons of the nodes of T is equal to the set of split pairs of G. It is possible to show that SPQR-trees of the same graph with respect to different reference edges are isomorphic and are obtained one from the other by selecting a different Q-node as the root.

SPQR-trees are closely related to the decomposition of biconnected graphs into triconnected components [31]. Namely, the triconnected components of a biconnected graph G are in one-to-one correspondence with the skeletons of the non-Q-nodes of the SPQR-tree T of G: the skeletons of the R-nodes correspond to the triconnected simple graphs, the skeletons of the S-nodes correspond to the polygons, and the skeletons of the P-nodes correspond to the bonds. In particular, for each non-Q-node μ of T, the nontrivial virtual edges of *skeleton*(μ) are in one-to-one correspondence with the virtual edges of a triconnected component of G, and the trivial virtual edges of *skeleton*(μ) are in one-to-one correspondence with the (nonvirtual) edges of a triconnected component of G.

The SPQR-tree T of a planar graph with n vertices and m edges has m Q-nodes and $O(n)$ S-nodes, P-nodes, and R-nodes. Also, the total number of vertices of the skeletons stored at the nodes of T is $O(n)$.

3. A CHARACTERIZATION OF (STRICTLY) CONVEX PLANAR GRAPHS

Let Γ be a planar straight-line drawing of a biconnected planar graph G. A vertex of G is said to be *external* (respectively, *internal*) in Γ if it is (respectively, it is not) a vertex of the external face of Γ. An *external* (respectively, *internal*) edge in Γ is defined analogously. A subgraph G' of G is *drawn outside* (respectively, *inside*) in Γ if G' has (respectively, does not have) external edges in Γ.

LEMMA 2. *Let Γ be a strictly convex drawing of a biconnected planar graph G. The nontrivial split components of G are drawn outside in Γ.*

Proof. Suppose, for a contradiction, that a nontrivial split component C of a split pair $\{u, v\}$ is drawn inside in Γ (see Fig. 4). Let p_1 (p_2) be the path of C between u and v such that all the vertices and edges of C not in p_1 (p_2) are on its right (left) side in Γ. Note that p_1 and p_2 may have some vertices (besides u and v) and edges in common. Path p_1 (p_2) is part of an internal face $f_1(f_2)$ of G . By easy geometric considerations, it follows that, if f_1 is drawn as a strictly convex polygon in Γ, then f_2 is not and vice versa. Thus, Γ is not a strictly convex drawing, which is a contradiction. ■

COROLLARY 1. *Let Γ be a strictly convex drawing of a biconnected planar graph G. For each separation pair $\{u, v\}$ of G, vertices u and v must be external in Γ.*

Proof. Suppose, for a contradiction, that one vertex of a separation pair $\{u, v\}$, say v, is internal in Γ. Hence, all the vertices and edges of G that are external in Γ, except u, belong to a common split component of $\{u, v\}$, while all the other split components of $\{u, v\}$ are drawn inside in Γ. Thus, by Lemma 2, Γ is not a strictly convex drawing, which is a contradiction. ■

We are now ready to state the main results of this section.

THEOREM 1. *Let G be a biconnected planar graph. Graph G is strictly convex planar if and only if, for each triconnected component C of G, there exists an embedding of C such that all the virtual edges of C are on the same face.*

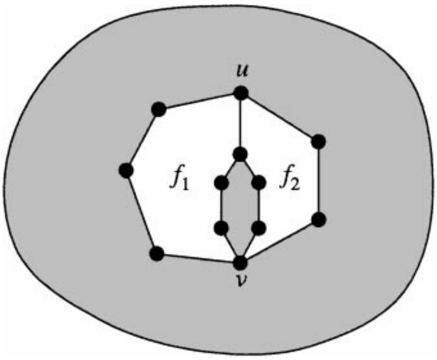


FIG. 4. A planar straight-line drawing Γ of a biconnected planar graph. One of the split components of split pair $\{u, v\}$ is drawn inside in Γ.

In Section 2, we have described how the triconnected components of a biconnected graph G are in one-to-one correspondence with the skeletons of the non-Q-nodes of the SPQR-tree T of G, and how the virtual edges of the triconnected components of G are in one-to-one correspondence with the nontrivial virtual edges of the skeletons of the non-Q-nodes of T. This allows us to restate and prove Theorem 1 as follows.

THEOREM 2. *Let G be a biconnected planar graph and let T be the SPQR-tree of G. Graph G is strictly convex planar if and only if, for each node μ of T, there exists an embedding of skeleton (μ) such that all the nontrivial virtual edges of skeleton (μ) are on the same face.*

Proof. Only if. Let Γ indicate a strictly convex drawing of G.

If μ is a Q-node or an S-node, then *skeleton*(μ) is a pair of multiple edges or a cycle, respectively, and the claim is trivially true.

If μ is a P-node, then suppose, for a contradiction, that *skeleton*(μ) contains three (multiple) nontrivial virtual edges with common endvertices u and v. Even if u and v are external vertices in Γ, one of the three (nontrivial) split components of $\{u, v\}$ is drawn "between" the other two, that is, inside in Γ. Thus, by Lemma 2, Γ is not strictly convex, which is a contradiction.

If μ is an R-node, then suppose, for a contradiction, that *skeleton*(μ) contains two nontrivial virtual edges (u_1, v_1) and (u_2, v_2) that are not on the same face. We recall that *skeleton*(μ) is a triconnected simple planar graph, and thus not all four vertices u_1, v_1, u_2, and v_2 can be on the same face in the unique embedding of *skeleton*(μ). A straight-line drawing of *skeleton*(μ) can be obtained from Γ by using the points and the segments representing the vertices and the trivial virtual edges of *skeleton*(μ), and by drawing the nontrivial virtual edges of *skeleton*(μ) as straight-line segments (that is, by replacing the drawings of some split components with straight-line segments). It follows that, also in Γ, not all four vertices u_1, v_1, u_2, and v_2 can be on the same face, in particular the external one; thus, at least one of them is internal in Γ. Since $\{u_1, v_1\}$ and $\{u_2, v_2\}$ are separation pairs of G, from Corollary 1 it follows that Γ is not strictly convex, which is a contradiction.

If. We show how to construct a strictly convex drawing Γ of G while performing a preorder visit of T. All the external vertices of G in Γ are mapped to distinct points of a circle c. For each node μ of T, we choose as external the face of *skeleton*(μ) containing the nontrivial virtual edges and we draw *skeleton*(μ) in a circular segment of c.

At the beginning of the preorder visit of T, the circular segment coincides with c and we draw the skeleton of the root of T (two multiple virtual edges, one of which is trivial) as a chord of c. At each following step, let μ be the node currently visited and let ν be its parent. If μ is not a Q-node, the virtual edge e_μ in *skeleton*(ν) is represented by a chord of c, which identifies a circular segment s_μ (see Fig. 5a).

If μ is a Q-node, *skeleton*(μ) is drawn by placing the poles of μ (i.e., the common endvertices of e_ν and of the trivial virtual edge in *skeleton*(μ)) at the endpoints of the chord identifying s_μ.

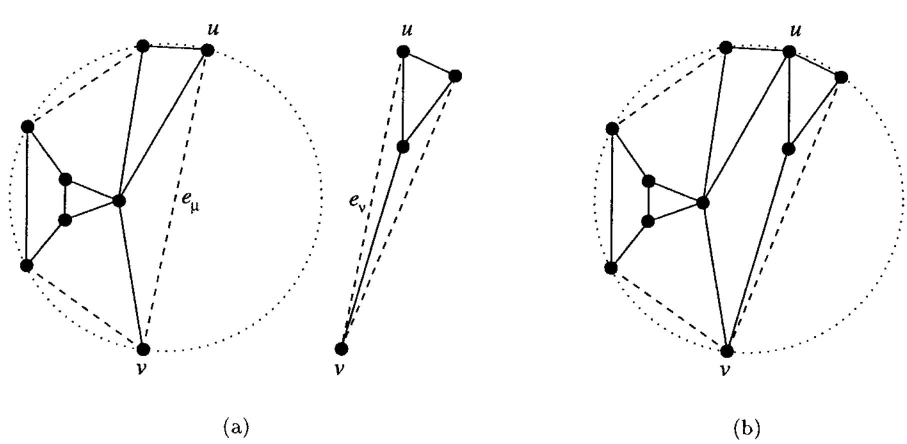


FIG. 5. An example of the construction in the proof of Theorem 2. (a) The current drawing and the skeleton of the node currently visited. (b) The new drawing.

If μ is a P-node, *skeleton*(μ) is drawn by placing the poles of μ (i.e., the common endvertices of e_ν and of the other two virtual edges in *skeleton*(μ), one of which is trivial) at the endpoints of the chord identifying s_μ.

If μ is an S-node, *skeleton*(μ) is drawn by placing the poles of μ (i.e., the endvertices of e_ν in *skeleton*(μ)) at the endpoints of the chord identifying s_μ, and the other vertices at distinct points of the circular arc of s_μ.

If μ is an R-node, *skeleton*(μ) is a triconnected simple planar graph. A strictly convex drawing of *skeleton*(μ) with a prescribed shape for an arbitrarily chosen external face can be obtained by using, e.g., the algorithm of Tutte [55], or the algorithm of Chiba *et al.* [6]. In particular, the poles of μ (i.e., the endvertices of e_ν in *skeleton*(μ)) are placed at the endpoints of the chord identifying s_μ, and the other external vertices of *skeleton*(μ) are placed at distinct points of the circular arc of s_μ.

Then e_μ and e_ν are removed from the drawing. If μ is a Q-node, the whole step consists of replacing a trivial virtual edge of the drawing with an edge of G . If μ is a P-node, it consists of replacing a nontrivial virtual edge of the drawing with two multiple virtual edges, one of which is trivial. If μ is an S-node, it consists of appending a strictly convex polygon to the drawing along a nontrivial virtual edge, which is then removed. If μ is an R-node, it consists of appending a strictly convex drawing of a triconnected simple planar graph to the drawing along a nontrivial virtual edge, which is then removed (see Fig. 5b).

Note that, at each step, the following invariants hold for the drawing that is being constructed:

1. The nontrivial virtual edges are external in the drawing, and are represented by chords of c.
2. If μ is an S-node or an R-node, the internal face f generated by the removal of e_μ and e_ν is a strictly convex polygon since: (i) the two faces sharing $e_\mu = e_\nu$ before the removal are strictly convex polygons; and (ii) the common endvertices u and v of e_μ and e_ν are placed on c and the drawing is contained in c, and thus the two angles of f around u and v are less than 180°.
3. The external face is a strictly convex polygon, since all its vertices are on c.

Finally, the planarity of Γ can be proved by observing that, for each node μ of T, the drawing of *skeleton*(μ) used in the construction of Γ is planar; by the third invariant, s_μ only contains e_μ, which is then removed together with e_ν; and the drawing of *skeleton*(μ) is contained in s_μ. ■

COROLLARY 2. *The strictly convex planarity of an n-vertex biconnected planar graph can be tested in $O(n)$ time.*

Proof. Let G be an n-vertex biconnected planar graph. Computing the triconnected components of G takes $O(n)$ time [31]. The total number of virtual edges in the triconnected components of G is $O(n)$ [31]; hence testing the condition of Theorem 1 takes $O(n)$ time. ■

It is easy to verify that the SPQR-tree in Fig. 2b satisfies the condition of Theorem 2. Hence, the graph in Fig. 2a is strictly convex planar. Consider, instead, the SPQR-trees in Figs. 1b and 3b. In both figures, the skeleton of R-node μ does not admit an embedding with all the nontrivial virtual edges on the same face. Hence, the condition of Theorem 2 is not satisfied, and the graphs in Figs. 1a and 3a are not strictly convex planar.

In the rest of this section we extend the characterization of Theorem 2 to nonstrictly convex drawings. Let G be a biconnected graph different from a cycle, let T be the SPQR-tree of G, and let μ be an S-node of T whose adjacent nodes, except one, ν, are Q-nodes. Then all the virtual edges of *skeleton*(μ) are trivial, except $e_\nu = (u, v)$, which is nontrivial. The pair of vertices $\{u, v\}$ is a split pair of G, and the edges of G corresponding to the trivial virtual edges of *skeleton*(μ) form a nontrivial split component C of $\{u, v\}$. C is a path and is called a (u, v)-*chain* of G. Node ν is either a P-node or an R-node, and the nontrivial virtual edge e_μ of *skeleton*(ν) is called a *chain* virtual edge. In Figs. 1b, 2b, 3b, 6b, 8b, and 10a, the chain virtual edges are represented by dotted lines.

LEMMA 3. *Let Γ be a convex drawing of a biconnected planar graph G. For each split pair $\{u, v\}$ of G, at most one (u, v)-chain can be drawn inside in Γ.*

Proof. Suppose, for a contradiction, that two (u, v)-chains C_1 and C_2 are drawn inside in Γ, and that no other split component of $\{u, v\}$ is drawn inside in Γ. Chain C_1 (C_2) is part of two internal faces

f_1 and f_3 (f_2 and f_3) of G. By easy geometric considerations, it follows that if f_1 and f_3 are drawn as convex polygons in Γ (by placing the vertices of C_1 on a straight-line segment) then f_2 is not, and if f_2 and f_3 are drawn as convex polygons in Γ (by placing the vertices of C_2 on a straight-line segment) then f_1 is not. Thus, Γ is not a convex drawing, which is a contradiction. ■

COROLLARY 3. *Let Γ be a convex drawing of a biconnected planar graph G. For each split pair $\{u, v\}$ of G, the following properties hold*:

1. *there exist at most three* (u, v)*-chains*; *and*
2. *if there exists a* (u, v)*-chain drawn inside in* Γ, *then u and v are not adjacent.*

Proof. Property 1 is proved by contradiction. Suppose that there exist four (u, v)-chains. Even if u and v are external vertices in Γ, two of the (u, v)-chains are drawn "between" the other two, that is, inside in Γ, but this contradicts Lemma 3.

Property 2 is proved, again, by contradiction. Suppose that there exists a (u, v)-chain drawn inside in Γ and that u and v are adjacent. As seen in the proof of Lemma 3, C is drawn by placing the vertices of C on a straight-line segment with endpoints corresponding to u and v. Thus, Γ is not planar, which is a contradiction. ■

A *reduced* graph of a biconnected graph G is a graph G', homeomorphic to G, obtained from G in the following way. If G is a cycle, then G' is equal to G. If G is not a cycle, then, for each nontrivial split pair $\{u, v\}$ of G that has one or more (u, v)-chains, exactly one (u, v)-chain is replaced with edge (u, v), called a *bypass edge*. Note that, for the nontrivial split pairs $\{u, v\}$ that have more than one (u, v)-chain, different choices of the (u, v)-chain to be replaced with a bypass edge lead to different reduced graphs. Thus, in general, a biconnected graph has more than one reduced graph.

