Output-Sensitive Reporting of Disjoint Paths

Giuseppe Di Battista, Roberto Tamassia, and
Luca Vismara

Department of Computer Science
Brown University
Providence, Rhode Island 02912

CS-96-25
September 1996

Output-Sensitive Reporting of Disjoint Paths*

Giuseppe Di Battista
Dipartimento di Discipline Scientifiche
Sezione Informatica
Università degli Studi di Roma Tre
Via della Vasca Navale 84, 00146 Roma, Italy
`dibattista@iasi.rm.cnr.it`

Roberto Tamassia
Center for Geometric Computing
Department of Computer Science
Brown University
Providence, Rhode Island 02912–1910
`rt@cs.brown.edu`

Luca Vismara
Center for Geometric Computing
Department of Computer Science
Brown University
Providence, Rhode Island 02912–1910
`lv@cs.brown.edu`

Abstract

A k-path query on a graph consists of computing k vertex-disjoint paths between two given vertices of the graph, whenever they exist. In this paper, we study the problem of performing k-path queries, with $k \leq 3$, in a graph G with n vertices. We denote with ℓ the total length of the paths reported. For $k \leq 3$, we present an optimal data structure for G that uses $O(n)$ space and executes k-path queries in output-sensitive $O(\ell)$ time. For triconnected planar graphs, our results make use of a new combinatorial structure that plays the same role as bipolar (st) orientations for biconnected planar graphs. This combinatorial structure also yields an alternative construction of convex grid drawings of triconnected planar graphs.

*Research supported in part by the National Science Foundation under grant CCR–9423847, by the NATO Scientific Affairs Division under collaborative research grant 911016, by the Progetto Coordinato Ambienti di Supporto alla Progettazione di Sistemi Informativi of the Consiglio Nazionale delle Ricerche, by the Progetto Bilaterale 94.23.CT07 Italia-USA of the Consiglio Nazionale delle Ricerche, and by the Project Alcom-IT of the Esprit Program.

1 Introduction

Connectivity is a fundamental property of graphs, and has been extensively studied in the graph algorithms literature. In particular, biconnectivity and triconnectivity properties play a special role in a number of graph algorithms.

In this paper, we investigate data structures that support the following fundamental *k-path query*, with $k \leq 3$, on a graph: given vertices u and v, compute k vertex-disjoint paths between u and v, whenever they exist. A variation of the above query, called a *k-connectivity query*, determines whether such paths exist (i.e., provides a yes/no answer) but does not return the paths. We denote with n and m the number of vertices and edges of the graph, respectively, and with ℓ the total length (number of edges) of the paths returned by a k-path query.

We are interested in constructing a space-efficient data structure for the graph such that the time for a k-path query is output-sensitive, i.e., $O(f(n) + \ell)$ with $f(n) = o(n)$. Ideally, we would like to achieve $f(n) = O(1)$ with linear space.

1.1 Previous Results on Path and Connectivity Queries

In this section, we overview previous results on k-path and k-connectivity queries. First, we consider algorithms that do not exploit preprocessing. Using network flow techniques [17], a k-path query can be answered in $O(m\sqrt{n})$ time for arbitrary k, and in $O(n+m)$ time for any fixed k. Regarding planar graphs, it has been recently shown that a k-path query can be performed in $O(n)$ time for any k [49].

Faster query time can be achieved if preprocessing is allowed. For $k = 1$, it is easy to see that a spanning forest allows one to perform 1-connectivity queries in $O(1)$ time and 1-path queries in $O(\ell)$ time. For general graphs and $k \leq 4$, or for $(k-1)$-connected graphs and fixed $k > 4$, there are $O(n)$-space data structures that perform k-connectivity queries in $O(1)$ time, but do not support output-sensitive k-path queries (see [42, 51] for $k = 2$, [14] for $k = 3$, [28] for $k = 4$, and [9] for $k > 4$).

Table 1 in Appendix A summarizes previous and new results on methods for k-path and k-connectivity queries.

1.2 Previous Results on Orientations and Orderings of Graphs

Orientations and orderings are powerful combinatorial structures that have been successfully applied to solving various graph problems. Here, we overview previous work related to our combinatorial results.

Bipolar orientations and st-numberings of biconnected graphs were first defined in conjunction with a planarity testing algorithm [18, 33], and were later used for a variety of topological and geometric graph problems, such as embedding (see, e.g., [6, 15, 42]), visibility (see, e.g., [36, 43, 53]), drawing (see, e.g., [1, 12, 44]), point location (see, e.g., [35, 45]), and floorplanning (see, e.g., [30]). One of the notable properties of planar bipolar orientations is that they induce a 2-dimensional lattice [31] on the vertices of the graph. See [10] for a recent comprehensive study of bipolar orientations.

Canonical orderings were first defined by de Fraysseix, Pach and Pollack [11] for maximal planar graphs and later extended by Kant [29] to triconnected planar graphs. They have been successfully applied to the construction of various types of planar drawings (straight-line, orthogonal, and polyline) (see, e.g., [8, 11, 29]).

Schnyder [37] defines realizers of maximal planar graphs in his study of the order dimension of planar graphs, and shows their application to planar straight-line drawings [38]. The construction of realizers of maximal planar graphs can also be efficiently parallelized [20]. Brightwell and Trotter [3, 4] define normal families of paths for a class of planar graphs that includes triconnected planar graphs. Normal families of paths are related to Schnyder's realizers. However, they do not analyze the time complexity of their construction. Normal families of paths are important for the study of the order dimension of convex polytopes and planar maps.

Graph drawing methods based on orientations, numberings and realizers are surveyed in [22].

1.3 Previous Results on Independent Spanning Trees

In recent years the problem of finding independent spanning trees of a given graph has received increasing attention. Two spanning trees of a graph G having the same root r are said *independent* if for each vertex v of G the two paths between v and r along the two trees are vertex-disjoint. Independent spanning trees find applications in fault-tolerant protocols for distributed computing networks.

An interesting conjecture about independent spanning trees is the following: for each k-connected graph G and each vertex r of G, there exist k independent spanning trees of G rooted at r. The conjecture has been proved for $k = 2$ by Itai and Rodeh [27], and for $k = 3$, independently, by Cheriyan and Maheshwari [5], and Zehavi and Itai [54]. While the proof of Zehavi and Itai is existential, the proofs of Itai and Rodeh, and of Cheriyan and Maheshwari are constructive. In particular, Itai and Rodeh used bipolar orientations, while Cheriyan and Maheshwari proved that every triconnected graph has a nonseparating ear decomposition and presented an algorithm to construct such decomposition and the three spanning trees.

For general k-connected graphs with $k \geq 4$ the conjecture is still open, but recently Huck has proved it for k-connected planar graphs with $k = 4$ [24] and $k = 5$ [26] (i.e., for all planar graphs, since 6-connected graphs are nonplanar).

Similar conjectures have been formulated considering edge-connectivity instead of vertex-connectivity [27, 32] and for directed graphs [16, 25, 46, 52].

1.4 New Results

Our new results are outlined as follows:

- We define realizers of triconnected planar graphs, and show how to construct them in linear time. Our definition naturally extends the one by Schnyder [37] using a chromatic framework such that each edge of the graph has one or two colors from the set {blue, green, red}. Our realizers induce an orientation of a triconnected planar graph with properties closely related to those of bipolar orientations for biconnected planar graphs. Our $O(n)$-time construction of a realizer of triconnected planar graph G with n vertices has the following additional applications:
  - We show how to compute a normal family of paths [3, 4] for G in $O(n)$ time. Brightwell and Trotter [3, 4] previously showed the existence of such families, but did not study the time complexity of their construction.
  - We give an alternative $O(n)$-time algorithm for constructing a convex grid drawing of G with $O(n^2)$ area. (A convex grid drawing is a planar straight-line drawing with faces drawn as convex polygons and vertices placed at integer coordinates.) This extends to triconnected planar graphs Schnyder's barycentric drawing method for maximal planar graphs [37], and gives an alternative proof of Kant's result [29].

- Based on realizers, we show how to construct a linear-space data structure that supports output-sensitive 3-path queries on a triconnected planar graph. Using this result, we show how to construct in $O(n)$ time a data structure for an n-vertex planar graph G (of arbitrary connectivity) that uses $O(n)$ space and supports k-path queries, for $k \leq 3$, in $O(\ell)$ time, where ℓ is the total size of the paths reported.
- By exploiting the result of Cheriyan and Maheshwari [5], we show how to construct a linear-space data structure that supports output-sensitive 3-path queries on a triconnected graph. Using this result, we show how to construct in $O(n^2)$ time a data structure for an n-vertex graph G (of arbitrary connectivity) that uses $O(n)$ space and supports k-path queries, for $k \leq 3$, in $O(\ell)$ time, where ℓ is the total size of the paths reported.

The rest of this paper is organized as follows. In Section 2, we present preliminary results on output-sensitive 2-path queries. Realizers of triconnected planar graphs and their combinatorial properties are introduced in Section 3. The data structure and the output-sensitive algorithm for 3-path queries in triconnected planar graphs are given in Section 4. The data structure and the output-sensitive algorithm for 3-path queries in general triconnected graphs are given in Section 5. The extension to graphs of arbitrary connectivity is contained in Section 6. In Section 7 we present the algorithm for convex grid drawing of triconnected planar graphs. Conclusions are contained in Section 8.

2 Preliminaries

In this section, we define basic concepts used in the paper, present preliminary results on output-sensitive 2-path queries, and overview previous results on canonical orderings.

2.1 Basic Definitions

We assume familiarity with graph theory [2, 21]. 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 there exists no separating $(k-1)$-set; 1-connected, 2-connected, and 3-connected graphs are usually called connected, biconnected, and triconnected, respectively.

Unless otherwise specified, all the paths referred to in this paper are simple. Two paths are vertex-disjoint when they have no vertex in common except, possibly, the endpoints. Since we deal only with vertex connectivity, for brevity we will say *disjoint* instead of vertex-disjoint. Two paths *cross* when they share at least one vertex distinct from their endpoints or one edge. The set of vertices and edges shared by two crossing paths is called a *crossing*.

A *drawing* of a graph G is a mapping of each vertex of G to a distinct point of the plane and of each edge (u, v) of G to a simple Jordan curve with end-points u and v. A drawing is *planar* if no two edges intersect, except, possibly, at common end-points. A graph is planar if it has a planar drawing.

Two planar drawings of a planar graph G are *equivalent* if, for each vertex v, they have the same circular clockwise 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 delimited by cycles; such cycles are called *faces*. The *external* face is the cycle delimiting the unbounded region. Two drawings with the same embedding have the same faces.

Let G be a plane graph. A vertex or edge of G is said to be *external* if it lies on the external face, and *internal* otherwise. A path or crossing of G is said to be *external* if it consists only of external vertices and edges and is said to be *internal* if it consists only of internal vertices and edges.

2.2 Bipolar Orientations and 2-Path Queries

In this section we show how to perform output-sensitive 2-path queries on biconnected graphs.

Let G be an n-vertex graph with an edge (s,t). A *bipolar orientation* (also called *st-orientation*) [10, 33] of G with respect to an edge (s,t) is an orientation of the edges of G such that the resulting digraph D is acyclic, s is the unique source of D, and t is the unique sink of D. A biconnected graph admits a bipolar orientation with respect to any edge (s,t), which can be computed in linear time [18]. An *st-numbering* of G is a numbering $v_1, \ldots, v_n$ of the vertices of G such that $s = v_1$, $t = v_n$, and each other vertex v_i, $1 < i < n$, is adjacent to at least one vertex v_j, $j < i$, and to at least one vertex v_k, $k > i$.

Given a bipolar orientation of a biconnected graph G, we construct two spanning trees of G, T_s and T_t, rooted at s and t, respectively, as shown by Itai and Rodeh [27]. Tree T_s is obtained by selecting an incoming edge for every vertex distinct from s (for vertex t an incoming edge distinct from (s,t)). Tree T_t is similarly obtained by selecting an outgoing edge for every vertex distinct from t (for vertex s an outgoing edge distinct from (s,t)). Clearly, for every vertex v of G, the path $p_s(v)$ along T_s between v and s and the path $p_t(v)$ along T_t between v and t are disjoint. As shown in the following lemma, trees T_s and T_t contain all the information needed to answer 2-path queries in G.

Lemma 1 *For any two vertices u and v of G, the subgraph of G formed by edge (s,t) and by the four paths $p_s(u)$, $p_s(v)$, $p_t(u)$, and $p_t(v)$, contains two disjoint paths between u and v.*

Proof: W.r.t. the bipolar orientation used to construct T_s and T_t, we indicate, for each vertex w of G, the st-number of w with $stn(w)$. Let lca_s (lca_t) be the lowest common ancestor of u and v in T_s (T_t). Three cases are possible for u and v:

1. neither vertex is an ancestor of the other in the two trees; the first path between u and v is obtained by concatenating the path between u and lca_s with the path between v and lca_s; the second is obtained by concatenating the path between u and lca_t with the path between v and lca_t; the two paths between u and v are clearly disjoint: for each ancestor x of u or v in T_s and each ancestor y of u or v in T_t, $stn(x) < stn(y)$ holds;

2. one vertex is an ancestor of the other in one of the two trees; w.l.o.g., let u be an ancestor of v in T_s; the first path between u and v is the one along T_s; the second is obtained by concatenating the path between u and lca_t with the path between v and lca_t; the disjointnes of the two paths between u and v can be proved as in the previous case;

3. u is an ancestor of v in one tree and v is an ancestor of u in the other; w.l.o.g., let let u be an ancestor of v in T_s and v be an ancestor of u in T_t; the first path between u and v is the one along T_s or the one along T_t; the second is obtained by concatenating the path between u and s, with edge(s,t), with the path between v and t; the two paths between u and v are clearly disjoint: let x be an ancestor of u in T_s, y be an ancestor of v in T_t, and w be a vertex

of the path between u and v along T_s or along T_t, $stn(x) < stn(w) < stn(y)$ holds and (s,t) is neither an edge of T_s nor an edge of T_t.

Note that, by the construction of T_s and T_t, the case in which one vertex is an ancestor of the other in both trees is not possible. □

Theorem 1 *Let G be a biconnected graph with n vertices and m edges. There exists an $O(n)$-space data structure for G that can be constructed in $O(n+m)$ time and supports 2-path queries in $O(\ell)$ time, where ℓ is the size of the reported paths.*

Proof: The data structure simply stores rooted trees T_s and T_t with parent pointers. It is easy to see that this data structure can be constructed in time $O(n+m)$ and requires $O(n)$ space [18]. A 2-path query for vertices u and v is performed by traversing paths $p_s(u)$, $p_s(v)$, $p_t(u)$, and $p_t(v)$ one edge at the time, alternating between them, until the two following halting events occur:

- lca_s is reached and
- lca_t is reached.

Note that $lca_s \neq lca_t$ may be coincident with u or v. If both lowest common ancestors are different from u and v, then Case 1 of the proof of Lemma 1 applies. If exactly one of the lowest common ancestors coincides with u or v, then Case 2 of the proof of Lemma 1 applies. If one of the lowest common ancestors coincides with u and the other coincides with v, then Case 3 of the proof of Lemma 1 applies.

Once the proper case has been determined, reporting the two paths between u and v can be done in $O(\ell)$ time by simply traversing trees T_s and T_t. Thus, it remains to be proved that the computation of the two lowest common ancestors lca_s or lca_t can be carried out in $O(\ell)$ time. This is guaranteed by the alternating traversal technique and by the fact that the paths explored to compute lca_s or lca_t are reused for constructing one or two paths between u and v. □

2.3 Canonical Orderings

In this section we recall the definition of canonical orderings of triconnected plane graphs, as given by Kant [29].

Let G be a triconnected plane graph with n vertices, and u_0, u_1, u_2 be three consecutive external vertices of G. A *canonical ordering* of G (see Fig. 1) is an ordering $v_1, \ldots, v_n$ of the vertices of G that can be partitioned into subsequences $V_1, \ldots, V_h$, where $V_k = \{v_{s_k}, \ldots, v_{s_{k+1}-1}\}$, $1 \leq k \leq h$, and $1 = s_1 < s_2 < \ldots < s_h < s_{h+1} = n+1$, such that the following conditions are verified:

1. $v_1 = u_1$, $v_2 = u_2$, and $V_1 = \{v_1, v_2\}$.
2. Let G_k be the plane subgraph of G induced by $V_1 \cup \ldots \cup V_k$, $k \leq h$, and C_k be the external face of G_k. For each $2 \leq k \leq h-1$ one of the following cases occurs:
   (a) $V_k = \{v_{s_k}\}$ is a vertex of C_k and has at least one neighbor in $G - G_k$;
   (b) $V_k = \{v_{s_k}, \ldots, v_{s_k+d_k}\}$ is a subpath of C_k, and each v_i, $s_k < i < s_k + d_k$, has at least one neighbor in $G - G_k$ and no neighbor in G_{k-1}.
3. Each subgraph G_k is biconnected and internally triconnected, i.e., removing two internal vertices of G_k does not disconnect it.

4. $v_n = u_0$ and $V_h = \{v_n\}$.

In the example of Fig. 1, each vertex is labeled with its rank in the canonical ordering, and the partition of the vertices is given by $V_1 = \{v_1, v_2\}$, $V_2 = \{v_3, v_4, v_5\}$, $V_3 = \{v_6, v_7\}$, $V_4 = \{v_8\}$, $V_5 = \{v_9, v_{10}\}$, $V_6 = \{v_{11}\}$, $V_7 = \{v_{12}\}$, $V_8 = \{v_{13}\}$, $V_9 = \{v_{14}\}$, $V_{10} = \{v_{15}, v_{16}\}$, $V_{11} = \{v_{17}, v_{18}\}$, $V_{12} = \{v_{19}\}$, $V_{13} = \{v_{20}\}$, $V_{14} = \{v_{21}\}$.

Lemma 2 [29] *Each triconnected plane graph has a canonical ordering, which can be computed in linear time and space.*

3 Realizers of Triconnected Planar Graphs

3.1 Definition

A *realizer* of a triconnected plane graph G is a triplet of rooted directed spanning trees of G with the following properties (see Fig. 1.a–1.c):

1. In each spanning tree, the edges of G are directed from children to parent.

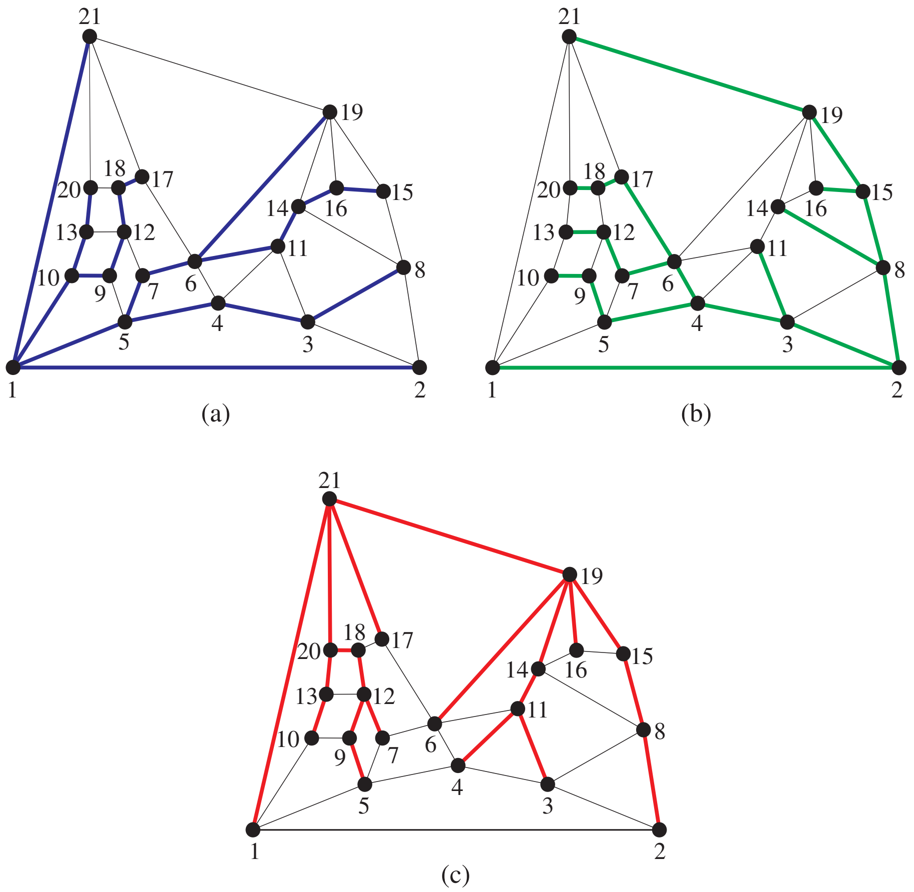

Figure 1: A realizer of a triconnected planar graph G. (a) The blue tree of G. (b) The green tree of G. (c) The red tree of G.

2. The sinks (roots) of the spanning trees are three external vertices of G.

3. Each edge of G is contained in at least one and in at most two spanning trees.

4. If an edge of G is contained in two spanning trees, then it has different directions in the two trees.

5. Consider the edges of G with the directions they have in the three spanning trees, where an edge with two opposite directions is considered twice (see Fig. 2):

   (a) Each non-sink vertex v of G, has exactly three *outgoing edges*; the circular order of the outgoing edges around v induces a circular order of the spanning trees around v; all the non-sink vertices of G have the same circular order of the spanning trees.

   (b) For each vertex of G the *incoming edges* that belong to the same spanning tree appear consecutively between the outgoing edges of the two other spanning trees (the first and last incoming edges are possibly coincident with the outgoing edges).