Observe that the SPQR-tree T' of G' can be obtained from the SPQR-tree T of G as follows. If G is a cycle, then T' is equal to T. If G is not a cycle, then, for each S-node μ of T identifying a (u, v)-chain C, let ν be its only adjacent non-Q-node. If C is replaced with a bypass edge, then node μ and its adjacent Q-nodes are replaced with a Q-node ρ, and the chain virtual edge e_μ in *skeleton*(ν) is replaced with the trivial virtual edge e_ρ.

A reduced graph of the biconnected planar graph in Fig. 1a is shown in Fig. 6a; it is obtained by replacing one of the (v_1, v_3)-chains, the (v_1, v_{16})-chain, and the (v_7, v_{14})-chain with bypass edges. Its SPQR-tree with respect to reference edge (v_3, v_7) is shown in Fig. 6b.

LEMMA 4. *Let G be a biconnected graph and let G' be a reduced graph of G. Then G' is simple and biconnected.*

Proof. G' is simple since: (i) a (u, v)-chain is replaced with a bypass edge only if $\{u, v\}$ is a nontrivial split pair; (ii) if there exist two or more (u, v)-chains for a nontrivial split pair $\{u, v\}$, exactly one of them is replaced with a bypass edge. G' is biconnected since each path of G containing a (u, v)-chain as a subpath is not affected by its replacement with a bypass edge. ■

THEOREM 3. *Let G be a biconnected planar graph and let G' be a reduced graph of G. G is convex planar if and only if G' is strictly convex planar.*

Proof. If G is a cycle the claim is trivially proved. In the rest of the proof we assume that G is not a cycle.

Only if. Let Γ_c be a convex drawing of G. W.l.o.g., we can assume that there are no 180° angles around vertices of degree greater than 2, since they can be easily reduced to less than 180° angles by local adjustments at those vertices. We modify Γ_c as follows. We consider each nontrivial split pair $\{u, v\}$ of G that has at least one (u, v)-chain. By Property 1 of Corollary 3, we have three possible cases:

- There exists only one (u, v)-chain C, which can be drawn inside or outside in Γ_c.
- There exist exactly two (u, v)-chains. Since G is not a cycle, there exists a third split component C' of $\{u, v\}$, which is not a (u, v)-chain. With an argument similar to that used in the proof of Lemma 3, we can prove that C' must be drawn outside in Γ_c. It follows that one of the two (u, v)-chains is drawn "between" the other one and C', that is, inside in Γ_c. Let C be such a (u, v)-chain.

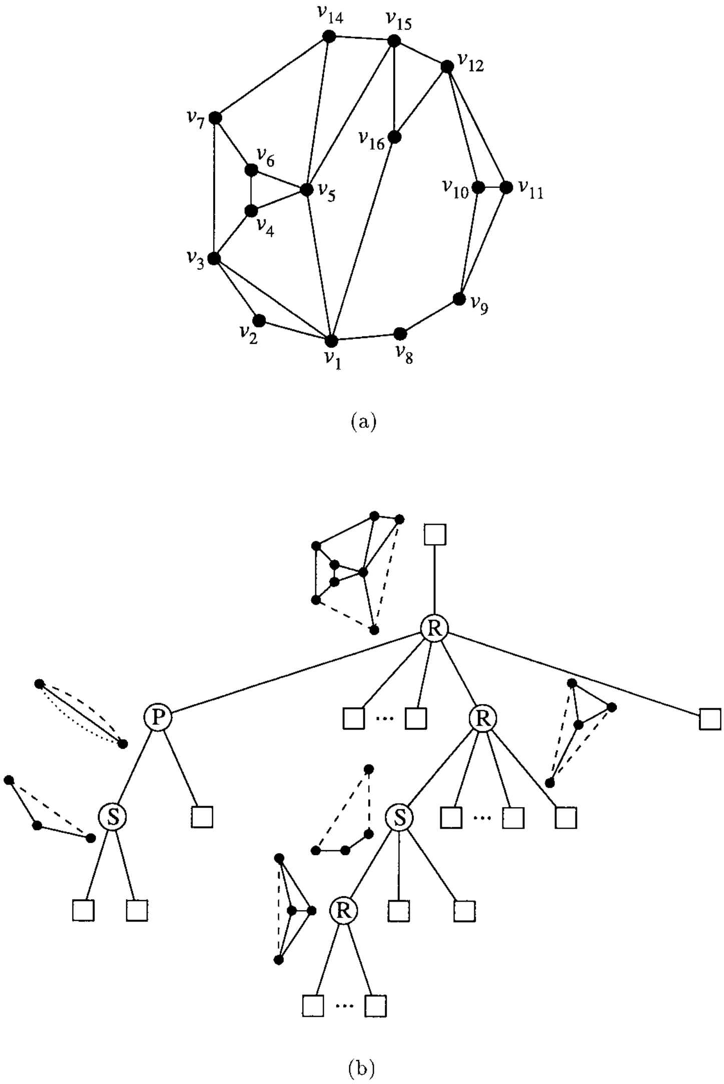


FIG. 6. (a) A strictly convex drawing of a reduced graph G' of the biconnected planar graph in Fig. 1a. (b) The SPQR-tree of G' with respect to reference edge (v_3, v_7) and the skeletons of its non-Q-nodes.

- There exist three (u, v)-chains. Since at most two (u, v)-chains can be drawn outside in Γ_c, one of the three (u, v)-chains is drawn "between" the other two, that is, inside in Γ_c. Let C be such a (u, v)-chain.

We replace C with bypass edge (u, v), drawn as a straight-line segment. We now show that the convex planarity of the drawing is not affected by this modification. From the discussion above, two cases are possible:

- C is drawn inside in Γ_c. Then, as seen in the proof of Lemma 3, the vertices of C are placed on a straight-line segment.
- C is drawn outside in Γ_c. Then there is no (u, v)-chain drawn inside in Γ_c. Let l be the straight-line through the points representing u and v. Since Γ_c is convex and there are no 180° angles around vertices of degree greater than 2, the vertices and edges of C are on one side of l (or possibly on l), while the vertices and edges of $G - C$ are on the other side.

In both cases, bypass edge (u, v) does not overlap any vertex or edge of $G - C$, and the replacement of C with bypass edge (u, v) does not alter the convexity of the drawing.

The overall result of the modification of Γ_c is a convex drawing Γ'_c of G'. There may still be 180° angles around vertices of degree 2 that are external in Γ'_c. A strictly convex drawing Γ'_{sc} of G' can be obtained from Γ'_c by local adjustment at those vertices.

If. Let Γ'_{sc} be a strictly convex drawing of G'. A convex drawing of G can be obtained from Γ'_{sc} by replacing each bypass edge (u, v) with the corresponding (u, v)-chain, drawn by placing the vertices on a straight-line segment. ■

COROLLARY 4. *The convex planarity of an n-vertex biconnected planar graph can be tested in $O(n)$ time.*

Proof. Let G be an n-vertex biconnected planar graph and let G' be a reduced graph of G. Computing the triconnected components of G takes $O(n)$ time [31]. The triconnected components of G' can be computed from those of G as follows. We consider each polygon triconnected component C of G with only one virtual edge e; $C - e$ is a (u, v)-chain of G. If the triconnected component C_e of G associated with e is either a triconnected simple planar graph or a bond consisting only of virtual edges, then C is not a triconnected component of G', and the graph obtained from C_e by replacing the virtual edge corresponding to C with a (nonvirtual) bypass edge is a triconnected component of G'. All the other triconnected components of G are also triconnected components of G'. Thus, computing the triconnected components of G' takes $O(n)$ time. The claim follows from Corollary 2 and from Theorem 3. ■

It is easy to verify that the SPQR-tree in Fig. 6b satisfies the condition of Theorem 2. Hence, the graph in Fig. 1a, of which the graph in Fig. 6a is a reduced graph, is convex planar. Consider, instead, the SPQR-tree in Fig. 3b. Since the skeleton of R-node μ contains no chain virtual edge, it is not modified in the construction of the SPQR-tree of a reduced graph of the biconnected planar graph in Fig. 3a. Hence, as shown before, the condition of Theorem 2 is not satisfied, and the graph in Fig. 3a is not convex planar.

4. REPERTORY OF QUERY AND UPDATE OPERATIONS

In the rest of the paper, we consider an incremental environment where a biconnected planar graph G is updated by on-line insertions of vertices and edges that preserve planarity. We recall that in an on-line dynamic graph problem the sequence of operations is not known in advance. The repertory of query and update operations extends that given for biconnected planar graphs in [20]:

Strictly Convex: Determine whether G is strictly convex planar.

Convex: Determine whether G is convex planar.

Test (v_1, v_2): Determine whether edge (v_1, v_2) can be added to G while preserving planarity. As a particular case, the result of the query is *false* if edge (v_1, v_2) already exists.

Insert Vertex (v, e, e_1, e_2): Split edge e of G into two edges e_1 and e_2 by inserting vertex v.

Insert Edge (e, v_1, v_2): Add edge e between vertices v_1 and v_2 of G. The operation is allowed only if the resulting graph is planar.

As shown in [20], an n-vertex biconnected planar graph can be assembled starting from a three-vertex cycle by means of a sequence of $O(n)$ *InsertVertex* and *InsertEdge* operations, such that each intermediate graph is planar and biconnected.

As stated in the Introduction, the (strictly) convex planarity property for planar graphs is not monotone: there exist sequences of update operations from the above repertory such that the current graph alternates between being (strictly) convex planar and being nonconvex. One such sequence of operations is shown in Fig. 7. Let G be the strictly convex planar graph in Fig. 7a. The first operation of the sequence is *InsertEdge* (e_1, u, v), after which G is still strictly convex planar (see Fig. 7b). The second operation is *InsertVertex* (x, e_1, e'_1, e''_1), after which G is no longer strictly convex planar but is convex planar (see Fig. 7c). In fact, u and v are the poles of a P-node whose skeleton has three (multiple) nontrivial virtual edges; thus, the condition of Theorem 2 is no longer true. After the third operation, *InsertEdge*(e_2, u, v), G is no longer convex planar (see Fig. 7d). In fact, $\{u, v\}$ is now a trivial split pair and the only (u, v)-chain of G cannot be replaced with a bypass edge; thus, the reduced graph of G is G itself, and the condition of Theorem 3 is no longer true. Finally, after operation *InsertEdge*(e_3, w, x), G is strictly convex planar again (see Fig. 7e). In contrast, note that, in an incremental environment, the nonplanarity property for

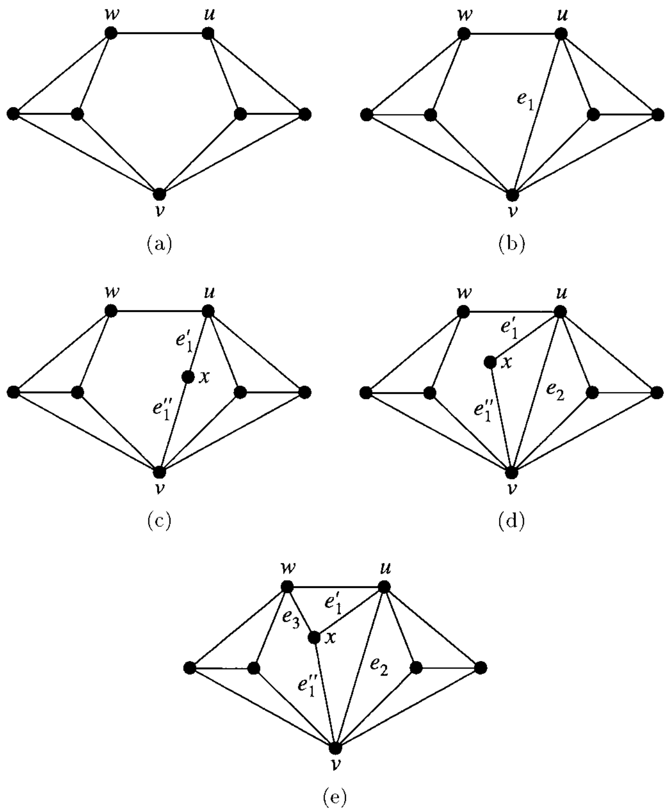


FIG. 7. A sequence of *InsertEdge* and *InsertVertex* operations in a biconnected planar graph G such that: (a, b) G is strictly convex planar, (c) G is convex planar, (d) G is not convex planar, and (e) G is strictly convex planar.

graphs is monotone: should the graph be allowed to become nonplanar as a result of an *InsertEdge* operation, it could not become planar again as a result of an update operation from the above repertory.

5. DATA STRUCTURE

The data structure for on-line incremental planarity testing described in [20] makes use of the dynamic trees of Sleator and Tarjan [42, 43] in order to maintain information about the SPQR-tree. These dynamic trees support link/cut operations and various queries (such as finding the lowest common ancestor of two nodes) in logarithmic time, and they can be modified to support ordered trees and expand/contract operations, as shown in [27, 28]. Our data structure for on-line incremental convex planarity testing extends that described in [20]. In particular, we add the following data structures, which we use in the implementation of query operations *StrictlyConvex* and *Convex* (see Section 6):

- For each P-node μ of T:

—A variable

$$P3nontrivial(\mu) = \begin{cases} 0 & \text{if } skeleton(\mu) \text{ consists of one trivial virtual edge and} \\ & \quad \text{two nontrivial virtual edges (see Fig. 8a)} \\ 1 & \text{otherwise (see Figs. 8b and 8c).} \end{cases}$$

Value 0 of *P3nontrivial* (μ) indicates that there exists an embedding of *skeleton*(μ) such that all the nontrivial virtual edges are on the same face.

—A variable

$$P3nonchain(\mu) = \begin{cases} 0 & \text{if } P3nontrivial(\mu) = 0 \text{ or if } skeleton(\mu) \text{ consists of three} \\ & \quad \text{nontrivial virtual edges, at least one of which is a chain} \\ & \quad \text{virtual edge (see Figs. 8a and 8b)} \\ 1 & \text{otherwise (see Fig. 8c).} \end{cases}$$

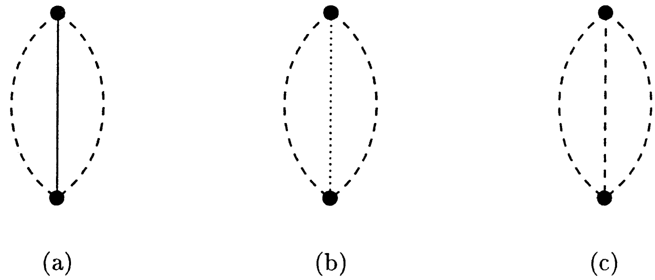


FIG. 8. Three skeletons of P-nodes consisting of: (a) one trivial virtual edge and two nontrivial virtual edges, (b) three nontrivial virtual edges, one of which is a chain virtual edge, and (c) three nontrivial virtual edges.