6. For the sink of each spanning tree, all the incoming edges belong to that spanning tree.

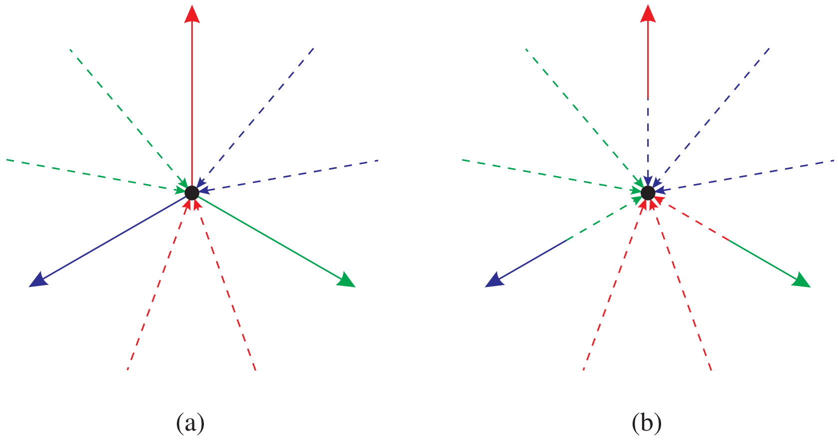

Figure 2: Two cases of Property 5 of the realizers.

Let T_b, T_g, and T_r be the spanning trees forming a realizer of a triconnected plane graph G (see Fig. 1.b–1.c). We assign a color to the edges of G contained in T_b, T_g, and T_r, say *blue*, *green*, and *red*, respectively. According to Property 3 of the realizers, each edge of G is assigned one or two colors, and is said to be *1-colored* or *2-colored*, respectively. For example, in the realizer shown in Fig. 1, edge (v_4, v_{11}) is 1-colored, while edge (v_4, v_5) is 2-colored.

Lemma 3 *Each triconnected plane graph G has a realizer, which can be computed in linear time and space.*

Proof: A realizer can be constructed by assigning colors and directions to the edges of G as follows:

1. a canonical ordering of the vertices of G is computed;

2. v_1, v_2, and v_n are the sinks of the blue, green, and red tree, respectively;

3. (v_1, v_2) is an outgoing blue edge for v_2 and an outgoing green edge for v_1;

4. for each $2 \leq k \leq h$:

   (a) if $V_k = \{v_{s_k}\}$, let $c_r, \ldots, c_l$ be the consecutive neighbors of v_{s_k} on C_{k-1}; (v_{s_k}, c_l) is an outgoing blue edge for v_{s_k}, and possibly an outgoing red edge for c_l if c_l has no neighbor in $G - G_k$; (v_{s_k}, c_r) is an outgoing green edge for v_{s_k}, and possibly an outgoing red edge for c_r if c_r has no neighbor in $G - G_k$; edges (v_{s_k}, c_i), $r < i < l$ are outgoing red edges for c_i (see Fig. 3.a; from now on we represent a 2-colored edge half with one color and half with the other; dashes represent optionality);

   (b) if $V_k = \{v_{s_k}, \ldots, v_{s_k+d_k}\}$, let c_r and c_l be the neighbors of v_{s_k} and $v_{s_k+d_k}$ on C_{k-1}, respectively; $(v_{s_k+d_k}, c_l)$ is an outgoing blue edge for $v_{s_k+d_k}$, and possibly an outgoing red edge for c_l if c_l has no neighbor in $G - G_k$; (v_{s_k}, c_r) is an outgoing green edge for v_{s_k}, and possibly an outgoing red edge for c_r if c_r has no neighbor in $G - G_k$; edge (v_i, v_{i+1}), $s_k \leq i < s_k + d_k$ is an outgoing blue edge for v_i and an outgoing green edge for v_{i+1} (see Fig. 3.b).

Note that v_1 has no outgoing blue edge, v_2 has no outgoing green edge, and v_n has no outgoing red edge. Besides, for each $2 \leq k \leq h$, the following invariants hold:

- every vertex of V_k has exactly one outgoing blue edge, exactly one outgoing green edge, and no outgoing red edge; the outgoing blue edge precedes the outgoing green edge in the clockwise circular order of the edges of C_k, and all the (possible) incoming red edges are incident with vertices of $G_k - V_k$;

- for every vertex of C_k the (possible) incoming blue edge of C_k follows the (possible) incoming green edge of C_k in the clockwise circular order of the edges of C_k;

- no vertex of C_{k-1} has an outgoing blue or green edge incident with a vertex of V_k;

- every vertex of C_{k-1} with no neighbor in $G - G_k$ has exactly one outgoing red edge, while every vertex of C_{k-1} with neighbors in $G - G_k$ has no outgoing red edge;

- G_k contains no cycle such that a common color is assigned to all its edges.

All the properties of a realizer easily follow from these invariants. By Lemma 2, the above construction can be carried out in linear time and space. □

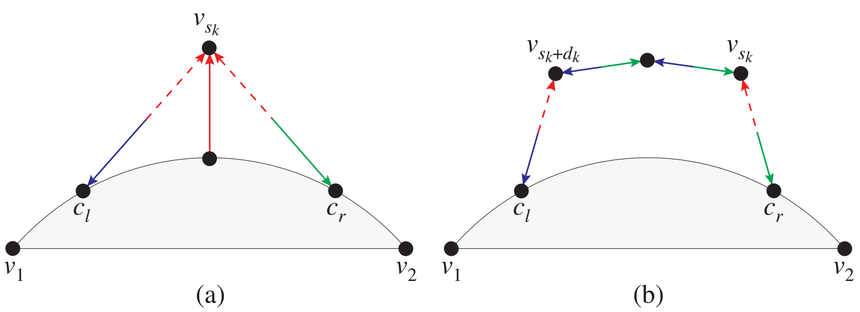

Figure 3: The coloring of the edges in the construction of a realizer. (a) $V_k = \{v_{s_k}\}$. (b) $V_k = \{v_{s_k}, \ldots, v_{s_k+d_k}\}$.

3.2 Properties

In this section, we consider a triconnected plane graph G equipped with a realizer T_b, T_g, T_r. We denote v_1, v_2, and v_n as s_b, s_g, and s_r, respectively. For each vertex v of G, the *blue path* $p_b(v)$ is the path of G along T_b with endpoints v and s_b. In the same way, we define the *green path* $p_g(v)$ as the path of G along T_g with endpoints v and s_g and the *red path* $p_r(v)$ as the path of G along T_r with endpoints v and s_r. In the rest of the paper, the subpath of path $p_i(v)$, $i \in \{b, g, r\}$, with endpoints v and the ancestor u of v in T_i is denoted by $p_i(v, u)$.

The subpath of the external face with endpoints s_g and s_r and not containing s_b is denoted by $ext(s_g, s_r)$. Similarly, the subpath of the external face with endpoints s_r and s_b and not containing s_g is denoted by $ext(s_r, s_b)$ and the subpath of the external face with endpoints s_b and s_g and not containing s_r is denoted by $ext(s_b, s_g)$.

The lowest common ancestor of vertices u and v in T_i, $i \in \{b, g, r\}$ is denoted by $lca_i(u, v)$; in the rest of the paper, we will use lca_i instead of $lca_i(u, v)$ for brevity.

From the construction in the proof of Lemma 3, it follows that, for each vertex of G, the colors of the three outgoing edges appear in the following counterclockwise circular order: blue, green, red. W.l.o.g., set $\{b, g, r\}$ will be considered accordingly ordered in the rest of the paper.

Lemma 4 *Let G be a triconnected plane graph with n vertices and m edges. For every realizer of G, the number of 2-colored edges of G is $3n - m - 3$.*

Proof: For each planar graph, $m \leq 3n - 6$. Each tree with n vertices has $n - 1$ edges; thus the total number of edges in the three spanning trees of the realizer is $3(n-1) > m$. The thesis follows from Property 3 of the realizers. □

Lemma 5 *Let v be a vertex of G and i, j, k be three consecutive colors in the set $\{b, g, r\}$. Let $x \neq s_j$ be a vertex of $p_j(v)$ and y be its parent in T_j. The i-colored (k-colored) outgoing edge of x is on the right (left) of $p_j(v)$, while each (possible) i-colored (k-colored) incoming edge of x different from (y, x) is on the left (right) of $p_j(v)$.*

Proof: Easily follows from Properties 5a and 5b of the realizers, from the circular order of set $\{b, g, r\}$, and from planarity of G. □

Lemma 6 *For each vertex v of G, $p_b(v)$, $p_g(v)$, and $p_r(v)$ have only vertex v in common.*

Proof: Let i, j and k be three consecutive colors in the set $\{b, g, r\}$. Suppose, for a contradiction, that $p_i(v)$ and $p_j(v)$ have vertex x in common and that $p_i(v, x)$ and $p_j(v, x)$ have no vertex in common with $p_k(v)$. By Property 6 of the realizers, $x \neq s_j$. From Property 5a of the realizers and by planarity of G, it follows that the edge of $p_i(v)$ incoming to x is on the right of $p_j(v)$, thus contradicting Lemma 5. □

Lemma 7 *Let u and v be two vertices of G. If there exist two colors $i, j \in \{b, g, r\}$, $i \neq j$, such that $v \in p_i(u)$ and $u \in p_j(v)$, then $p_i(u, v) = p_j(v, u)$.*

Proof: Suppose, for a contradiction, that $p_i(u, v)$ and $p_j(v, u)$ have only vertices u and v in common. Since G is planar, two cases are possible: $p_j(v, u)$ is an internal path in the subgraph with external face formed by $p_i(u)$, $p_j(u)$ and $ext(s_i, s_j)$, or $p_i(u, v)$ is an internal path in the subgraph with external face formed by $p_i(v)$, $p_j(v)$ and $ext(s_i, s_j)$. It is easy to see that in the first case Property 5b of the realizers is not satisfied for vertex u, and in the second case it is not satisfied for vertex v. Thus, $p_i(u, v)$ and $p_j(v, u)$ have a third vertex w in common besides u and v. The same argument can be recursively applied to $p_i(u, w)$ and $p_j(w, u)$, and to $p_i(w, v)$ and $p_j(v, w)$. This completes the proof. □

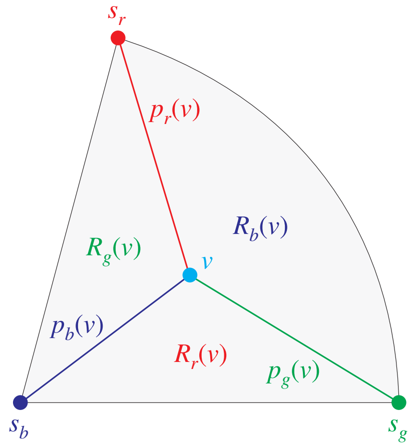

Figure 4: The blue, green, red paths and regions of a vertex.

Lemma 8 *For vertices s_b, s_g, and s_r of G the following properties hold: $p_r(s_g) = p_g(s_r) = ext(s_g, s_r)$; $p_b(s_r) = p_r(s_b) = ext(s_r, s_b)$; $p_g(s_b) = p_b(s_g) = ext(s_b, s_g)$.*

Proof: We prove that $p_r(s_g) = p_g(s_r) = ext(s_g, s_r)$; the other two cases are analogous.

Equality $p_r(s_g) = p_g(s_r)$ follows from Lemma 7, so we only have to prove that $p_r(s_g) = p_g(s_r)$ is external.

We first prove that the first edge (s_g, w_g) and the last edge (w_r, s_r) of $p_r(s_g) = p_g(s_r)$ are external. By Properties 5a and 6 of the realizers, the outgoing blue edge and the outgoing red edge of s_g are consecutive in the counterclockwise circular order of the edges around s_g. Suppose, for a contradiction, that the edge of $ext(s_g, s_r)$ incident with s_g is not the outgoing red edge of s_g. By planarity of G, $p_b(s_g)$ and $p_r(s_g)$ have at least one vertex in common, thus contradicting Lemma 6. Similarly, it can be proved that the edge of $ext(s_g, s_r)$ incident with s_r is the outgoing green edge of s_r.

We now complete the proof by showing that also the other edges of $p_r(s_g) = p_g(s_r)$ are external. Suppose, for a contradiction, that $p_r(s_g) = p_g(s_r) \neq ext(s_g, s_r)$; hence, there exists a vertex $x \neq s_g, w_g, s_r, w_r$ of $ext(s_g, s_r)$ which is not a vertex of $p_r(w_g) = p_g(w_r)$. Since the graph is planar, $p_g(x)$ ($p_r(x)$) has at least a vertex y (z) in common with $p_r(w_g) = p_g(w_r)$. It is easy to see that Property 5b of the realizers is not satisfied for vertices y and z. □

For each vertex v of G the *blue region* $R_b(v)$ is the subgraph of G with external face formed by $p_g(v)$, $p_r(v)$ and $ext(s_g, s_r)$ (see Fig. 4). In the same way, the *green region* $R_g(v)$ is the subgraph of G with external face formed by $p_b(v)$, $p_r(v)$ and $ext(s_r, s_b)$ and the *red region* $R_r(v)$ is the subgraph of G with external face formed by $p_b(v)$, $p_g(v)$ and $ext(s_b, s_g)$.

Lemma 9 *For each pair of vertices u and v of G, two cases are possible:*

1. *there are exactly two colors $i, j \in \{b, g, r\}$, $i \neq j$, such that $p_i(v)$ and $p_j(u)$ cross; three subcases are possible:*
   - *(a) $u \notin p_i(v)$ and $v \notin p_j(u)$ (see Fig. 5.a);*
   - *(b) either $u \in p_i(v)$ or $v \in p_j(u)$ (see Fig. 5.b);*
   - *(c) $u \in p_i(v)$ and $v \in p_j(u)$ (see Fig. 5.c);*

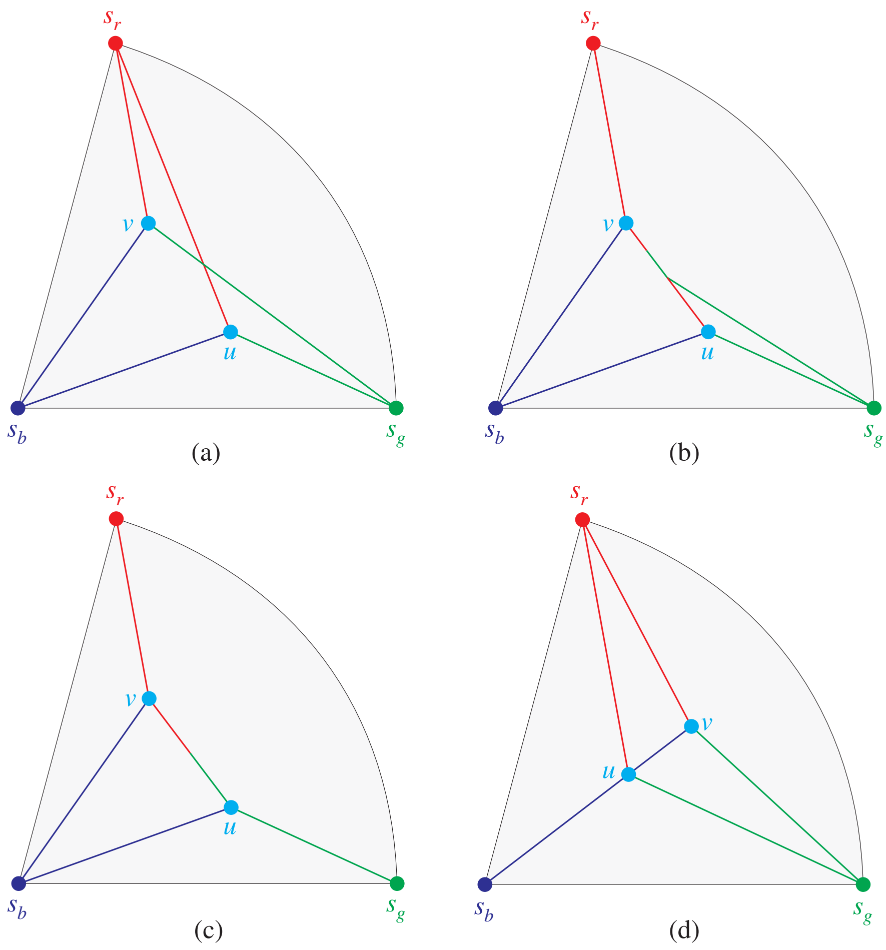

Figure 5: (a–c) Crossing paths. (d) Non-crossing paths.

2. *there are no two colors $i, j \in \{b, g, r\}$, $i \neq j$, such that $p_i(v)$ and $p_j(u)$ cross; in this case there is exactly one color $k \in \{b, g, r\}$ such that either $p_k(v) \subset p_k(u)$ or $p_k(u) \subset p_k(v)$ (see Fig. 5.d);*

Proof: Consider path $p_r(u)$ ($p_b(u)$ and of $p_g(u)$ are analogous). Also, suppose that u and v do not coincide with s_b, s_g, and s_r; otherwise the proof can be trivially extended but involves some more details. In order to simplify the exposition of the proof of this property, we define $\bar{p}_i(v) = p_i(v) - \{v\}$, $i \in \{b, g, r\}$, and $\bar{R}_i(v) = R_i(v) - \{p_j(v) \cup p_k(v)\}$, $i, j, k \in \{b, g, r\}$, $i \neq j \neq k$.

By exploiting Lemmas 5 and 7, we can prove the following properties of $p_r(u)$. Path $p_r(u)$ is composed by four consecutive subpaths $p_{r1}(u)$, $p_{r2}(u)$, $p_{r3}(u)$, and $p_{r4}(u)$, where an endvertex of $p_{r4}(u)$ is s_r. The vertices of path $p_{r1}(u)$ belong to $\bar{R}_r(v)$. For the vertices of $p_{r2}(u)$ and $p_{r3}(u)$ two cases are possible: (i) the vertices of $p_{r2}(u)$ belong to $\bar{p}_b(v)$ and the vertices of $p_{r3}(u)$ belong to $\bar{R}_g(v)$; (ii) the vertices of $p_{r2}(u)$ belong to $\bar{p}_g(v)$ and the vertices of $p_{r3}(u)$ belong to $\bar{R}_b(v)$. The vertices of $p_{r4}(u)$ belong to $\bar{p}_r(v)$.

According to the position of u w.r.t. v, some of these subpaths may be empty:

- if $u \in \bar{R}_r(v)$ then either $p_{r2}(u)$ and $p_{r3}(u)$ are both empty, or only $p_{r3}(u)$ is empty, or none of the subpaths is empty;

- if $u \in \bar{p}_b(v)$ or $u \in \bar{p}_g(v)$ then $p_{r1}(u)$ is empty; also, $p_{r3}(u)$ is possibly empty, while $p_{r2}(u)$ and $p_{r4}(u)$ are not empty;
- if $u \in \bar{R}_b(v)$ or $u \in \bar{R}_g(v)$ then $p_{r1}(u)$ and $p_{r2}(u)$ are empty; $p_{r3}(u)$ and $p_{r4}(u)$ are not empty;
- if $u \in \bar{p}_r(v)$ then $p_{r1}(u)$, $p_{r2}(u)$, and $p_{r3}(u)$ are empty; $p_{r4}(u)$ is not empty.

The above properties allow to easily prove the claims. □

Corollary 1 *Let u and v be two vertices of G. If there exist two colors $i, j \in \{b, g, r\}$, $i \neq j$, such that $p_i(v)$ and $p_j(u)$ cross, then $u \in R_j(v)$ and $v \in R_i(u)$.*

Proof: Easily follows from the proof of Lemma 9. □

Lemma 10 *Let i, j, and k be three consecutive colors in the circularly ordered set $\{b, g, r\}$. For each pair of vertices u and v of G, if $u \in R_k(v)$ the following five cases are possible:*

1. *if $u \notin p_i(v)$ and $u \notin p_j(v)$, then $R_k(u) \subset R_k(v)$;*
2. *if $u \in p_i(v)$ and $v \notin p_j(u)$, then $R_k(u) \subset R_k(v)$;*
3. *if $u \in p_j(v)$ and $v \notin p_i(u)$, then $R_k(u) \subset R_k(v)$;*
4. *if $u \in p_i(v)$ and $v \in p_j(u)$, then $R_k(u) = R_k(v)$;*
5. *if $u \in p_j(v)$ and $v \in p_i(u)$, then $R_k(u) = R_k(v)$.*

Proof: *Case 1.* By planarity of G and by Lemma 9, either $p_k(u)$ and $p_i(v)$ cross or $p_k(u)$ and $p_j(v)$ cross or $p_k(v) \subset p_k(u)$; in all three subcases, by Lemma 9, $p_i(u, lca_i)$ and $p_j(u, lca_j)$ are internal paths of $R_k(v)$, hence $R_k(u) \subset R_k(v)$.

Case 2. Since $u \in p_i(v)$, then $p_i(u) \subset p_i(v)$; by Lemma 9, $p_j(u, lca_j)$ is an internal path of $R_k(v)$, hence $R_k(u) \subset R_k(v)$.

Case 3. Analogous to Case 2.

Case 4. By Lemma 7, $p_i(v, u) = p_j(u, v)$, hence $R_k(u) = R_k(v)$.