Value 0 of *P3nonchain*(μ) indicates that there exists an embedding of *skeleton*(μ) such that all the nontrivial virtual edges, with the exception of at most one chain virtual edge, are on the same face.

- For each S-node μ of T:

—For an arbitrarily chosen face f of *skeleton*(μ) (recall that the skeleton of an S-node is a cycle), a balanced binary tree $B_S(\mu)$, where each leaf of $B_S(\mu)$ corresponds to an edge e of f, and stores value *nontrivial*(e), which is 0 or 1 according to whether e is a trivial or nontrivial virtual edge (see Fig. 9b). Each internal node of $B_S(\mu)$ stores the sum of the values of the leaves in its subtree (see Fig. 9b). Hence, the root of $B_S(\mu)$ stores the number of nontrivial virtual edges of *skeleton*(μ), denoted *Snontrivial*(μ) (see Fig. 9b). The edges of f are circularly ordered so that, if f is traversed according to this order, the region bounded by f is, say, on the left side. The circular order of the edges of f is represented by the left-to-right linear order of the leaves of $B_S(\mu)$. In particular, note that:

∗ *Snontrivial*(μ) $= 0$ if and only if G is a cycle, and thus μ is the only non-Q-node of T;

∗ *Snontrivial*(μ) $= 1$ if and only if the edges of G corresponding to the trivial virtual edges of *skeleton*(μ) form a (u, v)-chain of G.

For each non-Q-node ν adjacent to μ, variable *Snontrivial*(μ) allows us to test in $O(1)$ time whether nontrivial virtual edge e_μ of *skeleton*(ν) is a chain virtual edge.

- For each R-node μ of T:

—For each face f of *skeleton*(μ) (recall that the embedding of the skeleton of an R-node is unique), a balanced binary tree $B_R(f)$, where each leaf of $B_R(f)$ corresponds to an edge e of f, and stores two values (see Fig. 10b): *nontrivial*(e), which is 0 or 1 according to whether e is a trivial or nontrivial virtual edge, and *chain*(e), which is 1 or 0 according to whether e is or is not a chain virtual edge. Each internal node of $B_R(f)$ stores two values (see Fig. 10b):

1. the sum of the *nontrivial*(e) values of the leaves in its subtree; and
2. the sum of the *chain*(e) values of the leaves in its subtree.

Hence, the root of $B_R(f)$ stores two values (see Fig. 10b):

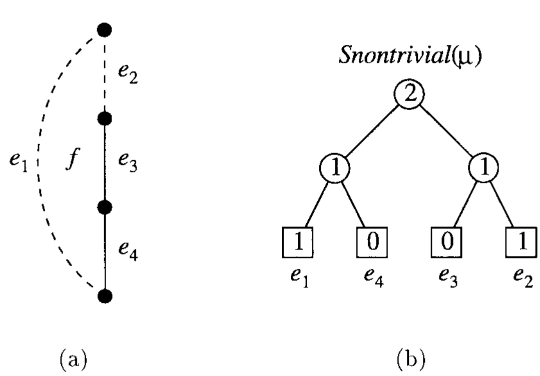


FIG. 9. (a) The skeleton of an S-node μ. (b) The balanced binary tree for μ.

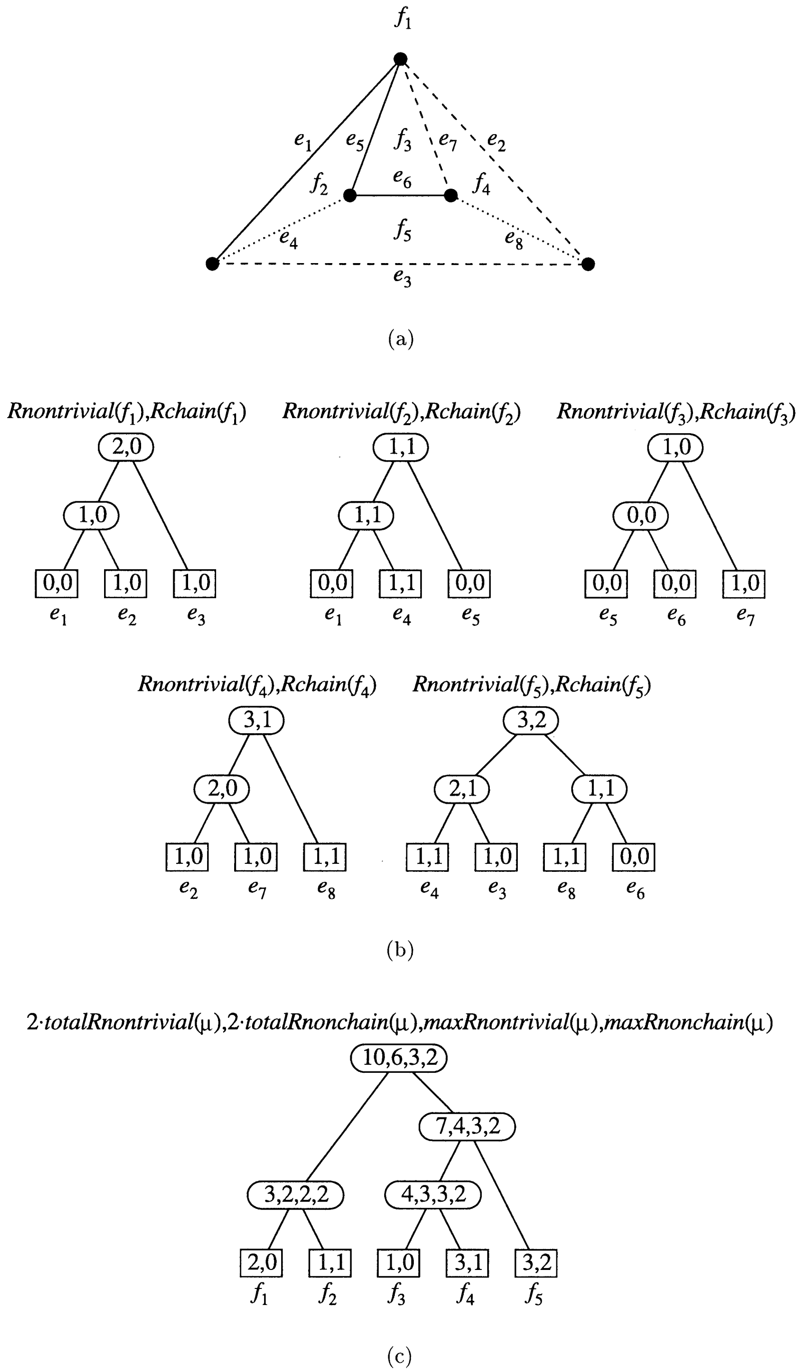


FIG. 10. (a) The skeleton of an R-node μ. (b) The balanced binary trees for the faces of *skeleton*(μ). (c) The balanced binary tree for μ.

1. the number of nontrivial virtual edges of f, denoted $\mathit{Rnontrivial}(f) = \sum_e \mathit{nontrivial}(e)$; and

2. the number of chain virtual edges of f, denoted $\mathit{Rchain}(f) = \sum_e \mathit{chain}(e)$; note that $\mathit{Rchain}(f) \leq \mathit{Rnontrivial}(f)$.

The edges of f are circularly ordered so that, if f is traversed according to this order, the region bounded by f is, say, on the left side. The circular order of the edges of f is represented by the left-to-right linear order of the leaves of $B_R(f)$.

—A balanced binary tree $B_{\mathrm{R}}(\mu)$ associated with μ, where each leaf of $B_{\mathrm{R}}(\mu)$ corresponds to a face f of μ and stores *Rnontrivial*(f) and *Rchain*(f) (see Fig. 10c). Each internal node of $B_{\mathrm{R}}(\mu)$ stores four values (see Fig. 10c):

1. the sum of the *Rnontrivial*(f) values of the leaves in its subtree;
2. the sum of the *Rnontrivial*(f) − *Rchain*(f) values of the leaves in its subtree;
3. the maximum *Rnontrivial*(f) value of the leaves in its subtree; and
4. the maximum *Rnontrivial*(f) − *Rchain*(f) value of the leaves in its subtree.

Hence, the root of $B_{\mathrm{R}}(\mu)$ stores four values (see Fig. 10c):

1. two times the total number of nontrivial virtual edges in *skeleton*(μ), this last denoted $\mathit{totalRnontrivial}(\mu) = \frac{1}{2}\sum_f \mathit{Rnontrivial}(f)$;
2. two times the total number of nontrivial virtual edges that are not chain virtual edges in *skeleton*(μ), this last denoted $\mathit{totalRnonchain}(\mu) = \frac{1}{2}\sum_f (\mathit{Rnontrivial}(f) - \mathit{Rchain}(f))$; note that $\mathit{totalRnonchain}(\mu) \geq 0$;
3. the maximum value of *Rnontrivial*(f) over all faces f of *skeleton*(μ), denoted $\mathit{maxRnontrivial}(\mu) = \max_f\{\mathit{Rnontrivial}(f)\}$; and
4. the maximum value of *Rnontrivial*(f) − *Rchain*(f) over all faces f of *skeleton*(μ), denoted $\mathit{maxRnonchain}(\mu) = \max_f\{\mathit{Rnontrivial}(f) - \mathit{Rchain}(f)\}$; note that $\mathit{maxRnonchain}(\mu) \geq 0$.

The purpose of the above four variables is the following: $\mathit{totalRnontrivial}(\mu) = \mathit{maxRnontrivial}(\mu)$ indicates that, in the unique embedding of *skeleton*(μ), all the nontrivial virtual edges of *skeleton*(μ) are on the same face; $\mathit{maxRnontrivial}(\mu) = \mathit{maxRnonchain}(\mu)$ indicates that, in the unique embedding of *skeleton*(μ), all the nontrivial virtual edges of *skeleton*(μ) that are not chain virtual edges are on the same face.

- For the entire graph G, the following variables are obtained by summing those above over all the P-nodes or all the R-nodes of T:

—the number of P-nodes of T whose skeleton contains more than two nontrivial virtual edges, denoted

$$\mathit{sumP3nontrivial}(G) = \sum_{\text{P-node } \mu} \mathit{P3nontrivial}(\mu);$$

—the number of P-nodes of T whose skeleton contains more than two nonchain, nontrivial virtual edges, denoted

$$\mathit{sumP3nonchain}(G) = \sum_{\text{P-node } \mu} \mathit{P3nonchain}(\mu);$$

—the total number of nontrivial virtual edges in the skeletons of the R-nodes of T, denoted

$$\mathit{sumtotalRnontrivial}(G) = \sum_{\text{R-node } \mu} \mathit{totalRnontrivial}(\mu);$$

—the total number of nonchain, nontrivial virtual edges in the skeletons of the R-nodes of T, denoted

$$\mathit{sumtotalRnonchain}(G) = \sum_{\text{R-node } \mu} \mathit{totalRnonchain}(\mu);$$

—the sum of the *maxRnontrivial*(μ) values over all the R-nodes of T, denoted

$$\mathit{summaxRnontrival}(G) = \sum_{\text{R-node } \mu} \mathit{maxRnontrivial}(\mu);$$

TABLE 1

The Values of Some of the Additional Variables for the Graphs in Figs. 1, 2, and 3

	Fig. 1	Fig. 2	Fig. 3
P3nontrivial(π)	1	0	0
P3nonchain(π)	0	0	0
Snontrivial(ρ)	2	2	2
totalRnontrivial(μ)	3	2	3
totalRnonchain(μ)	2	2	3
maxRnontrivial(μ)	2	2	2
maxRnonchain(μ)	2	2	2
sumP3nontrivial(G)	1	0	0
sumP3nonchain(G)	0	0	0
sumtotalRnontrivial(G)	7	6	8
sumtotalRnonchain(G)	5	5	7
summaxRnontrivial(G)	6	6	7
summaxRnonchain(G)	5	5	6

—the sum of the *maxRnonchain*(μ) values over all the R-nodes of T, denoted

$$summaxRnonchain(G) = \sum_{\text{R-node } \mu} maxRnonchain(\mu).$$

As an example, in Table 1 we give the values of some of the above variables for the graphs in Figs. 1, 2, and 3.

6. IMPLEMENTATION OF THE QUERY OPERATIONS

In this section, we describe the implementation of operations *StrictlyConvex* and *Convex*. As for operation *Test*, it does not use any of the additional data structures and thus it is implemented exactly as described in [20].

In the implementation of operation *StrictlyConvex*, we use three of the six variables for the entire graph described in Section 5. Namely, operation *StrictlyConvex* is implemented as the logical *and* of the following two conditions:

1. $sumP3nontrivial(G) = 0$; and
2. $sumtotalRnontrivial(G) = summaxRnontrivial(G)$.

LEMMA 5. *The above implementation of operation StrictlyConvex is correct.*

Proof. Condition 1 holds if and only if $P3nontrivial(\mu) = 0$ for each P-node μ of T: necessity can be proved by contradiction; sufficiency is trivial. It follows that Condition 1 expresses the fact that for every P-node μ of T, *skeleton*(μ) consists of one trivial and two nontrivial virtual edges.

For each R-node μ of T, $totalRnontrivial(\mu) \geq maxRnontrivial(\mu)$, where equality holds if and only if all the nontrivial virtual edges of *skeleton*(μ) are on the same face. Thus, for G, $sumtotalRnontrivial(G) \geq summaxRnontrivial(G)$. Condition 2 holds if and only if $totalRnontrivial(\mu) = maxRnontrivial(\mu)$ for each R-node μ of T: necessity can be proved by contradiction; sufficiency is trivial. It follows that Condition 2 expresses the fact that, for each R-node μ of T, all the nontrivial virtual edges of *skeleton*(μ) are on the same face.

Thus, the logical *and* of Conditions 1 and 2 is equivalent to Theorem 2. ■

In the implementation of operation *Convex*, we use the other three variables for the entire graph described in Section 5. Namely, operation *Convex* is implemented as the logical *and* of the following two conditions:

1. $sumP3nonchain(G) = 0$; and
2. $sumtotalRnonchain(G) = summaxRnonchain(G)$.

LEMMA 6. *Let G be a biconnected planar graph and let G' be a reduced graph of G. Then* *sumP3nonchain*$(G) = 0$ *if and only if* *sumP3nontrivial*$(G') = 0$.