Case 5. Analogous to Case 4. □

The properties of a *normal family of paths* [4] for a plane graph and three distinguished external vertices, are similar to the properties of Lemmas 6, 8, and 10. Brightwell and Trotter [4] proved that each triconnected plane graph has a normal family of paths for any three external vertices. Using the terminology of [4], we can say that Lemmas 6, 8, and 10 show that the set $\{p_i(v) | i \in \{b, g, r\}, v \in V\}$ is a normal family of paths for the three vertices s_b, s_g, and s_r. Also, a normal family of paths of a triconnected planar graph can be constructed, for any three external vertices x, y, and z, by adding a vertex w adjacent to x, y, and z, by constructing a single-sink realizer (which will be defined in Section 4.2) rooted at w, and then by removing w.

3.3 Faces Colored by Realizers

Let G be a triconnected plane graph equipped with a realizer T_b, T_g, T_r. Let f be an internal (external) face of G, and let e be an edge of f in T_b. We say that e is *positive blue* if the orientation of e in T_b follows f clockwise (counterclockwise); we say that e is *negative blue* if the orientation of e in T_b follows f counterclockwise (clockwise). We define *positive green*, *negative green*, *positive red*, and *negative red* in a similar way. The following lemmas characterize the chromatic structure of a face induced by the realizer.

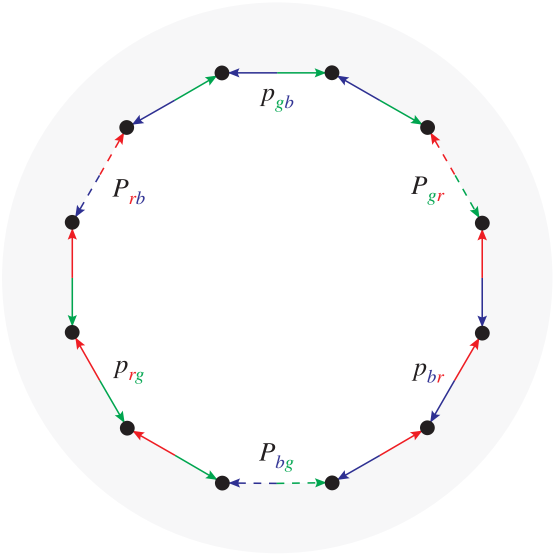

Figure 6: Internal face coloring.

Lemma 11 *An internal face of G can be decomposed into six clockwise consecutive paths P_{bg}, p_{rg}, P_{rb}, p_{gb}, P_{gr}, p_{br} where (see Fig. 6):*

- *P_{bg} consists of exactly one edge that is either positive blue, or positive blue and negative green, or negative green;*
- *p_{rg} consists of a possibly empty sequence of edges, each positive red and negative green;*
- *P_{rb} consists of exactly one edge that is either positive red, or positive red and negative blue, or negative blue;*
- *p_{gb} consists of a possibly empty sequence of edges, each positive green and negative blue;*
- *P_{gr} consists of exactly one edge that is either positive green, or positive green and negative red, or negative red;*
- *p_{br} consists of a possibly empty sequence of edges, each positive blue and negative red.*

Proof: Let f be an internal face of G, and let a clockwise circular order of the vertices around f be defined.

We consider the most general case in which f contains no vertex from the set $\{s_b, s_g, s_r\}$. The cases in which f contains one or two vertices from the set $\{s_b, s_g, s_r\}$ are particular cases of this one.

For each vertex of f, by Properties 3 and 4 of the realizers, at least one of the three outgoing edges does not belong to f.

We first prove that, for each color $i \in \{b, g, r\}$, there exists at least one vertex of f whose i-colored outgoing edge does not belong to f. Suppose the contrary; since each vertex of f has exactly one i-colored outgoing edge, these edges would form an i-colored cycle; a contradiction, since T_i is a tree.

Then we prove that, for each color $i \in \{b, g, r\}$, there exists at least one vertex v of f such that $f \subseteq R_i(v)$. In particular, we will prove the result for $i = r$; the other two cases are analogous.

Let v be a vertex of f whose red outgoing edge does not belong to f; let u (w) be the vertex of f preceding (following) v. We consider the clockwise circular order around v of its outgoing edges and of the two edges belonging to f. Three cases are possible:

1. (u,v), the outgoing green edge (possibly coincident with (u,v)), the outgoing blue edge, the outgoing red edge, and (v,w) appear in this order around v; thus, $f \not\subseteq R_r(v)$; however, by Property 5 of the realizers, (w,v) is an outgoing blue edge for w, and is followed, around w, by the outgoing red edge and by the outgoing green edge; thus, $f \subseteq R_r(w)$;

2. (u,v), the outgoing blue edge (possibly coincident with (u,v)), the outgoing red edge, the outgoing green edge (possibly coincident with (v,w)), and (v,w) appear in this order around v_i; thus, $f \subseteq R_r(v)$;

3. (u,v), the outgoing red edge, the outgoing green edge, the outgoing blue edge (possibly coincident with (v,w)), and (v,w) appear in this order around v; thus, $f \not\subseteq R_r(v)$; however, by Property 5 of the realizers, (u,v) is an outgoing green edge for u, and is preceded, around u, by the outgoing red edge and by the outgoing blue edge; thus, $f \subseteq R_r(u)$;

It is also easy to see that, if $f \subseteq R_i(v)$, then $f \not\subseteq R_j(v)$, $i,j \in \{b,g,r\}$, $i \neq j$; hence, for each color $i \in \{b,g,r\}$, the vertex of f such that $f \subseteq R_i(v)$ is distinct from the vertices of f for the other two colors.

By making use of the vertices of f whose red region contains f, we will now prove the claim for P_{rb}, p_{gb}, and P_{gr}.

We first consider the case in which there exists only one vertex v of f such that $f \subseteq R_r(v)$. Let u (w) be the vertex of f preceding (following) v. The outgoing blue edge of v either follows (u,v) in the clockwise circular order around v or coincides with (u,v); in the first case, by Property 5 of the realizers, (u,v) is the outgoing red edge of u; in the second case, still by Property 5 of the realizers, (u,v) may or may not be also the outgoing red edge of u. It follows that (u,v) is either positive red, or positive red and negative blue, or negative blue, i.e., $P_{rb} = (u,v)$. Analogously, (v,w) is either positive green, or positive green and negative red, or negative red, i.e., $P_{gr} = (v,w)$. In this case p_{gb} is empty.

We now consider the case in which there exists more than one vertex v of f such that $f \subseteq R_r(v)$. Let $v_1, \ldots, v_k$ be these vertices. By Lemma 10, it is easy to prove that all vertices v_h, $1 \leq h \leq k$ are consecutive in f, and that $R_r(v_1) = R_r(v_2) = \ldots = R_r(v_k)$. It follows that edge (v_h, v_{h+1}), $1 \leq h < k$ is an outgoing green edge for v_h and an outgoing blue edge for v_{h+1}; thus (v_h, v_{h+1}) is positive green and negative blue, i.e., $p_{gb} = (v_1, v_2), \ldots, (v_{h+1}, v_h)$. Let u (w) be the vertex of f preceding v_1 (following v_k); as in the previous case $P_{rb} = (u,v)$ and $P_{gr} = (v,w)$.

The proof of the claim for P_{gr}, p_{br}, and P_{bg} (P_{bg}, p_{rg}, and P_{rb}) is analogous and makes use of the vertices of f whose green (blue) region contains f. $\square$

Lemma 12 *The external face of G can be decomposed into three counterclockwise consecutive paths p_{gb}, p_{rg}, p_{br} where (see Fig. 7):*

- *p_{gb} consists of a sequence of edges, each positive green and negative blue;*
- *p_{rg} consists of a sequence of edges, each positive red and negative green;*
- *p_{br} consists of a sequence of edges, each positive blue and negative red.*

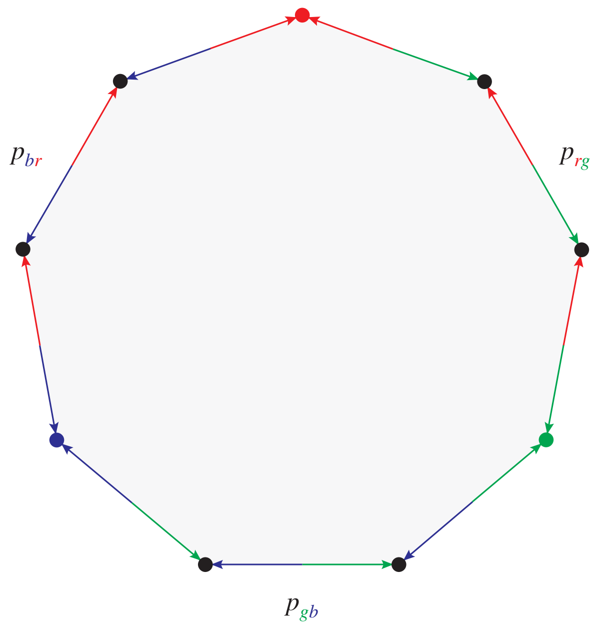

Figure 7: External face coloring.

Proof: Immediately follows from Lemma 8. □

It is well known that the dual graph of a triconnected planar graph is triconnected. We consider a triconnected planar graph G equipped with a realizer, and define the *extended dual* graph G^* of G as follows:

- each internal face of G has a corresponding vertex in G^*; the external face of G has three corresponding vertices v_b^*, v_g^*, and v_r^* in G^*;
- each edge of G has a corresponding edge in G^*;
- two vertices of G^*, different from v_b^*, v_g^*, and v_r^*, are adjacent if and only if the corresponding faces of G have an edge in common;
- v_b^* is adjacent to all the vertices of G^* corresponding to faces of G incident with an edge of p_{rg} (see Lemma 12); v_g^* is adjacent to all the vertices of G^* corresponding to faces of G incident with an edge of p_{br}; v_r^* is adjacent to all the vertices of G^* corresponding to faces of G incident with an edge of p_{gb};
- v_b^* is adjacent to v_g^*; v_g^* is adjacent to v_r^*; v_r^* is adjacent to v_b^*.

It is easy to see that also the extended dual graph of a triconnected planar graph is triconnected planar.

Lemma 13 *The realizer of a triconnected planar graph induces a realizer of its extended dual.*

Proof: Let G be a triconnected planar graph equipped with a realizer and G^* be its extended dual. Let v^* be a vertex of G^*, different from v_b^*, v_g^*, and v_r^*, e be an edge of G and e^* be its corresponding edge in G^*. We color the edges incident with v^* as follows (see Lemma 11):

- if e is the edge of P_{bg} and it is positive blue, then e^* is an outgoing red and incoming green edge for v^*; if e is positive blue and negative green, then e^* is an outgoing red edge for v^*; if e is negative green, then e^* is an outgoing red and incoming blue edge for v^*;
- if e is an edge of p_{rg}, e^* is an incoming blue edge for v^*;
- if e is the edge of P_{rb} and it is positive red, then e^* is an outgoing green and incoming blue edge for v^*; if e is positive red and negative blue, then e^* is an outgoing green edge for v^*; if e is negative blue, then e^* is an outgoing green and incoming red edge for v^*;
- if e is an edge of p_{gb}, e^* is an incoming red edge for v^*;
- if e is the edge of P_{gr} and it is positive green, then e^* is an outgoing blue and incoming red edge for v^*; if e is positive green and negative red, then e^* is an outgoing blue edge for v^*; if e is negative red, then e^* is an outgoing blue and incoming green edge for v^*;
- if e is an edge of p_{br}, e^* is an incoming green edge for v^*;
- (v_b^*, v_g^*) is an outgoing green edge for v_b^* and an outgoing blue edge for v_g^*; (v_g^*, v_r^*) is an outgoing red edge for v_g^* and an outgoing green edge for v_r^*; (v_r^*, v_b^*) is an outgoing blue edge for v_r^* and an outgoing red edge for v_b^*.