Proof. Let T and T' be the SPQR-trees of G and G', respectively. Condition *sumP3nonchain*$(G) = 0$ holds if and only if *P3nonchain*$(\mu) = 0$ for every P-node μ of T, and condition *sumP3nontrivial*$(G') = 0$ holds if and only if *P3nontrivial*$(\mu') = 0$ for every P-node μ' of T': necessity can be proved by contradiction; sufficiency is trivial.

Thus, to prove the claim, it is sufficient to prove that, for each P-node μ of T, *P3nonchain*$(\mu) = 0$ if and only if *P3nontrivial*$(\mu') = 0$, where μ' is the node of T' corresponding to μ.

As observed in Section 3, *skeleton*(μ') is obtained from *skeleton*(μ) by replacing at most one chain virtual edge with a trivial virtual edge. In particular, three cases are possible: (*i*) *skeleton*(μ) consists of one trivial and two nontrivial virtual edges; then *skeleton*$(\mu') =$ *skeleton*(μ) and *P3nonchain*$(\mu) =$ *P3nontrivial*$(\mu') = 0$; (*ii*) *skeleton*(μ) consists of three nontrivial virtual edges, at least one of which is a chain virtual edge; then *skeleton*(μ') consists of one trivial and two nontrivial virtual edges, and *P3nonchain*$(\mu) =$ *P3nontrivial*$(\mu') = 0$; (*iii*) *skeleton*(μ) consists of more than three virtual edges; then also *skeleton*(μ') consists of more than three virtual edges and *P3nonchain*$(\mu) =$ *P3nontrivial*$(\mu') = 1$. Hence the claim is proved. ■

LEMMA 7. *Let G be a biconnected planar graph and let G' be a reduced graph of G. Then,* *sumtotalRnonchain*$(G) =$ *summaxRnonchain*(G) *if and only if* *sumtotalRnontrivial*$(G') =$ *summaxRnontrivial*(G').

Proof. Let T and T' be the SPQR-trees of G and G', respectively. For each R-node μ of T, *totalRnonchain*$(\mu) \geq$ *maxRnonchain*(μ), where equality holds if and only if all the nonchain, nontrivial virtual edges of *skeleton*(μ) are on the same face. It follows that *sumtotalRnonchain*$(G) \geq$ *summaxRnonchain*(G), where equality holds if and only if *totalRnonchain*$(\mu) =$ *maxRnonchain*(μ) for every R-node μ of T: necessity can be proved by contradiction; sufficiency is trivial. Similarly, for each R-node μ' of T', *totalRnontrivial*$(\mu') \geq$ *maxRnontrivial*(μ'), where equality holds if and only if all the nontrivial virtual edges of *skeleton*(μ') are on the same face. It follows that *sumtotalRnontrivial*$(G') \geq$ *summaxRnontrivial*(G'), where equality holds if and only if *totalRnontrivial*$(\mu') =$ *maxRnontrivial*(μ') for every R-node μ' of T': again, necessity can be proved by contradiction; sufficiency is trivial.

Thus, to prove the claim, it is sufficient to prove that, for each R-node μ of T, *totalRnonchain*$(\mu) =$ *maxRnonchain*(μ) if and only if *totalRnontrivial*$(\mu') =$ *maxRnontrivial*(μ'), where μ' is the node of T' corresponding to μ.

As observed in Section 3, *skeleton*(μ') is obtained from *skeleton*(μ) by replacing each chain virtual edge with a trivial virtual edge. It follows that *totalRnonchain*$(\mu) =$ *totalRnontrivial*(μ') and *maxRnonchain*$(\mu) =$ *maxRnontrivial*(μ'). Hence the claim is proved. ■

LEMMA 8. *The above implementation of operation Convex is correct.*

Proof. It immediately follows from Lemmas 4, 5, 6, and 7, and from Theorem 3. ■

7. IMPLEMENTATION OF THE UPDATE OPERATIONS

In the description of operations *InsertVertex* and *InsertEdge*, we use the terminology and concepts of [20]. In particular, for each update operation, we recall the structural changes of the SPQR-tree, and describe in detail how the additional data structures are modified.

We adopt a top-down approach by defining a hierarchy of transformations. A pseudocode description of operation *InsertEdge* is given (see Algorithm 1), based on the following transformations: *FinalTransformation1*, *InitialTransformation*, *ElementaryTransformation*, *FinalTransformation2*, and *FinalTransformation3*. The first, third, and fourth of these transformations, plus operation *InsertVertex*, are described in terms of *X-transformations* or *RX-transformations*, where X is R, P, or S, depending on whether a specified node is an R-node, P-node, or S-node, respectively. In turn, the *X-transformations* and *RX-transformations* relative to operation *InsertEdge*, and *InitialTransformation* are described in terms of two auxiliary operations, called *SplitFace* and *MergeFaces*.

We describe here, once and for all, certain updates of the additional data structures that occur in all the transformations:

- For each R-node μ, every time one of the values stored at the root of the balanced binary tree $B_R(f)$ associated with a face f of *skeleton*(μ) changes, the same value stored at the leaf of $B_R(\mu)$ corresponding to f is updated.
- For each P-node μ, every time *P3nontrivial*(μ) or *P3nonchain*(μ) changes, *sumP3nontrivial*(G) or *sumP3nonchain*(G) is updated, respectively.
- For each R-node μ, every time *totalRnontrivial*(μ), *totalRnonchain*(μ), *maxRnontrivial*(μ), or *maxRnonchain*(μ) changes, *sumtotalRnontrivial*(G), *sumtotalRnonchain*(G), *summaxRnontrivial*(G), or *summaxRnonchain*(G) is updated, respectively.

All the additional data structures not explicitly mentioned in the various transformations are assumed to remain unchanged.

Finally, we have a notational remark. When a face f is split by operation *SplitFace*, the two resulting faces are denoted f' and f''. When two faces f_x and f_y are merged by operation *MergeFaces*, the resulting face is denoted f_{xy}.

7.1. Insert Vertex

In this section we consider operation *InsertVertex*(v, e, e_1, e_2). Let ρ be the Q-node corresponding to e and let π be the node adjacent to ρ. Node π can be either an R-node, a P-node, or an S-node; three different cases are possible for *InsertVertex*(v, e, e_1, e_2), respectively:

1. *R-transformation*. Node ρ is replaced with an S-node λ having two adjacent Q-nodes, ρ_1 and ρ_2, corresponding to e_1 and e_2, respectively. The trivial virtual edge e_ρ in *skeleton*(π) is replaced with a nontrivial virtual edge e_λ.

We create a new balanced binary tree $B_S(\lambda)$ with three leaves, and we set *nontrivial*(e_{ρ_1}) and *nontrivial*(e_{ρ_2}) equal to 0, and *nontrivial*(e_π) equal to 1.

Let f_1 and f_2 be the two faces of *skeleton*(π) containing e_ρ, now renamed e_λ. We set both *nontrivial*(e_λ) and *chain*(e_λ) equal to 1 in the two leaves of $B_R(f_1)$ and $B_R(f_2)$ corresponding to e_λ.

2. *P-transformation*. Node ρ is replaced with an S-node λ having two adjacent Q-nodes, ρ_1 and ρ_2, corresponding to e_1 and e_2, respectively. The trivial virtual edge e_ρ in *skeleton*(π) is replaced with a nontrivial virtual edge e_λ.

We create a new balanced binary tree $B_S(\lambda)$ with three leaves, and we set *nontrivial*(e_{ρ_1}) and *nontrivial*(e_{ρ_2}) equal to 0, and *nontrivial*(e_π) equal to 1.

If, before the transformation, *P3nontrivial*(π) $=$ 0 (and thus *P3nonchain*(π) $=$ 0), we set *P3nontrivial*(π) equal to 1 and leave *P3nonchain*(π) equal to 0. (Note that, being G simple, the skeleton of a P-node may contain at most one trivial virtual edge, while the other virtual edges are nontrivial.)

3. *S-transformation*. Node ρ is replaced with two Q-nodes, ρ_1 and ρ_2, corresponding to e_1 and e_2, respectively. The trivial virtual edge e_ρ in *skeleton*(π) is replaced with two trivial virtual edges, e_{ρ_1} and e_{ρ_2}, having an endvertex in common.

We delete the leaf of $B_S(\pi)$ corresponding to e_ρ and insert two new leaves corresponding to e_{ρ_1} and e_{ρ_2}. We set nontrivial(e_{ρ_1}) and nontrivial(e_{ρ_2}) equal to 0 in these two leaves.

The above discussion on the various transformations in operation *InsertVertex* can be summarized in the following lemma.

LEMMA 9. *The transformations in operation InsertVertex require:*

- *the creation of* $O(1)$ *balanced binary trees, each with an* $O(1)$ *number of leaves*;
- *the execution of* $O(1)$ *insert and delete operations on a balanced binary tree*; *and*
- *the update of* $O(1)$ *values stored either at a leaf of a balanced binary tree or in a variable.*

7.2. InsertEdge

In this section we consider operation *InsertEdge*(e, v_1, v_2). In order to describe the corresponding transformations of the SPQR-tree T of graph G, we need some more definitions. Let v be a vertex of G. The *allocation nodes* of v are the nodes of T whose skeleton contains v. The lowest common ancestor of the allocation nodes of v is itself an allocation node of v and is called the *proper* allocation node of v, denoted *proper*(v). If v is one of the endvertices of the reference edge, we conventionally define *proper*(v) as the unique child of the root of T. In all other cases, *proper*(v) is either an R-node or an S-node; also, *proper*(v) is the only allocation node μ of v such that v is not a pole of μ. As an example, in Fig. 1 R-nodes χ and μ, P-node π, and S-nodes σ and ρ are all allocation nodes of vertex v_1, with χ as the proper allocation node. R-node χ is also, by convention, the proper allocation node of vertex v_7.

In Algorithm 1 we recall the pseudo-code description of operation *InsertEdge*(e, v_1, v_2) from Section 5 of [20]. The proper allocation nodes μ_1 of v_1 and μ_2 and v_2, and their lowest common ancestor μ are computed. Four cases are possible: the three nodes are coincident, the three nodes are distinct, or one proper allocation node is an ancestor of the other (two cases). In all four cases, the subtree T_μ of T rooted at μ and the corresponding additional data structures are subject to some transformations. We describe these transformations in the rest of the section.

7.2.1. *FinalTransformation1* (χ)

From Algorithm 1, it follows that *skeleton*(χ) contains both v_1 and v_2. As described in Section 5 of [20], v_1 and v_2 belong to a common face f, and χ can be either an R-node or an S-node; two different cases are possible for *FinalTransformation1* (χ), respectively:

ALGORITHM 1. Operation *InsertEdge*(e, v_1, v_2) and its subroutine *PathCondensation*(μ_i, χ) *InsertEdge*(e, v_1, v_2)

begin
  find the proper allocation nodes μ_1 of v_1 and μ_2 of v_2, and their lowest common ancestor μ;
  case of
  $\mu_1 = \mu = \mu_2$:
    FinalTransformation1(μ);
  $\mu_1 \neq \mu \neq \mu_2$:
    PathCondensation(μ_1, μ);
    PathCondensation(μ_2, μ);
    FinalTransformation2(μ_1, μ_2);
  $\mu_1 = \mu \neq \mu_2$:
    determine the lowest node ω on the path from μ_2 to μ such that *skeleton*(ω) contains v_1;
    if $\omega = \mu_2$ **then**
      FinalTransformation1(μ_2);
    else
      PathCondensation(μ_2, ω);
      FinalTransformation3(μ_2);
    endif
  $\mu_1 \neq \mu = \mu_2$:
    {this case is analogous to the previous one and therefore omitted}
  endcase
end

PathCondensation(μ_i, χ)
begin
  InitialTransformation(μ_i);
  find the child λ_i of χ on the path from μ_i to χ;
  set ρ equal to μ_i;
  while $\rho \neq \lambda_i$ **do**   {μ_i "bubbles up" along T until it becomes a child of χ}
    set π equal to the parent of ρ;
    ElementaryTransformation(ρ, π);
    set ρ equal to π;
  endwhile
end

1. *R-transformation*. Two cases are possible:

(a) *skeleton*(χ) does not contain edge (v_1, v_2). A new Q-node, corresponding to edge e, is added as a child of χ, and a trivial virtual edge (v_1, v_2) is added to *skeleton*(χ), splitting face f into faces f' and f''.

We perform operation *SplitFace*($B_R(f)$, v_1, v_2, *trivial*) obtaining $B_R(f')$ and $B_R(f'')$. We delete the leaf of $B_R(\chi)$ corresponding to f and insert two new leaves corresponding to f' and f''.

(b) *skeleton*(χ) contains edge (v_1, v_2). Then (v_1, v_2) is the nontrivial virtual edge of a child ν of χ, and two cases are possible:

i. ν is a P-node. A new Q-node, corresponding to edge e, is added as a child of ν, and a trivial virtual edge (v_1, v_2) is added to *skeleton*(ν).

If, before the transformation, *P3nonchain*(ν) is equal to 0, we set it equal to 1. (Note that, before the transformation, *P3nontrivial*(ν) is equal to 1 since *skeleton*(ν) does not contain a trivial virtual edge (v_1, v_2).)

ii. ν is not a P-node. It is replaced with a new P-node λ, whose children are ν and a new Q-node ρ, corresponding to edge e; *skeleton*(λ) consists of the nontrivial virtual edges e_ν and e_χ and of the trivial virtual edge e_ρ.

We set both *P3nontrivial*(λ) and *P3nonchain*(λ) equal to 0.

Let f_1 and f_2 be the two faces of *skeleton*(χ) containing e_ν, now renamed e_λ. We set *nontrivial*(e_λ) equal to 1 and *chain*(e_λ) equal to 0 in the two leaves of $B_R(f_1)$ and $B_R(f_2)$ corresponding to e_λ.

If ν is an S-node, we consider the leaf of $B_S(\nu)$ corresponding to e_χ, now renamed e_λ. We set *nontrivial*(e_λ) equal to 1 in this leaf.

If ν is an R-node, let f_a and f_b be the two faces of *skeleton*(ν) containing e_χ, now renamed e_λ. We set *nontrivial*(e_λ) equal to 1 and *chain*(e_λ) equal to 0 in the two leaves of $B_R(f_a)$ and $B_R(f_b)$ corresponding to e_λ.

2. *S-transformation*. Two cases are possible:

(a) *skeleton*(χ) does not contain edge (v_1, v_2). Let σ be the parent of χ, let p be the path of *skeleton*(χ) between v_1 and v_2 not containing e_σ (see Fig. 11a), and let $\beta_1, \ldots, \beta_k$, $k \geq 2$, be the children of χ corresponding to the edges of p. Nodes $\beta_1, \ldots, \beta_k$ are replaced with a new P-node λ whose children are a new Q-node ρ, corresponding to edge e, and a new S-node ν, whose children are $\beta_1, \ldots, \beta_k$. Path p is replaced in *skeleton*(χ) with the nontrivial virtual edge e_λ; *skeleton*(λ) consists of the nontrivial virtual edges e_χ and e_ν, and of the trivial virtual edge e_ρ; *skeleton*(ν) consists of p plus a nontrivial virtual edges $e_\lambda = (v_1, v_2)$ (see Fig. 11b).

We set both *P3nontrivial*(λ) and *P3nonchain*(λ) equal to 0.

We perform operation *SplitFace*($B_S(\chi)$, v_1, v_2, *nontrivial*) obtaining $B_S(\nu)$ and the new $B_S(\chi)$. We consider the leaf of $B_S(\chi)$ corresponding to e_ν, now renamed e_λ. We set *nontrivial*(e_λ) equal to 1 in this leaf. We then consider the leaf of $B_S(\nu)$ corresponding to e_χ, now renamed e_λ. We set *nontrivial*(e_λ) equal to 1 in this leaf.

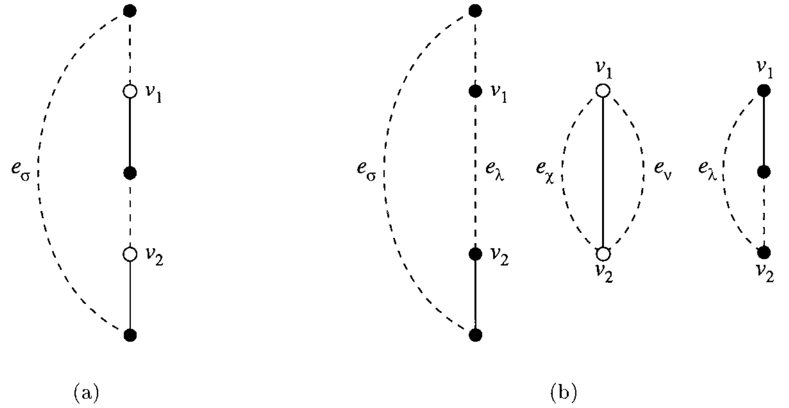


FIG. 11. An example of *S-transformation* in *FinalTransformation1*: (a) *skeleton*(χ) before the *S-transformation*, and (b) *skeleton*(χ), *skeleton*(λ), and *skeleton*(ν), after the *S-transformation*.

If σ is a P-node whose skeleton consists of e_χ and two other virtual edges e_ξ and e_ψ, and e_ξ and e_ψ are neither chain virtual edges (*Snontrivial*$(\xi) > 1$ and *Snontrivial*$(\psi) > 1$) nor trivial virtual edges, then we set *P3nonchain*(σ) equal to 1.

(b) *skeleton*(χ) contains edge (v_1, v_2). Analogous to the second case of the *R-transformation*.

7.2.2. *InitialTransformation*(μ_i)

If μ_i is an S-node, it is transformed into an R-node. Let σ be the parent of μ_i; note that σ is neither an S-node, since two S-nodes cannot be adjacent in T, nor a Q-node, since μ_i, having at least μ as an ancestor (see Algorithm 1), cannot be the child of the root of T.

If s_{μ_i} and v_i are not adjacent in *skeleton*(μ_i), let p_s be the path of *skeleton*(μ_i) between s_{μ_i} and v_i not containing e_σ (see Fig. 12a), and let $\alpha_1, \ldots, \alpha_k$, $k \geq 2$, be the children of μ_i corresponding to the edges of p_s. Nodes $\alpha_1, \ldots, \alpha_k$ are replaced with a new S-node ν' whose children are $\alpha_1, \ldots, \alpha_k$. Path p_s is replaced in *skeleton*(μ_i) with the nontrivial virtual edge $e_{\nu'}$; *skeleton*(ν') consists of p_s plus a nontrivial virtual edge $e_{\mu_i} = (s_{\mu_i}, v_i)$ (see Fig. 12b).

We perform operation *SplitFace*$(B_S(\mu_i), s_{\mu_i}, v_i,$ *nontrivial*$)$ obtaining $B_S(\nu')$ and the new $B_S(\mu_i)$.

Similarly, if v_i and t_{μ_i} are not adjacent in *skeleton*(μ_i), let p_t be the path of *skeleton*(μ_i) between v_i and t_{μ_i} not containing e_σ (see Fig. 12a), and let $\gamma_1, \ldots, \gamma_h$, $h \geq 2$, be the children of μ_i corresponding to the edges of p_t. Nodes $\gamma_1, \ldots, \gamma_h$ are replaced with a new S-node ν'' whose children are $\gamma_1, \ldots, \gamma_h$. Path p_t is replaced in *skeleton*(μ_i) with the nontrivial virtual edge $e_{\nu''}$; *skeleton*(ν'') consists of p_t plus a nontrivial virtual edge $e_{\mu_i} = (v_i, t_{\mu_i})$ (see Fig. 12b).

We perform operation *SplitFace*$(B_S(\mu_i), v_i, t_{\mu_i},$ *nontrivial*$)$ obtaining $B_S(\nu'')$ and the new $B_S(\mu_i)$.

To complete the transformation, we must convert the new μ_i into an R-node. Note that μ_i will be a degenerate R-node until operation *InsertEdge* is completed, since its skeleton is not a triconnected simple planar graph, but a cycle of three virtual edges. We discard $B_S(\mu_i)$, and create two new balanced binary trees $B_R(f_1)$ and $B_R(f_2)$, with three leaves each, for the two faces f_1 and f_2 of *skeleton*(μ_i). In the leaves of both trees, we set:

- *nontrivial*$(e_\sigma) = 1$ and *chain*$(e_\sigma) = 0$
- *nontrivial*$(e_{\nu'}) = 1$ and *chain*$(e_{\nu'}) = \begin{cases} 1 & \text{if } \textit{Snontrivial}(\nu') = 1 \\ 0 & \text{otherwise} \end{cases}$
- *nontrivial*$(e_{\nu''}) = 1$ and *chain* $(e_{\nu''}) = \begin{cases} 1 & \text{if } \textit{Snontrivial}(\nu'') = 1 \\ 0 & \text{otherwise.} \end{cases}$

Finally, we create a new balanced binary tree $B_R(\mu_i)$ with two leaves corresponding to f_1 and f_2.

Note that, if σ is a P-node, the possible update of *P3nonchain*(σ) is performed either in *Elementary Transformation* or in *FinalTransformation2* (see below).

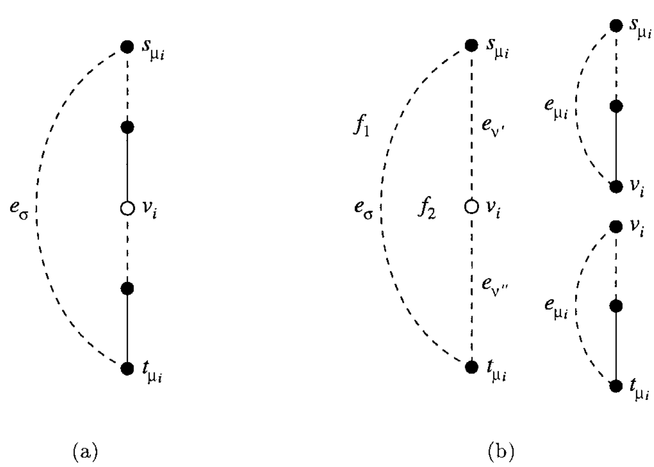


FIG. 12. An example of *InitialTransformation*: (a) *skeleton*(μ_i) before the *InitialTransformation*, and (b) *skeleton*(μ_i), *skeleton*(ν'), and *skeleton*(ν''), after the *InitialTransformation*.

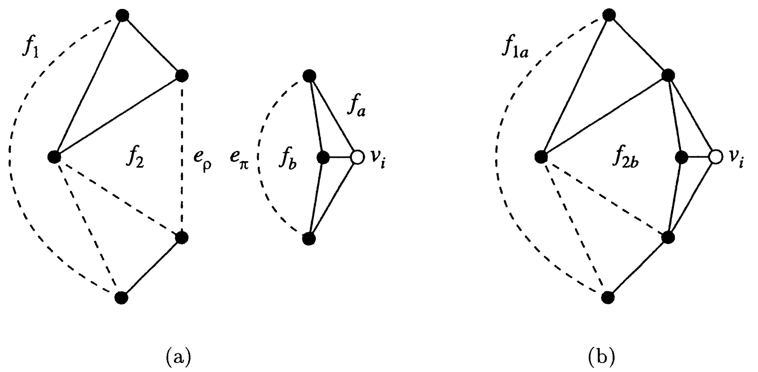


FIG. 13. An example of *RR-transformation* in *ElementaryTransformation*: (a) *skeleton*(π) and *skeleton*(ρ) before the *RR-transformation,* and (b) *skeleton*(π) after the *RR-transformation*.

7.2.3. *ElementaryTransformation*(ρ, π)

As described in Section 5 of [20], ρ is an R-node, while its parent π can be either an R-node, or a P-node, or an S-node; three different cases are possible for *ElementaryTransformation*(ρ, π), respectively:

1. *RR-transformation*. Node ρ is absorbed into node π; edge e_ρ in *skeleton*(π) is replaced with *skeleton*(ρ) $-$ e_π (see Fig. 13). Note that π will be a degenerate R-node until operation *InsertEdge* is completed, since its skeleton is not a triconnected simple planar graph, but contains a nontrivial split pair.

We first consider the balanced binary trees associated with the faces of *skeleton*(π) and *skeleton*(ρ). Let f_1 be the external face of *skeleton*(π), and let f_2 be the other face of *skeleton*(π) containing e_ρ (see Fig. 13a). Let f_a be the face of *skeleton*(ρ) containing e_π and v_i, and let f_b be the other face of *skeleton*(ρ) containing e_π (see Fig. 13a). We perform operation *MergeFaces*($B_R(f_1), e_\rho, B_R(f_a), e_\pi$), obtaining balanced binary tree $B_R(f_{1a})$ for the new face f_{1a}, and operation *MergeFaces*($B_R(f_2), e_\rho, B_R(f_b), e_\pi$), obtaining balanced binary tree $B_R(f_{2b})$ for the new face f_{2b} (see Fig. 13b).

We now consider the balanced binary trees associated with nodes π and ρ. We delete the leaves of $B_R(\pi)$ corresponding to f_1 and f_2, and the leaves of $B_R(\rho)$ corresponding to f_a and f_b; then we modify $B_R(\pi)$ by joining it with $B_R(\rho)$; and finally we insert two new leaves corresponding to f_{1a} and f_{2b} into $B_R(\pi)$.

2. *RP-transformation*. Nodes ρ and π are swapped in T. Let σ be the parent of π; edge e_σ is removed from *skeleton*(π) and inserted in *skeleton*(ρ) (see Fig. 14). If, after the swap, π has only one child ψ, node π is absorbed into node ρ, and edge e_π in *skeleton*(ρ) is replaced with e_ψ. Note that, in both cases, ρ will be a degenerate R-node until operation *InsertEdge* is completed, since its skeleton is not a triconnected simple planar graph, but contains a nontrivial split pair.

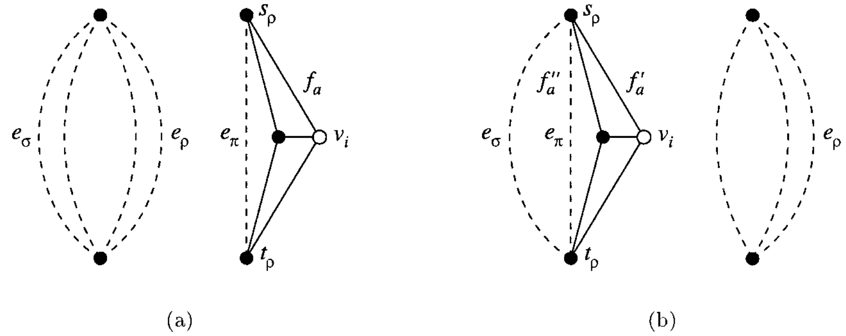


FIG. 14. An example of *RP-transformation* in *ElementaryTransformation*: (a) *skeleton*(π) and *skeleton*(ρ) before the *RP-transformation,* and (b) *skeleton*(ρ) and *skeleton*(π) after the *RP-transformation*.

We first consider the balanced binary tree associated with the face f_a of *skeleton*(ρ) containing e_π and v_i (see Fig. 14a). We perform operation *SplitFace*($B_R(f_a)$, s_ρ, t_ρ, *nontrivial*), obtaining balanced binary trees $B_R(f_a')$ and $B_R(f_a'')$ for the new faces f_a' and f_a'' into which f_a is split (see Fig. 14b).

We now consider the balanced binary tree associated with node ρ. We delete the leaf of $B_R(\rho)$ corresponding to f_a and insert two new leaves corresponding to f_a' and f_a''.

If, after the swap, π has only one child ψ, we discard *P3nontrivial*(π) and *P3nonchain*(π). Let f_1 and f_2 be the two faces of *skeleton*(ρ) containing e_π, now renamed e_ψ. We suitably set *nontrivial*(e_ψ) and *chain*(e_ψ) in the two leaves of $B_R(f_1)$ and $B_R(f_2)$ corresponding to e_ψ.

Otherwise, if, after the swap, *skeleton*(π) consists of three virtual edges, we may have to modify *P3nontrivial*(π) and *P3nonchain*(π). In particular, if *skeleton*(π) contains a trivial virtual edge, we set both *P3nontrivial*(π) and *P3nonchain*(π) equal to 0; otherwise, if *skeleton*(π) contains a chain virtual edge e_η (*Snontrivial*(η) $= 1$), we leave *P3nontrivial*(π) equal to 1 and set *P3nonchain*(π) equal to 0.

3. *RS-transformation*. Let σ be the parent of π; note that σ is neither an S-node, since two S-nodes cannot be adjacent in T, nor a Q-node, since π, having at least μ as an ancestor (see Algorithm 1), cannot be the child of the root of T.

If s_π and s_ρ are neither coincident nor adjacent in *skeleton*(π), let p_s be the path of *skeleton*(π) between s_π and s_ρ not containing e_σ (see Fig. 15a), and let $\alpha_1, \ldots, \alpha_k$, $k \geq 2$, be the children of π corresponding to the edges of p_s. Nodes $\alpha_1, \ldots, \alpha_k$ are replaced with a new S-node ν' whose children are $\alpha_1, \ldots, \alpha_k$. Path p_s is replaced in *skeleton*(π) with the nontrivial virtual edge $e_{\nu'}$; *skeleton*(ν') consists of p_s plus a nontrivial virtual edge $e_\pi = (s_\pi, s_\rho)$ (see Fig. 15b).