Let i, j, and k be three consecutive colors in the circularly ordered set $\{b, g, r\}$. We prove that, for each color k, the k-colored edges form a spanning tree T_k^* of G^*. Each vertex v^* of G^*, different from v_k^* has exactly one k-colored outgoing edge. For each face of G such that p_{ji} is not empty, i.e., for each vertex v^* of G^* which is not a leaf in T_k^*, let $v_1, v_2, \ldots, v_d$ be the vertices of p_{ji} and let u_1 and u_2 be the endpoints of P_{ij}. From the coloring of p_{ji} and P_{ij} in Lemma 11, it follows that Case 4 of Lemma 10 applies for v_1 and $v_2, \ldots, v_{d-1}$ and v_d, and for u_1 and u_2. Still from Lemma 11, it follows that either Case 1 or Case 2 of Lemma 10 applies for u_1 and v_1. Then $R_k(u_1) = R_k(u_2) \subset R_k(v_1) = R_k(v_2) = \ldots = R_k(v_d)$, hence there are no k-colored cycles.

As for Properties 1–6 of the realizers, they easily follow from the coloring above and from Lemmas 11 and 12. □

4 Planar 3-Path Queries

In this section we apply the combinatorial results of Section 3 to devise a data structure that supports output-sensitive 3-path queries on a triconnected planar graph. The algorithm and its underlying data structure are simple to implement.

4.1 Preprocessing

In order to simplify the algorithm, we use a *single-sink realizer* for a triconnected planar graph G, i.e., a realizer in which a common vertex s of degree three is chosen as s_b, s_g and s_r. If G has no vertex of degree three, we first apply the algorithm of Nagamochi and Ibaraki [34] to obtain a sparse triconnected spanning subgraph G' of G, which is guaranteed to have a vertex of degree three (see Lemma 2.6 of [34]). Otherwise, G' is identical to G. Then, a realizer of G' is computed, as shown in the proof of Lemma 3, with $v_1 = s$. A realizer of G' is also a realizer of G. Finally, a single-sink realizer of G' is obtained in the following way: let (s_g, w_g) be the edge following (s_g, s_b) in the clockwise order around s_g, and let (s_r, w_r) be the edge preceding (s_r, s_b) in the clockwise order around s_r; (s_g, s_b) is made an outgoing green edge for s_g, (s_g, w_g) is made an outgoing blue edge for s_g and an outgoing green edge for w_g; (s_r, s_b) is made an outgoing red edge for s_r; (s_r, w_r) is made an outgoing blue edge for s_r and an outgoing red edge for w_r; s_g and s_r are identified with s_b.

Note that the single-sink realizer of G' induces a realizer of the subgraph obtained from G' by removing s_b and its three incident edges. The three distinct sinks of the induced realizer are the three vertices adjacent to s_b in G'. Such induced realizer satisfies all the properties of the realizers described in Section 3.

4.2 Three Disjoint Paths

Let G be a triconnected plane graph equipped with a single-sink realizer. To answer a 3-path query for vertices u and v of G, we assemble three paths between u and v by suitably traversing the paths $p_i(u)$, $p_i(v)$, $i \in \{b, g, r\}$. Since such paths can share vertices and edges, a careful choice is needed.

In the rest of paper the following notation is used. The concatenation of two paths $p_i(u, w)$ and $p_j(v, w)$, $i, j \in \{b, g, r\}$, $i \neq j$, having only vertex w in common is denoted by $p_i(u, w) + p_j(v, w)$. If $w = s_i = s_j$, the concatenation of paths $p_i(u)$ and $p_j(v)$ is denoted by $p_i(u) + p_j(v)$. If $p_i(v)$ and $p_j(u)$, $i \neq j$ have a common subpath, then we define $stopvertex_{ij}(u, v)$ any vertex of the subpath; in the rest of the paper we will use $stopvertex_{ij}$ instead of $stopvertex_{ij}(u, v)$ for brevity, and in the figures we will use sv_{ij} instead of $stopvertex_{ij}$.

Lemma 14 *For each pair of vertices u and v of G, the subgraph of G formed by the six paths $p_b(u)$, $p_g(u)$, $p_r(u)$, $p_b(v)$, $p_g(v)$, and $p_r(v)$ contains three disjoint paths between u and v.*

Proof: Let (s_b, x) be the blue incoming edge of s_b, (s_b, y) be the green incoming edge of s_b, and (s_b, z) be the red incoming edge of s_b. As noted in Section 4.1, the single-sink realizer of G induces a realizer of the subgraph obtained from G by removing s_b and its three incident edges. In the induced realizer, x is the blue sink, y is the green sink, and z is the red sink.

We first consider the case in which either u or v is coincident with s_b. W.l.o.g., let u be this vertex. By Lemma 6, $p_b(v, x)$, $p_g(v, y)$, and $p_r(v, z)$ have only vertex v in common; thus the three disjoint paths between u and v are simply $p_b(v)$, $p_g(v)$, and $p_r(v)$.

We then consider the case in which neither u nor v coincides with s_b. By Lemma 9, two cases are possible for u and v:

1. If Case 1 of Lemma 9 applies, then there are exactly two colors $i, j \in \{b, g, r\}$, $i \neq j$ such that paths $p_i(v)$ and $p_j(u)$ cross. These two paths are exploited to determine a first path with endpoints u and v. A second path is determined using $p_j(v)$ and $p_i(u)$. Let $k \neq i, j$ be the other color in $\{b, g, r\}$. The third path is the one along T_k. More formally, the three disjoint paths between u and v are the following (see Fig. 8.a–8.c, where the portions of the paths that are used to assemble the three disjoint paths are thicker):

   - $p_1 = p_k(u, lca_k) + p_k(v, lca_k)$;
   - $p_2 = p_i(u) + p_j(v)$;
   - $p_3 = p_j(u, stopvertex_{ij}) + p_i(v, stopvertex_{ij})$.

   Note that, if $i, j \neq b$, the crossing between $p_i(v)$ and $p_j(u)$ may be external.

2. If Case 2 of Lemma 9 applies, then there is exactly one color $k \in \{b, g, r\}$ such that either $p_k(u) \subset p_k(v)$ or $p_k(v) \subset p_k(u)$. W.l.o.g., let $p_k(u) \subset p_k(v)$, and let $i, j \neq k$ be the other two colors in $\{b, g, r\}$. The three paths are the non-common part of $p_k(u)$ and $p_k(v)$, and the

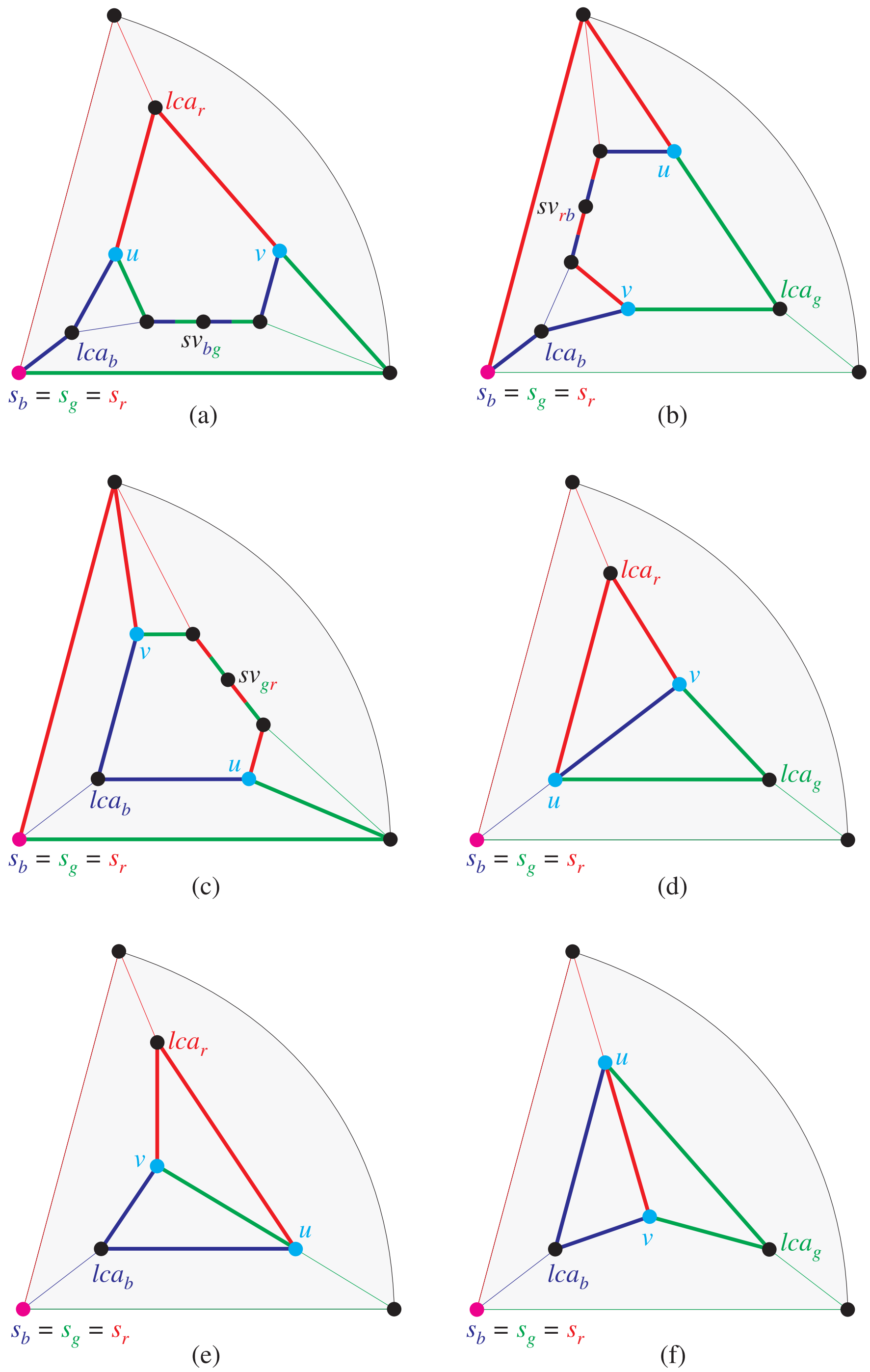

Figure 8: The cases of disjoint paths with endpoints u and v.

paths along T_i and along T_j. More formally, the three disjoint paths between u and v are the following (see Fig. 8.d–8.f):

- $p_1 = p_i(u, lca_i) + p_i(v, lca_i)$;
- $p_2 = p_j(u, lca_j) + p_j(v, lca_j)$;
- $p_3 = p_k(v, u)$.