We perform operation *SplitFace*($B_S(\pi)$, s_π, s_ρ, *nontrivial*) obtaining $B_S(\nu')$ and the new $B_S(\pi)$.

Similarly, if t_ρ and t_π are neither coincident nor adjacent in *skeleton*(π), let p_t be the path of *skleton*(π) between t_ρ and t_π not containing e_σ (see Fig. 15a), and let $\gamma_1, \ldots, \gamma_h$, $h \geq 2$, be the children of π corresponding to the edges of p_t. Nodes $\gamma_1, \ldots, \gamma_h$ are replaced with a new S-node ν'' whose children are $\gamma_1, \ldots, \gamma_h$. Path p_t is replaced in *skeleton*(π) with the nontrivial virtual edge $e_{\nu''}$; *skeleton*(ν'') consists of p_t plus a nontrivial virtual edge $e_\pi = (t_\rho, t_\pi)$ (see Fig. 15b).

We perform operation *SplitFace*($B_S(\pi)$, t_ρ, t_π, *nontrivial*) obtaining $B_S(\nu'')$ and the new $B_S(\pi)$.

To complete the transformation we first must convert the new π into an R-node. After that, node ρ is absorbed into node π by replacing edge e_ρ in *skeleton*(π) with *skeleton*(ρ) $- e_\pi$ (see Fig. 15b). Note that π will be a degenerate R-node until operation *InsertEdge* is completed, since its *skeleton* is not a triconnected simple planar graph, but contains a nontrivial split pair.

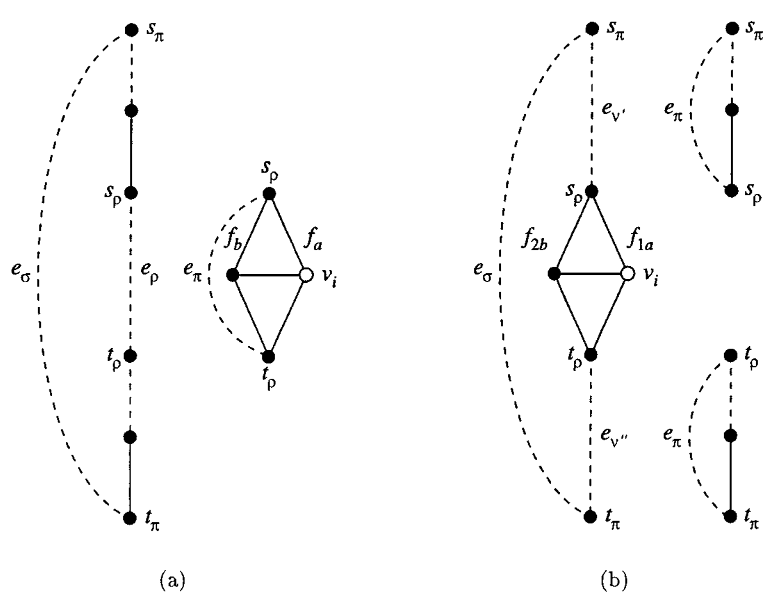


FIG. 15. An example of *RS-transformation* in *ElementaryTransformation*: (a) *skeleton*(π) and *skeleton*(ρ) before the *RS-transformation*, and (b) *skeleton*(π), *skeleton*(ν'), and *skeleton*(ν'') after the *RS-transformation*.

We discard $B_S(\pi)$, and create two new balanced binary trees $B_R(f_1)$ and $B_R(f_2)$, with at most four leaves each, for the two faces f_1 and f_2 of *skeleton*(π). In the leaves of both trees, we set:

- $\mathit{nontrivial}(e_\sigma) = 1$ and $\mathit{chain}(e_\sigma) = 0$
- $\mathit{nontrivial}(e_{\nu'}) = 1$ and $\mathit{chain}(e_{\nu'}) = \begin{cases} 1 & \text{if } \mathit{Snontrivial}(\nu') = 1 \\ 0 & \text{otherwise} \end{cases}$
- $\mathit{nontrivial}(e_\rho) = 1$ and $\mathit{chain}(e_\rho) = 0$
- $\mathit{nontrivial}(e_{\nu''}) = 1$ and $\mathit{chain}(e_{\nu''}) = \begin{cases} 1 & \text{if } \mathit{Snontrivial}(\nu'') = 1 \\ 0 & \text{otherwise.} \end{cases}$

Let f_a be the face of *skeleton*(ρ) containing e_π and v_i, and let f_b be the other face of *skeleton*(ρ) containing e_π. W.l.o.g., assume that t_ρ immediately precedes s_ρ in the circular ordering of f_a (see Fig. 15a). Let f_1 be the face of *skeleton*(π) in whose circular ordering t_ρ immediately follows s_ρ, and let f_2 be the other face of *skeleton*(π). We perform operation *MergeFaces*$(B_R(f_1), e_\rho, B_R(f_a), e_\pi)$, obtaining balanced binary tree $B_R(f_{1a})$ for the new face f_{1a}, and operation *MergeFaces*$(B_R(f_2), e_\rho, B_R(f_b), e_\pi)$, obtaining balanced binary tree $B_R(f_{2b})$ for the new face f_{2b} (see Fig. 15b).

Finally, we consider the balanced binary tree associated with node ρ. We delete the leaves of $B_R(\rho)$ corresponding to f_a and f_b; we make $B_R(\rho)$ the new $B_R(\pi)$; and we insert two new leaves corresponding to f_{1a} and f_{2b} into $B_R(\pi)$.

7.2.4. *Final Transformation2*(λ_1, λ_2)

Node λ_1 is the R-node whose skeleton contains v_1, node λ_2 is the R-node whose skeleton contains v_2. Let χ be their common parent. As described in Section 5 of [20], χ can be either an R-node, or a P-node, or an S-node; three different cases are possible for *FinalTransformation2*(χ), respectively:

1. *R-transformation.* Nodes λ_1 and λ_2 are absorbed into node χ. In *skeleton*(χ), nontrivial virtual edge e_{λ_1} is replaced with *skeleton*$(\lambda_1) - e_\chi$, nontrivial virtual edge e_{λ_2} is replaced with *skeleton*$(\lambda_2) - e_\chi$, and a trivial virtual edge (v_1, v_2) is finally added (see Fig. 16).

We first consider the balanced binary trees associated with the faces of *skeleton*(χ), *skeleton*(λ_1), and *skeleton*(λ_2). Let f_1 be the face of *skeleton*(χ) containing e_{λ_1} but not e_{λ_2}, let f_2 be the face of *skeleton*(χ) containing e_{λ_2} but not e_{λ_1}, and let f_3 be the face of *skeleton*(χ) containing both e_{λ_1} and e_{λ_2}. Let f_a be the face of *skeleton*(λ_1) containing e_χ and v_1, and f_b be the other face of *skeleton*(λ_1) containing e_χ. Let f_c be the face of *skeleton*(λ_2) containing e_χ and v_2, and f_d be the other face of *skeleton*(λ_2) containing e_χ (see Fig. 16a).

We perform operations *MergeFaces*$(B_R(f_1), e_{\lambda_1}, B_R(f_b), e_\chi)$ and *MergeFaces*$(B_R(f_3), e_{\lambda_1}, B_R(f_a), e_\chi)$, obtaining balanced binary trees $B_R(f_{1b})$ and $B_R(f_{3a})$ for the two new faces f_{1b} and f_{3a}, respectively. We also perform operations *MergeFaces*$(B_R(f_2), e_{\lambda_2}, B_R(f_d), e_\chi)$ and *MergeFaces*$(B_R(f_{3a}), e_{\lambda_2}, B_R(f_c), e_\chi)$, obtaining the balanced binary trees $B_R(f_{2d})$ and $B_R(f_{3ac})$ for the two new faces f_{2d} and f_{3ac}, respectively.

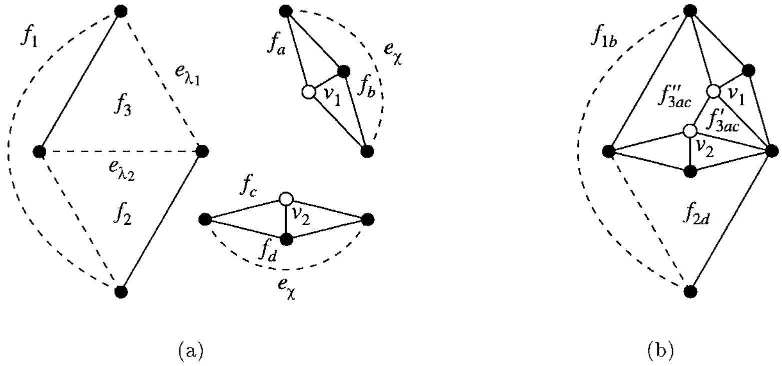


FIG. 16. An example of *R-transformation* in *FinalTransformation2*: (a) *skeleton*(χ), *skeleton*(λ_1), and *skeleton*(λ_2) before the *R-transformation*, and (b) *skeleton*(χ) after the *R-transformation*.

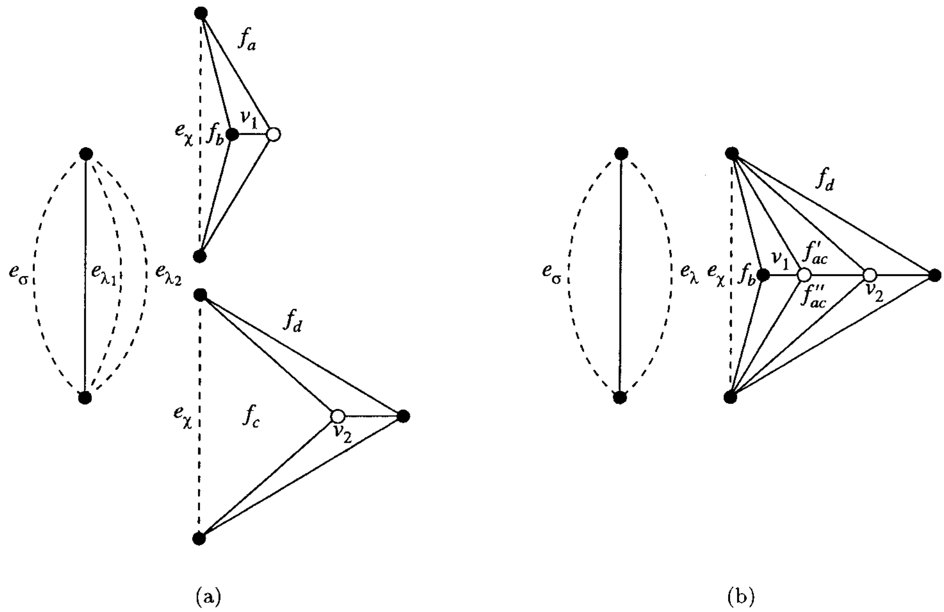


FIG. 17. An example of *P-transformation* in *FinalTransformation2*: (a) *skeleton*(χ), *skeleton*(λ_1), and *skeleton*(λ_2) before the *P-transformation*, and (b) *skeleton*(χ) and *skeleton*(λ) after the *P-transformation*.

We still must add edge (v_1, v_2), which will divide f_{3ac} into two new faces, f'_{3ac} and f''_{3ac}. We perform operation *SplitFace*($B_R(f_{3ac}), v_1, v_2,$ *trivial*), obtaining $B_R(f'_{3ac})$ and $B_R(f''_{3ac})$ (see Fig. 16b).

We now consider the balanced binary trees associated with nodes χ, λ_1, and λ_2. We delete the leaves of $B_R(\chi)$ corresponding to f_1, f_2 and f_3, the leaves of $B_R(\lambda_1)$ corresponding to f_a and f_b, and the leaves of $B_R(\lambda_2)$ corresponding to f_c and f_d. Next, we modify $B_R(\chi)$ by joining it first with $B_R(\lambda_1)$ and then with $B_R(\lambda_2)$. Finally, we insert four leaves corresponding to f_{1b}, f_{2d}, f'_{3ac}, and f''_{3ac} into $B_R(\chi)$.

2. *P-transformation*. Nodes λ_1 and λ_2 are contracted into a new R-node λ. Graph *skeleton*(λ) is obtained by the union of *skeleton*(λ_1) $- e_\chi$, *skeleton*(λ_2) $- e_\chi$, a nontrivial virtual edge e_χ between the poles, and a trivial virtual edge (v_1, v_2). In *skeleton*(χ), the nontrivial virtual edges e_{λ_1} and e_{λ_2} are replaced with a single nontrivial virtual edge e_λ (see Fig. 17). If, after the contraction, the only child of χ is λ, χ is absorbed into its parent σ, edge e_χ in *skeleton*(λ) is replaced with e_σ, and edge e_χ in *skeleton*(σ) is replaced with e_λ.

We first consider the balanced binary trees associated with the faces of *skeleton*(λ_1) and *skeleton*(λ_2). Let f_a be the face of *skeleton*(λ_1) containing e_χ and v_1, and let f_c be the face of *skeleton*(λ_2) containing e_χ and v_2 (see Fig. 17a).

We perform operation *MergeFaces*($B_R(f_a), e_\chi, B_R(f_c), e_\chi$), obtaining balanced binary tree $B_R(f_{ac})$ for the new face f_{ac}.

We still must add edge (v_1, v_2), which will divide f_{ac} into two new faces, f'_{ac} and f''_{ac}. We perform operation *SplitFace*($B_R(f_{ac}), v_1, v_2,$ *trivial*), obtaining $B_R(f'_{ac})$ and $B_R(f''_{ac})$ (see Fig. 17b).

We now consider the balanced binary trees associated with nodes λ_1, and λ_2. We delete the leaf of $B_R(\lambda_1)$ corresponding to f_a, and the leaf of $B_R(\lambda_2)$ corresponding to f_c. We then join $B_R(\lambda_1)$ and $B_R(\lambda_2)$ to obtain a new balanced binary tree $B_R(\lambda)$, and insert two leaves corresponding to f'_{ac} and f''_{ac} into $B_R(\lambda)$.

If after the contraction, the only child of χ is λ, we discard *P3nontrivial*(χ) and *P3nonchain*(χ). If the parent σ of χ is an S-node and *Snontrivial*(σ) $= 1$, let f_1 and f_2 be the two faces of *skeleton*(λ) containing e_χ, now renamed e_σ. We leave *nontrivial*(e_σ) equal to 1 and set *chain*(e_σ) equal to 1.

Otherwise, if, after the contraction, *skeleton*(χ) consists of three virtual edges, we may have to modify *P3nontrivial*(χ) and *P3nonchain*(χ). In particular, if *skeleton*(χ) contains a trivial virtual edge, we set both *P3nontrivial*(χ) and *P3nonchain*(χ) equal to 0; otherwise, if *skeleton*(χ) contains a chain virtual edge e_η (*Snontrivial*(η) $= 1$), we leave *P3nontrivial*(χ) equal to 1 and set *P3nonchain*(χ) equal to 0.