We now prove the disjointness of p_1, p_2, and p_3 in both cases. Of the six possible choices of colors for i, j, and k in each case, we will consider only one; the proof for the other choices is analogous.

Case 1. Let $i = g$, $j = r$, and $k = b$ (see Fig. 8.c). It is easy to see that neither lca_b nor $stopvertex_{gr}$ coincides with s_b; thus, in proving the disjointness of the three paths we can consider $p_g(u, y)$ instead of $p_g(u)$, and $p_r(v, z)$ instead of $p_r(v)$.

First, we prove that p_1 and p_2 are disjoint. By Lemma 6, $p_b(u, lca_b)$ and $p_g(u, y)$ are disjoint. By Case 1 of Lemma 9, and since $p_r(u)$ and $p_g(v)$ cross, $p_b(u, lca_b)$ and $p_r(v, z)$ are disjoint. Analogously, $p_b(v, lca_b)$ and $p_r(v, z)$ are disjoint, and $p_b(v, lca_b)$ and $p_g(u, y)$ are disjoint.

Second, we prove that p_1 and p_3 are disjoint. By Lemma 6, $p_b(u, lca_b)$ and $p_r(u, stopvertex_{gr})$ are disjoint. By Case 1 of Lemma 9 and since $p_r(u)$ and $p_g(v)$ cross, $p_b(u, lca_b)$ and $p_g(v, stopvertex_{gr})$ are disjoint. Analogously, $p_b(v, lca_b)$ and $p_g(v, stopvertex_{gr})$ are disjoint, and $p_b(v, lca_b)$ and $p_r(u, stopvertex_{gr})$ are disjoint.

Third, we prove that p_2 and p_3 are disjoint. By Lemma 6, $p_r(u, stopvertex_{gr})$ and $p_g(u, y)$ are disjoint. Since $p_r(u)$ and $p_g(v)$ cross, it follows from Corollary 1 that $p_r(u, stopvertex_{gr}) \subset R_r(v)$, while, by Lemma 6, $p_r(v) \cap R_r(v) = \{v\}$; hence, $p_r(u, stopvertex_{gr})$ and $p_r(v, z)$ are disjoint. Analogously, $p_g(v, stopvertex_{gr})$ and $p_r(v, z)$ are disjoint, and $p_g(v, stopvertex_{gr})$ and $p_g(u, y)$ are disjoint.

Finally, we prove that p_1, p_2, and p_3 are simple paths. Path p_1 is composed by the two paths along T_b between u or v and their lowest common ancestor. Path p_2 is simple by being $p_r(v)$ and $p_g(u)$ simple and non-crossing. Path p_3 is simple by being $p_r(u)$ and $p_g(v)$ simple and by Lemma 7.

Case 2. Let $i = g$, $j = r$, and $k = b$ (see Fig. 8.d). It is easy to see that neither lca_g nor lca_r coincides with s_b.

First, we prove that p_1 and p_2 are disjoint. By Lemma 6, $p_g(u, lca_g)$ and $p_r(u, lca_r)$ are disjoint. By Case 2 of Lemma 9, and by being $p_b(u)$ a proper subpath of $p_b(v)$, $p_g(u, lca_g)$ and $p_r(v, lca_r)$ are disjoint. Analogously, $p_g(v, lca_g)$ and $p_r(v, lca_r)$ are disjoint, and $p_r(u, lca_r)$ and $p_g(v, lca_g)$ are disjoint.

Second, we prove that p_1 and p_3 are disjoint. By Lemma 6, $p_g(v, lca_g)$ and $p_b(u, v)$ are disjoint. By Case 2 of Lemma 9 and by being $p_b(u)$ a proper subpath of $p_b(v)$, $p_g(u, lca_g)$ and $p_b(u, v)$ are disjoint.

Third, we prove that p_2 and p_3 are disjoint. By Lemma 6, $p_r(v, lca_r)$ and $p_b(u, v)$ are disjoint. By Case 2 of Lemma 9 and by being $p_b(u)$ a proper subpath of $p_b(v)$, $p_r(u, lca_r)$ and $p_b(u, v)$ are disjoint.

Finally, to prove that p_1, p_2, and p_3 are simple paths, we observe that p_1 is composed by the two paths along T_g between u or v and their lowest common ancestor, p_2 is composed by the two paths along T_r between u or v and their lowest common ancestor, and that p_3 is a subpath of the simple path $p_b(v)$. □

4.3 Data Structure and Complexity

In this section we present a data structure for performing 3-path queries on a triconnected planar graph G with n vertices. By Lemma 3 and by Theorem 2.1 of [34], we assume that G has been embedded and a single-sink realizer T_b, T_g, T_r of G has been constructed; this can be done in $O(n)$ time.

Trees T_b, T_g, and T_r are implemented with parent pointers. We then add to those trees a component for computing $stopvertex_{ij}$ and lca_k, $i, j, k \in \{b, g, r\}$.

For this purpose, we define for each tree T_i, $i \in \{b, g, r\}$, a binary relations $\downarrow_i$ on the vertex set of G. For a pair of vertices $\{u, v\}$, such relations determine the relative positions of u and v in T_i.

Namely, $u \downarrow_i v$ if u is a vertex of the subtree of T_i rooted at v.

We implement relations $\downarrow_i$, $i \in \{b, g, r\}$, by associating to each vertex w of T_i three positive integers: $min_i(w)$, $max_i(w)$, and $level_i(w)$. We consider the leaves of T_i in the order induced by a visit of T_i: for the first leaf, we set $min_i(w) = max_i(w) = 1$; for the second leaf, we set $min_i(w) = max_i(w) = 2$, etc. For any other vertex of T_i, we set $min_i(w)$ equal to the value of its first descendant leaf, and $max_i(w)$ equal to the value of its last descendant leaf. Besides, we set $level_i(s_i) = 1$, and, for any other vertex w of T_i with parent x, we set $level(w) = level(x) + 1$.

The binary relation can be easily tested in the following way. For each pair of vertices u and v of G, $u \downarrow_i v$ if and only if one of the following three cases apply:

- $min_i(u) > min_i(v)$ and $max_i(u) \leq max_i(v)$;
- $min_i(u) \geq min_i(v)$ and $max_i(u) < max_i(v)$;
- $min_i(u) = min_i(v)$ and $max_i(u) = max_i(v)$ and $level_i(u) \geq level_i(v)$.

Thus, testing if $u \downarrow_i v$ can be done in $O(1)$ time. It is easy to see that the above data structure can be constructed in $O(n)$ time.

Using this data structure, we can compute the three disjoint paths between u and v. The two cases in which either u or v coincides with s_b are trivial. In all other cases, we consider vertex u and first traverse path $p_b(u)$ until one of the following halting events occurs:

- lca_b is reached;
- $stopvertex_{bg}$ or $stopvertex_{rb}$ is reached.

In particular, testing whether we have reached lca_b, requires testing, for each vertex w of $p_b(u)$, if $v \downarrow_b w$. Testing whether we have reached $stopvertex_{bg}$ or $stopvertex_{rb}$, requires testing, for each vertex w of $p_b(u) - \{u\}$, if $v \downarrow_g w$ or $v \downarrow_r w$, respectively. Note that lca_b can coincide with u, and that $stopvertex_{bg}$ or $stopvertex_{rb}$ cannot coincide with u (but can coincide with v).

We then traverse $p_g(u)$ and $p_r(u)$ in the same way. At the end of the process, if we have reached one $stopvertex_{ij}$, then Case 1 in the proof of Lemma 14 applies, else Case 2 applies. During this process, we have only visited vertices and edges which are contained in the three disjoint paths between u and v. The report of the three paths can now be completed by suitably traversing $p_b(v)$, $p_g(v)$, and $p_r(v)$ and by possibly continuing the traversal of one path among $p_b(u)$, $p_g(u)$, and $p_r(u)$.

Theorem 2 *Let G be a triconnected planar graph with n vertices. There exists an $O(n)$-space data structure for G that can be constructed in $O(n)$ time and supports 3-path queries in $O(\ell)$ time, where ℓ is the size of the reported paths.*

5 General 3-Path Queries

In this section we extend to general triconnected graphs the results on planar triconnected graphs of Section 4.

5.1 Preprocessing

The realizer used for triconnected planar graphs is replaced by three independent spanning trees [5, 54]. For three independent spanning trees of a triconnected graph G, the following properties hold:

1. In each spanning tree, the edges of G are directed from children to parent.

2. The sinks (roots) of the spanning trees are three (possibly coincident) vertices of G.

3. Each edge of G is contained in at least one and in at most two spanning trees.

4. If an edge of G is contained in two spanning trees, then it has different directions in the two trees.

5. For each vertex v of G, the paths from v to the sinks along the three spanning trees have only vertex v in common.

We briefly review the algorithm by Cheriyan and Maheshwari [5] for constructing three independent spanning trees of a triconnected graph G with n vertices.

The main step of their algorithm is the computation of a nonseparating ear decomposition of the triconnected graph. An *ear decomposition* of a graph G is a partition of G into an ordered collection of edge-disjoint simple paths $P_0, P_1, \ldots, P_h$, such that P_0 is a cycle, and each P_k, $1 \leq k \leq h$ has only its two distinct endpoints in common with $G_{k-1} = P_0 \cup P_1 \cup \ldots \cup P_{k-1}$. Each path P_k is an *ear*. An ear decomposition is said to be *through edge* (v_1, v_2) *and avoiding vertex* v_n if cycle P_0 contains edge (v_1, v_2), and the last ear $P_{h'}$ different from a single edge contains vertex v_n as its only internal vertex. An ear decomposition through edge (v_1, v_2) and avoiding vertex v_n is said a *nonseparating ear decomposition* if, for each $0 \leq k \leq h'$, graph $G - G_k$ is connected and each internal vertex of ear P_k has at least one neighbor in $G - G_k$.

A nonseparating ear decomposition has at most n ears different from a single edge. For each vertex v of G, we define the *ear number* $ear(v)$ as the index k of the first ear in $P_0, P_1, \ldots, P_h$ containing v.

Given an ear decomposition of G and an edge (s, t) of the first ear P_0, an st-numbering of G is *consistent* with the ear decomposition if, for each $1 \leq k \leq h$, the numbering induced by G_k is an st-numbering of G_k. For each vertex v of G, we indicate with $stn(v)$ the st-number of v.

Note that the canonical ordering defined in Section 2.3 is a particular case of nonseparating ear decomposition for triconnected planar graphs.