3. *S-transformation*. Nodes λ_1 and λ_2 are contracted into a new R-node λ. Let s_1 and t_1 (s_2 and t_2) be the endvertices of e_{λ_1} (e_{λ_2}) in *skeleton*(χ); w.l.o.g., assume that s_1, t_1, s_2, and t_2 appear in this order

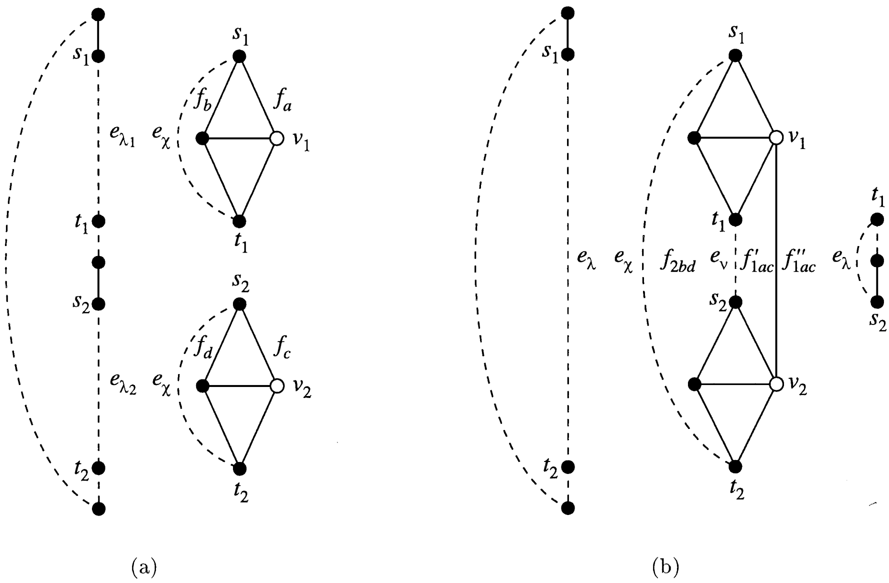


FIG. 18. An example of *S-transformation* in *FinalTransformation2*: (a) *skeleton*(χ), *skeleton*(λ_1), and *skeleton*(λ_2) before the *S-transformation*, and (b) *skeleton*(χ), *skeleton*(λ), and *skeleton*(ν) after the *S-transformation*.

between the poles of *skeleton*(χ). Let p be the path of *skeleton*(χ) between s_1 and t_2 not containing the virtual edge of the parent of χ (see Fig. 18a). Path p is replaced in *skeleton*(χ) with a nontrivial virtual edge e_λ; *skeleton*(λ) consists of p plus a nontrivial virtual edge $e_\chi = (s_1, t_2)$. Then, if t_1 and s_2 are neither coincident nor adjacent in *skeleton*(λ), the subpath p' of p between t_1 and s_2 is replaced with a nontrivial virtual edge e_ν, and a new S-node ν is created; *skeleton*(ν) consists of p' plus a nontrivial virtual edge $e_\lambda = (t_1, s_2)$. Finally, the nontrivial virtual edge e_{λ_1} in *skeleton*(λ) is replaced with *skeleton*(λ_1) - e_χ, the nontrivial virtual edge e_{λ_2} in *skeleton*(λ) is replaced with *skeleton*(e_{λ_2}) $-$ e_χ, and a trivial virtual edge (v_1, v_2) is added (see Fig. 18b).

We first consider the balanced binary trees associated with a face of *skeleton*(χ), and with the faces of *skeleton*(λ_1) and *skeleton*(λ_2).

We perform operation *SplitFace*($B_S(\chi)$, s_1, t_2, *nontrivial*), obtaining $B_S(\lambda)$ and the new $B_S(\chi)$. Then, if t_1 and s_2 are neither coincident nor adjacent in *skeleton*(λ), we perform operation *SplitFace*($B_S(\lambda)$, t_1, s_2, *nontrivial*), obtaining $B_S(\nu)$ and the new $B_S(\lambda)$.

We now must convert the new λ into an R-node. Note that λ will be a degenerate R-node until operation *InsertEdge* is completed, since its skeleton is not a triconnected simple planar graph, but a cycle of at most four virtual edges. We discard $B_S(\lambda)$, and create two new balanced binary trees $B_R(f_1)$ and $B_R(f_2)$, with at most four leaves each, for the two faces f_1 and f_2 of *skeleton*(λ). In the leaves of both trees, we set:

- *nontrivial*$(e_\chi) = 1$ and *chain*$(e_\chi) = \begin{cases} 1 & \text{if } \textit{Snontrivial}(\chi) = 1 \\ 0 & \text{otherwise} \end{cases}$
- *nontrivial*$(e_{\lambda_1}) = 1$ and *chain*$(e_{\lambda_1}) = 0$
- *nontrivial*$(e_\nu) = 1$ and *chain*$(e_\nu) = \begin{cases} 1 & \text{if } \textit{Snontrivial}(\nu) = 1 \\ 0 & \text{otherwise} \end{cases}$
- *nontrivial*$(e_{\lambda_2}) = 1$ and *chain*$(e_{\lambda_2}) = 0$.

Let f_a be the face of *skeleton*(λ_1) containing e_χ and v_1, and f_b be the other face of *skeleton*(λ_1) containing e_χ. W.l.o.g., assume that t_1 immediately precedes s_1 in the circular ordering of f_a (see Fig. 18a). Let f_1 be the face of *skeleton*(λ) in whose circular ordering t_1 immediately follows s_1, and let f_2 be the other face of *skeleton*(λ). We perform operation *MergeFaces*($B_R(f_1)$, e_{λ_1}, $B_R(f_a)$, e_χ), obtaining balanced binary tree $B_R(f_{1a})$ for the new face f_{1a}, and operation *MergeFaces*($B_R(f_2)$, e_{λ_1}, $B_R(f_b)$, e_χ), obtaining balanced binary tree $B_R(f_{2b})$ for the new face f_{2b}.

Analogously, let f_c be the face of *skeleton*(λ_2) containing e_χ and v_2, and f_d be the other face of *skeleton*(λ_2) containing e_χ (see Fig. 18a). We perform operation *MergeFaces*($B_R(f_{1a})$, e_{λ_2}, $B_R(f_c)$, e_χ), obtaining balanced binary tree $B_R(f_{1ac})$ for the new face f_{1ac}, and operation *MergeFaces*($B_R(f_{2b})$, e_{λ_2}, $B_R(f_d)$, e_χ), obtaining balanced binary tree $B_R(f_{2bd})$ for the new face f_{2bd}.

We still must add edge (v_1, v_2), which will divide f_{1ac} into two new faces, f'_{1ac} and f''_{1ac}. We perform operation *SplitFace*($B_R(f_{1ac})$, v_1, v_2, *trivial*), obtaining $B_R(f'_{1ac})$ and $B_R(f''_{1ac})$ (see Fig. 18b).

Finally, we consider the balanced binary trees associated with nodes λ_1 and λ_2. We delete the leaves of $B_R(\lambda_1)$ corresponding to f_a and f_b, and the leaves of $B_R(\lambda_2)$ corresponding to f_c and f_d. We then join $B_R(\lambda_1)$ and $B_R(\lambda_2)$ to obtain a new balanced binary tree $B_R(\lambda)$, and insert three new leaves corresponding to f'_{1ac}, f''_{1ac}, and f_{2bd} into $B_R(\lambda)$.

7.2.5. *FinalTransformation3*(λ_2)

Node λ_2 is the R-node whose skeleton contains v_2. Let χ be its parent. As described in Section 5 of [20], χ can be either an R-node or an S-node. *FinalTransformation3*(λ_2) can be viewed as a particular case of *FinalTransformation2*(λ_1, λ_2), with *skeleton*(λ_1) collapsed to a single vertex v_1 of *skeleton*(χ). The updates of the additional data structures are simple variations of those described for *R-transformation* and *S-transformation* in Section 7.2.4.

7.2.6. *Summary of Operation InsertEdge*

The above discussion on the various transformations in operation *InsertEdge* can be summarized in the following lemma.

LEMMA 10. *The transformations in operation InsertEdge require*:

- *the creation of* $O(1)$ *balanced binary trees*, *each with an* $O(1)$ *number of leaves*;
- *the execution of* $O(1)$ *join*, *insert*, *and delete operations on a balanced binary tree*;
- *the update of* $O(1)$ *values stored either at a leaf of a balanced binary tree or in a variable*; *and*
- *the execution* $O(1)$ *SplitFace and MergeFaces operations.*

7.3. *SplitFace* and *MergeFace*

In the previous section we have described the *X-transformations* and *RX-transformations* of operation *InsertEdge* in terms of the auxiliary operations *SplitFace* and *MergeFaces*. We have seen how operation *SplitFace* is performed when a face of a skeleton is split into two new faces by inserting a new virtual edge, and we have seen how operation *MergeFaces* is performed when two faces (of two different skeletons) having a virtual edge with the same endvertices are merged into a new face. In this section we show how these auxiliary operations are implemented.

We first consider operation *SplitFace*(B, u, v, *edge-type*), where B is the balanced binary tree associated with a face f of the skeleton of an R-node or S-node μ, u and v are two vertices of f, and *edge-type* $\in$ {*trivial, nontrivial*} is the type of the virtual edge $e = (u, v)$ to be inserted into the two new faces f' and f'' created by this operation. Note that if μ is an R-node, then f' and f'' belong to *skeleton*(μ); if μ is an S-node, then μ is split into two new S-nodes μ' and μ'', with f' belonging to *skeleton*(μ') and f'' belonging to *skeleton*(μ'').

Let *prev*(w) and *next*(w) be the edges preceding and following, respectively, vertex w in f. We describe the most general case, where neither the leaf corresponding to *prev*(u) nor the leaf corresponding to *prev*(v) is the rightmost leaf of B. The cases in which either the leaf corresponding to *prev*(u) or the leaf corresponding to *prev*(v) is the rightmost leaf of B are similar.

We first split B at the lowest common ancestor of the leaves corresponding to *prev*(u) and *next*(u), thus obtaining two balanced binary trees B_p and B_n, neither of which is empty. W.l.o.g., assume that the leaf corresponding to *prev*(v) is contained in B_p. We split B_p at the lowest common ancestor of the leaves corresponding to *prev*(v) and *next*(v), thus obtaining two balanced binary trees B_{pp} and B_{pn}, neither of which is empty. We join B_n and B_{pp} (in this left-to-right order) to obtain the new balanced binary tree B' for f', while B_{pn} is the new balanced binary tree B'' for f''. Finally, we insert a new leaf corresponding to e into B' and B''. If *edge-type* $=$ *trivial*, we set *nontrivial*(e) equal to 0; otherwise, we set *nontrivial*(e) equal to 1. In both cases, if f is a face of the skeleton of an R-node, we set *chain*(e) equal to 0.

We now consider operation *MergeFaces*(B', e_ρ, B'', e_π), where ρ and π are two R-nodes, B' is the balanced binary tree associated with a face f' of *skeleton*(π), B'' is the balanced binary tree associated with a face f'' of *skeleton*(ρ), e_ρ is the nontrivial virtual edge of ρ in *skeleton*(π), e_π is the nontrivial virtual edge of π in *skeleton*(ρ), and e_ρ and e_π have the same endvertices. Note that ρ and π are merged into a new node λ, with the new face f created by this operation belonging to *skeleton*(λ).

We first split B' at the leaf corresponding to e_ρ, thus obtaining two balanced binary trees (one of which is possibly empty): B'_{l} containing the leaves to the left of the leaf corresponding to e_ρ, and B'_{r} containing the leaves to the right. Similarly, we split B'' at the leaf corresponding to e_π, thus obtaining balanced binary trees B''_{l} and B''_{r} (one of which is possibly empty). We then join B'_{l}, B''_{r}, B''_{l}, and B'_{r} (in this left-to-right order) to obtain the balanced binary tree B for f.

The above discussion on operations *SplitFace* and *MergeFaces* can be summarized in the following lemma.

LEMMA 11. *Operation SplitFace requires the execution of $O(1)$ split, join, and insert operations on balanced binary trees. Operation MergeFaces requires the execution of $O(1)$ split and join operations on balanced binary trees.*

8. COMPLEXITY ANALYSIS

In this section, we analyze the space complexity of the data structure and the time complexity of the query and update operations. Throughout the section we indicate with G a biconnected planar graph that is updated on-line by adding vertices and edges, and with n the current number of vertices of G. In order to make the paper more self-contained, we quote one of the main theorems of [20], which we will refer to in our analysis.

THEOREM 4 [20]. *Let G be a biconnected planar graph that is dynamically updated by adding vertices and edges, and let n be the current number of vertices of G. There exists a data structure for the on-line incremental planarity testing problem on G with the following performance: the space requirement is $O(n)$, operations Test and InsertVertex take $O(\log n)$ worst-case time, and operation InsertEdge takes $O(\log n)$ amortized time.*

Our data structure requires $O(n)$ space. This follows from Theorem 4 and from the easily checkable $O(n)$ space complexity of the additional data structures.

Operations *StrictlyConvex* and *Convex* take $O(1)$ worst-case time (see Section 6). Since operation *Test* does not use any of the additional data structures, by Theorem 4 it takes $O(\log n)$ worst-case time.

The time complexity of the update operations follows from Theorem 4, once we prove that the additional data structures can be maintained within the specified time bounds. This immediately follows from Lemmas 9, 10, and 11, and from the following observations:

- Splitting a balanced binary tree, joining two balanced binary trees, and inserting or deleting a leaf of a balanced binary tree takes $O(\log n)$ worst-case time, and the resulting binary trees are themselves balanced (see, e.g., Chapter 4 of [50]).
- As a consequence of each split, join, insert, and delete operation, or update of the values stored at a leaf of a balanced binary tree, the values stored at the nodes of one or two leaf-to-root (sub)paths must be updated, and this also takes $O(\log n)$ worst-case time.
- Maintaining variables *P3nontrivial* and *P3nonchain*, and updating variables *sumP3-nontrivial*, *sumP3nonchain*, *sumtotalRnontrivial*, *sumtotalRnonchain*, *summaxRnontrivial*, and *sum-maxRnonchain* takes $O(1)$ time.

The entire discussion on the on-line incremental convex planarity testing problem on biconnected planar graphs can be summarized in the following theorem.

THEOREM 5. *Let G be a biconnected planar graph that is updated on-line by adding vertices and edges, and let n be the current number of vertices of G. There exists a data structure for the online* *incremental convex planarity testing problem on G with the following performance: the space requirement is $O(n)$, operations StrictlyConvex and Convex take $O(1)$ worst-case time, operations Test and InsertVertex take $O(\log n)$ worst-case time, and operation InsertEdge takes $O(\log n)$ amortized time.*

Two slightly more complicated data structures can be devised for the on-line incremental convex planarity testing problem on nonbiconnected planar graphs, similarly to what is done in [20] for the on-line incremental planarity testing problem. For connected planar graphs, we augment the above repertory with the following update operation:

AttachVertex(v, e, u): Add vertex v and connect it to vertex u by means of edge e.

As shown in [20], an n-vertex connected planar graph can be assembled starting from a single vertex by means of a sequence of O(n) *AttachVertex* and *InsertEdge* operations, such that each intermediate graph is planar and connected. For general planar graphs, we augment the above repertory with the following update operation:

MakeVertex(v): Add an isolated vertex v.

We recall that an n-vertex planar graph can be assembled starting from a single vertex by means of a sequence of $O(n)$ *MakeVertex* and *InsertEdge* operations, such that each intermediate graph is planar.

With techniques similar to those used to prove Theorem 5, it is possible to prove that there exist two data structures for the on-line incremental convex planarity testing problem on connected and on general planar graphs with the same performance as in Theorem 5, and the following performance for the additional operations: operation *AttachVertex* takes $O(\log n)$ worst-case time, and operation *MakeVertex* takes $O(1)$ worst-case time.

9. OPEN PROBLEMS

Open problems related to this work include:

- Reducing the amortized time complexity of operations *Test*, *InsertVertex*, *InsertEdge*, and *Attach Vertex* to $O(\alpha(k, n))$, where $\alpha(k, n)$ is the inverse of Ackermann's function, n is the final number of vertices of the graph, and $k \geq n$ is the total number of query and update operations. The inverse of Ackermann's function grows very slowly; namely

$$\alpha(k, n) \leq 4 \quad \text{for} \quad n < 2^{2^{\cdot^{\cdot^{2}}}}\Big\}17,$$

that is, for all values of n up to a number much greater than the estimated number of atoms in the observable universe (see, e.g., [10]). La Poutré [35] has shown that on-line incremental planarity can be tested within this time bound.

- Devising a data structure for the on-line fully dynamic convex planarity testing problem. The best data structure for the on-line fully dynamic planarity testing problem supports query and update operations in $O(\sqrt{n})$ amortized time [26].

- Characterizing the area required by a strictly convex grid drawing. Kant [33] has shown that convex grid drawings of triconnected planar graphs can be constructed with quadratic area (see also [21, 41]). Lin and Skiena [36] have shown that drawing a cycle as a strictly convex polygon with integer vertex coordinates requires $\Omega(n^3)$ area. Chrobak *et al*. [7] have presented an algorithm for constructing strictly convex grid drawings of triconnected planar graphs with $O(n^3) \times O(n^3)$ area.

- Devising a data structure for efficiently maintaining straight-line drawings of planar graphs, in particular (strictly) convex drawings, in a semi-dynamic or fully dynamic environment. This is a long-standing open problem in graph drawing. Its difficulty arises from the fact that even a single update to the graph may cause a major restructuring of the drawing. One can consider, as an example, the insertion of an edge between two antipodal vertices in a convex drawing; it is easy to see that drawing the new edge as a straight-line segment and, if possible, making the two new faces convex may require changing the coordinates of a large number of vertices. In addition, other aspects play an important role in dynamic graph drawing. For instance, it is important that the new drawing be as similar as possible to the one before the update, in order to preserve the *mental map* the viewer has of the drawing [25, 37], even though this is at the expense of some other aesthetic criteria.

ACKNOWLEDGMENTS

We thank the anonymous referees for their helpful comments and suggestions on how to improve the presentation.

REFERENCES

1. Aho, A. V., Hopcroft, J. E., and Ullman, J. D. (1974), "The Design and Analysis of Computer Algorithms," Addison-Wesley, Reading, MA.
2. Alberts, D., Gutwenger, C., Mutzel, P., and Näher, S. (1997), AGD-Library: A library of algorithms for graph drawing, *in* "*Proc. Workshop on Algorithm Engineering*," (G. F. Italiano and S. Orlando, Eds.), pp. 112–123. [Available at http://www.dsi.unive.it/~wae97/proceedings/ONLY_PAPERS/pap12.ps.gz.]
3. Brandenburg, F. J. (Ed.) (1996), "Graph Drawing (Proc. GD '95)," Lecture Notes Comput. Sci., Vol. 1027, Springer-Verlag, Berlin.
4. Bridgeman, S., Garg, A., and Tamassia, R. (1999), A graph drawing and translation service on the World Wide Web, *Internat. J. Comput. Geom. Appl*. **9**, 419–446.
5. Chiba, N., Onoguchi, K., and Nishizeki, T. (1985), Drawing planar graphs nicely, *Acta Inform*. **22**, 187–201.
6. Chiba, N., Yamanouchi, T., and Nishizeki, T. (1984), Linear algorithms for convex drawings of planar graphs, *in* "Progress in Graph Theory" (J. A. Bondy and U. S. R. Murty, Eds.), pp. 153–173, Academic Press, New York.
7. Chrobak, M., Goodrich, M. T., and Tamassia, R. (1996), Convex drawings of graphs in two and three dimensions, *in* "Proc. 12th Annu. ACM Sympos. Comput. Geom.," pp. 319–328.
8. Chrobak, M., and Kant, G. (1997), Convex grid drawings of 3-connected planar graphs, *Internat. J. Comput. Geom. Appl*. **7**, 211–223.
9. Cohen, R. F., Di Battista, G., Tamassia, R., and Tollis, I. G. (1995), Dynamic graph drawings: Trees, series-parallel digraphs, and planar *st*-digraphs, *SIAM J. Comput*. **24**, 970–1001.
10. Cormen, T. H., Leiserson, C. E., and Rivest, R. L. (1990), "Introduction to Algorithms," MIT Press, Cambridge, MA.
11. Cruz, I. F., and Eades, P. (Eds.) (1995), Special issue on graph visualization. *J. Visual Lang. Comput*. **6**.
12. de Fraysseix, H., Pach, J., and Pollack, R. (1990), How to draw a planar graph on a grid, *Combinatorica* **10**, 41–51.
13. Di Battista, G. (Ed.) (1997), "Graph Drawing (Proc. GD '97)," *Lecture Notes Comput. Sci*., Vol. 1353, Springer-Verlag, Berlin.
14. Di Battista, G., Didimo, W., Leonforte, A., Patrignani, M., and Pizzonia, M. GDToolkit. [Available at http://www.dia.uniroma3.it/~gdt/.]
15. Di Battista, G., Eades, P., de Fraysseix, H., Rosenstiehl, P., and Tamassia, R. (Eds.) (1993), "Graph Drawing '93 (Proc. ALCOM Internat. Workshop on Graph Drawing)." [Available at http://www.cs.brown.edu/people/rt/gd-93.html.]
16. Di Battista, G., Eades, P., Tamassia, R., and Tollis, I. G. (1999), "Graph Drawing," Prentice-Hall, Upper Saddle River, NJ.
17. Di Battista, G., Liotta, G., and Vargiu, F. (1995), Diagram server, *J. Visual Lang. Comput*. **6**, 275–298.
18. Di Battista, G., and Mutzel, P. (Eds.) (1999), New trends in graph drawing: Special issue on selected papers from the 1997 Symposium on Graph Drawing, *J. Graph Algorithms Appl*. **3**. [Available at http://www.cs.brown.edu/publications/jgaa/papers.html.]
19. Di Battista, G., and Tamassia, R. (1996), On-line maintenance of triconnected components with SPQR-trees, *Algorithmica* **15**, 302–318.
20. Di Battista, G., and Tamassia, R. (1996), On-line planarity testing, *SIAM J. Comput*. **25**, 956–997.
21. Di Battista, G., Tamassia, R., and Vismara, L. (1999), Output-sensitive reporting of disjoint paths. *Algorithmica* **23**, 302–340.
22. Di Battista, G., and Tamassia, R. (Eds.) (1996), Special issue on graph drawing, *Algorithmica* **16**.
23. Di Battista, G., and Tamassia, R. (Eds.) (1998), Special issue on geometric representations of graphs, *Comput. Geom. Theory Appl*. **9**.
24. Djidjev, H. N. (1995), On drawing a graph convexly in the plane, *in* "Graph Drawing (Proc. GD '94)" (R. Tamassia and I. G. Tollis, Eds.) Vol. 894, *Lecture Notes Comput. Sci*., pp. 76–83, Springer-Verlag, Berlin.
25. Eades, P., Lai, W., Misue, K., and Sugiyama, K. (1991), Preserving the mental map of a diagram, *in* "Proc. 1st Internat. Conf. on Computational Graphics and Visualization Techniques," pp. 34–43.
26. Eppstein, D., Galil, Z., Italiano, G. F., and Spencer, T. H. (1996), Separator based sparsification. I. Planarity testing and minimum spanning trees, *J. Comput. Syst. Sci*. **52**, 3–27.
27. Eppstein, D., Italiano, G. F., Tamassia, R., Tarjan, R. E., Westbrook, J., and Yung, M. (1992), Maintenance of a minimum spanning forest in a dynamic plane graph, *J. Algorithms* **13**, 33–54.
28. Eppstein, D., Italiano, G. F., Tamassia, R., Tarjan, R. E., Westbrook, J., and Yung, M. (1993), Corrigendum (Maintenance of a minimum spanning forest in a dynamic plane graph), *J. Algorithms* **15**, 173.
29. Fáry, I. (1948), On straight lines representation of planar graphs, *Acta Sci. Math*. **11**, 229–233.
30. Himsolt, M. (2000), Graphlet: Design and implementation of a graph editor, *Softw.-Pract. Exp*. **30**, 1303–1324.
31. Hoproft, J., and Tarjan, R. E. (1973), Dividing a graph into triconnected components, *SIAM J. Comput.* **2**, 135–158.

32. Italiano, G. F., La Poutré, J. A., and Rauch, M. H. (1993), Fully dynamic planarity testing in planar embedded graphs, *in* “Algorithms-ESA’93” (T. Lengauer, Ed.), Lecture Notes Comput. Sci., Vol. 726, pp. 212–223, Springer-Verlag, Berlin.
33. Kant, G. (1996), Drawing planar graphs using the canonical ordering, *Algorithmica* **16**, 4–32.
34. Kratochvíl, J. (Ed.) (1999), “Graph Drawing (Proc. GD ’99),” *Lecture Notes Comput. Sci.*, Vol. 1731, Springer-Verlag, Berlin.
35. La Poutré, J. A. (1994), Alpha-algorithms for incremental planarity testing, *in* “Proc. 26th Annu. ACM Sympos. Theory Comput.,” pp. 706–715.
36. Lin, Y.-L., and Skiena, S. S. (1995), Complexity aspects of visibility graphs, *Internat. J. Comput. Geom. Appl.* **5**, 289–312.
37. Misue, K., Eades, P., Lai, W., and Sugiyama, K. (1995), Layout adjustment and the mental map, *J. Visual Lang. Comput.* **6**, 183–210.
38. Nishizeki, T., and Chiba, N. (1988), “Planar Graphs: Theory and Algorithms,” *Ann. Discrete Math.*, Vol. 32, North-Holland, Amsterdam.
39. North, S. (Ed.) (1997), “Graph Drawing (Proc. GD ’96),” *Lecture Notes Comput. Sci.*, Vol. 1190, Springer-Verlag, Berlin.
40. Schnyder, W. (1990), Embedding planar graphs on the grid, *in* “Proc. 1st Annu. ACM-SIAM Sympos. Discrete Algorithms,” pp. 138–148.
41. Schnyder, W., and Trotter, W. T. (1992), Convex embeddings of 3-connected plane graphs, *Abstracts AMS* **13**, 502.
42. Sleator, D. D., and Tarjan, R. E. (1983), A data structure for dynamic trees, *J. Comput. System Sci.* **26**, 362–381.
43. Sleator, D. D., and Tarjan, R. E. (1985), Self-adjusting binary search trees, *J. Assoc. Comput. Mach.* **32**, 652–686.
44. Stein, S. K. (1951), Convex maps, *Proc. Amer. Math. Soc.* **2**, 464–466.
45. Steinitz, E., and Rademacher, H. (1934), *Vorlesungen über die Theorie der Polyeder*, Julius Springer, Berlin.
46. Tamassia., R., Graph drawing. [Available at `http://www.cs.brown.edu/people/rt/gd.html.]`
47. Tamassia, R. (1996), On-line planar graph embedding, *J. Algorithms* **21**, 201–239.
48. Tamassia, R. (1997), Graph drawing, *in* “Handbook of Discrete and Computational Geometry,” (J. E. Goodman and J. O’Rourke, Eds.), Chap. 44, pp. 815–832. CRC Press, Boca Raton, FL.
49. Tamassia, R., and Tollis, I. G. (Eds.) (1995), “Graph Drawing (Proc. GD ’94),” *Lecture Notes Comput. Sci.*, Vol. 894, Springer-Verlag, Berlin.
50. Tarjan, R. E. (1983), “Data Structures and Network Algorithms,” *CBMS-NSF Regional Conference Series in Applied Mathematics*, Vol. 44, Society for Industrial and Applied Mathematics, Philadelphia.
51. Tarjan, R. E. (1985), Amortized computational complexity, *SIAM J. Algebraic Discrete Methods* **6**, 306–318.
52. Thomassen, C. (1980), Planarity and duality of finite and infinite planar graphs, *J. Combin. Theory Ser. B* **29**, 244–271.
53. Thomassen, C. (1984), Plane representations of graphs, *in* “Progress in Graph Theory” (J. A. Bondy and U. S. R. Murty, Eds.), pp. 43–69, Academic Press, New York.
54. Tutte, W. T. (1960), Convex representations of graphs, *Proc. London Math. Soc.* **10**, 304–320.
55. Tutte, W. T. (1963), How to draw a graph, *Proc. London Math. Soc.* **13**, 743–768.
56. Wagner, K. (1936), Bemerkungen zum vierfarbenproblem, *Jahresbericht der Deutschen Mathematiker-Vereinigung* **46**, 26–32.
57. Westbrook, J. (1992), Fast incremental planarity testing, *in* “Automata, Languages and Programming (Proc. ICALP’92)” (W. Kuich, Ed.), *Lecture Notes Comput. Sci.*, Vol. 623, pp. 342–353, Springer-Verlag, Berlin.
58. Whitesides, S. H. (Ed.) (1998), “Graph Drawing (Proc. GD ’98),” *Lecture Notes Comput. Sci.*, Vol. 1547, Springer-Verlag, Berlin.