Lemma 15 **[5]** *Let G be a triconnected graph with n vertices and m edges. Let (v_1, v_2) be an edge and $v_n \neq v_1, v_2$ be a vertex of G. There exists a nonseparating ear decomposition of G through (v_1, v_2) and avoiding v_n. It can be computed in $O(nm)$ time and $O(m)$ space.*

The time complexity of the algorithm can be reduced from $O(nm)$ to $O(n^2)$ by computing a sparse triconnected spanning subgraph G' of G in $O(m)$ time [34] and by then computing a nonseparating ear decomposition of G'. As noted in Section 4.1, G' is guaranteed to have a vertex of degree three.

The three independent spanning trees can be constructed in the following way:

1. let v_1 be a vertex of degree three, and let v_2, and v_n be two vertices adjacent to v_1; a nonseparating ear decomposition of G' through (v_1, v_2) and avoiding v_n is computed;

2. let $s = v_1$ and $t = v_2$; an st-numbering of G consistent with the ear decomposition is computed;

3. v_1, v_2, and v_n are the sinks of the blue, green, and red tree, respectively;

4. (v_1, v_2) is an outgoing blue edge for v_2 and an outgoing green edge for v_1;

5. for each $1 \leq k \leq h$, let c_l and c_r be the two endpoints of ear P_k, such that, either $ear(c_l) < ear(c_r)$, or $ear(c_l) = ear(c_r)$ and $stn(c_l) < stn(c_r)$; two cases are possible:
   (a) if P_k is a single edge, then (c_l, c_r) is an ougoing red edge for c_l;
   (b) if P_k is not a single edge, let $c_r, v_{s_k}, \ldots, v_{s_k+d_k}, c_l, d_k \geq 0$, be the consecutive vertices of P_k; $(v_{s_k+d_k}, c_l)$ is an outgoing blue edge for $v_{s_k+d_k}$, and possibly an outgoing red edge for c_l if c_l has no neighbor in $G - G_k$; (v_{s_k}, c_r) is an outgoing green edge for v_{s_k}, and possibly an outgoing red edge for c_r if c_r has no neighbor in $G - G_k$; edge (v_i, v_{i+1}), $s_k \leq i < s_k + d_k$ is an outgoing blue edge for v_i and an outgoing green edge for v_{i+1}.

As for the planar case, we denote v_1, v_2, and v_n as s_b, s_g, and s_r, respectively. Properties 1–4 of the independent spanning trees immediately follow from the previous construction, while Property 5 can be proved by observing that, from the previous construction, it follows:

- for each vertex $v \neq s_b$ of G, let x be the parent of v in T_b; $ear(u) \leq ear(v)$ and $stn(x) < stn(v)$;
- for each vertex $v \neq s_g$ of G, let y be the parent of v in T_g; $ear(y) \leq ear(v)$ and $stn(y) > stn(v)$;
- for each vertex $v \neq s_r$ of G, let z be the parent of v in T_r; $ear(z) > ear(v)$.

In the rest of the section, we will use three independent spanning trees with a common sink, which can be obtained from the three independent spanning trees computed above in the following way: let (s_g, w_g) be the edge of P_0 incident with s_g and different from (s_g, s_b); (s_g, s_b) is made an outgoing green edge for s_g; (s_g, w_g) is made an outgoing blue edge for s_g and an outgoing green edge for w_g; note that, by construction, (s_r, s_b) is an outgoing red edge for s_r; s_g and s_r are identified with s_b.

5.2 Three Disjoint Paths

Lemma 16 *For each pair of vertices u and v of G, if there are two colors $i, j \in \{b, g, r\}$, $i \neq j$, such that $p_i(v)$ and $p_j(u)$ cross, then $p_j(v)$ and $p_i(u)$ do not cross.*

Proof: Six cases are possible for i and j:

1. $i = b$ and $j = g$; let w be a vertex of the crossing between $p_b(v)$ and $p_g(u)$, $x \neq u$ be a vertex of $p_b(u)$, and $y \neq v$ be a vertex of $p_g(v)$; $stn(x) < stn(u) < stn(w) < stn(v) < stn(y)$ holds;
2. $i = g$ and $j = b$; analogous to Case 1;
3. $i = g$ and $j = r$; let w be a vertex of the crossing between $p_g(v)$ and $p_r(u)$, $y \neq u$ be a vertex of $p_g(u)$, and $z \neq v$ be a vertex of $p_r(v)$; $ear(y) \leq ear(u) \leq ear(w) \leq ear(v) < ear(z)$ holds;
4. $i = r$ and $j = g$; analogous to Case 3;
5. $i = r$ and $j = b$; let w be a vertex of the crossing between $p_r(v)$ and $p_b(u)$, $x \neq v$ be a vertex of $p_b(v)$, and $z \neq u$ be a vertex of $p_r(u)$; $ear(x) \leq ear(v) \leq ear(w) \leq ear(u) < ear(z)$ holds;
6. $i = b$ and $j = r$; analogous to Case 5.

In all the cases, it is easy to see that $p_j(v)$ and $p_i(u)$ do not cross. □

With analogous techniques, we can prove the following two lemmas.

Lemma 17 *For each pair of vertices u and v of G, if $p_r(v)$ and $p_i(u)$, $i \in \{b, g\}$, cross, then $p_r(u)$ and $p_j(v)$, $j \in \{b, g\}$, $j \neq i$, do not cross.*

Lemma 18 *For each pair of vertices u and v of G, $p_b(v)$ and $p_g(u)$, or $p_g(v)$ and $p_b(u)$, may cross at most once, $p_b(v)$ and $p_r(u)$, or $p_r(v)$ and $p_b(u)$, may cross multiple times, $p_g(v)$ and $p_r(u)$, or $p_r(v)$ and $p_g(u)$, may cross multiple times.*

We now state the equivalent, for general triconnected graphs, of Lemma 9 for planar graphs. Note how, being the graph nonplanar, the number of possible cases has increased.

Lemma 19 *For each pair of vertices u and v of G, six cases are possible:*

1. *there are three colors $i, k \in \{b, g, r\}$, $j \in \{b, g\}$, $i \neq j \neq k$, such that $p_i(v)$ and $p_j(u)$, $p_i(v)$ and $p_k(u)$, $p_j(v)$ and $p_k(u)$ cross;*
2. *there are three colors $i, j, k \in \{b, g, r\}$, $i \neq j \neq k$, such that $p_i(v)$ and $p_j(u)$, $p_i(v)$ and $p_k(u)$ cross;*
3. *there are three colors $i, j, k \in \{b, g, r\}$, $i \neq j \neq k$, such that $p_j(v)$ and $p_i(u)$, $p_k(v)$ and $p_i(u)$ cross (analogous to Case 2 with u and v switched);*
4. *there are three colors $i \in \{b, g\}$, $j, k \in \{b, g, r\}$, $i \neq j \neq k$, such that $p_i(v)$ and $p_j(u)$, $p_k(v)$ and $p_i(u)$ cross;*
5. *there are exactly two colors $i, j \in \{b, g, r\}$, $i \neq j$, such that $p_i(v)$ and $p_j(u)$ cross;*
6. *there are no two colors $i, j \in \{b, g, r\}$, $i \neq j$, such that $p_i(v)$ and $p_j(u)$ cross.*

Proof: By Lemma 16, out of the six potential crossings between differently colored paths from u and v, at most three may exist. It is easy to see that, by Lemmas 16 and 17, the six claimed cases are exhaustive. □

As for planar graphs, we define $stopvertex_{ij}$, $i, j \in \{b, g, r\}$, $i \neq j$, any vertex of the crossing between $p_i(v)$ and $p_j(u)$ or between $p_j(v)$ and $p_i(u)$.

Lemma 20 *For any two vertices u and v of G, the subgraph of G formed by the six paths $p_b(u)$, $p_b(v)$, $p_g(u)$, $p_g(v)$, $p_r(u)$, and $p_r(v)$ contains three disjoint paths between u and v.*

Proof: We will prove the claim by considering Cases 1–4 of Lemma 19. Cases 5 and 6 are analogous to those of Lemma 14 for planar graphs.

We will prove in detail only Case 1 of Lemma 19, in which three crossings occur between differently colored paths from u and v. This is the most complex case. The proofs for Cases 2–4, in which two crossings occur, are similar.

Path $p_i(v)$ crosses both $p_j(u)$ and $p_k(u)$; path $p_k(u)$ crosses both $p_i(v)$ and $p_j(v)$. W.l.o.g., let $i = g$, $j = b$, and $k = r$. We first prove that $stopvertex_{gr}$ "is closer to v" along $p_g(v)$ than $stopvertex_{bg}$, or, more formally, that $p_g(v, stopvertex_{gr}) \subset p_g(v, stopvertex_{bg})$. This follows from $ear(stopvertex_{bg}) \leq ear(u) < ear(stopvertex_{gr}) \leq ear(v)$.

Then we consider $stopvertex_{gr}$ and $stopvertex_{bg}$. Two cases are possible:

1. $stopvertex_{gr}$ "is closer to u" along $p_r(u)$ than $stopvertex_{rb}$, or, more formally, $p_r(u, stopvertex_{gr}) \subset p_r(u, stopvertex_{rb})$; the three disjoint paths are, as in the planar case, the following:

- $p_1 := p_b(u, lca_b) + p_b(v, lca_b)$;
- $p_2 := p_g(u) + p_r(v)$;
- $p_3 := p_r(u, stopvertex_{gr}) + p_g(v, stopvertex_{gr})$;

since we use neither $p_g(stopvertex_{gr})$ nor $p_r(stopvertex_{gr}) - p_g(v)$ in the construction of the three disjoint paths, we can simply ignore $stopvertex_{bg}$ and $stopvertex_{rb}$;

2. $stopvertex_{rb}$ "is closer to u" along $p_r(u)$ than $stopvertex_{gr}$, or, more formally, $p_r(u, stopvertex_{rb}) \subset p_r(u, stopvertex_{gr})$; in this case it is not possible to construct the three disjoint paths as in the planar case; however three disjoint paths still exist:

   - $p_1 := p_g(u) + p_r(v)$;
   - $p_2 := p_b(u, stopvertex_{bg}) + p_g(v, stopvertex_{bg})$;
   - $p_3 := p_r(u, stopvertex_{rb}) + p_b(v, stopvertex_{rb})$;

   since we do not use $p_r(stopvertex_{rb}) - p_g(v)$ in the construction of the three disjoint paths, we can simply ignore $stopvertex_{gr}$.

In both cases, the disjointness of p_1, p_2, and p_3 can be easily proved by the ear number and st-number properties of the colored paths from u and v. □

5.3 Data Structure and Complexity

In this section we present a data structure for performing 3-path queries in a triconnected graph G with n vertices. By Lemma 15 and by Theorem 2.1 of [34], we assume that G has been embedded and three independent spanning trees T_b, T_g, and T_r of G with a common sink have been constructed; this can be done in $O(n^2)$ time.

As for planar graphs, trees T_b, T_g, and T_r are implemented with parent pointers, and are augmented with the component implementing the relations $\downarrow_i$, $i \in \{b, g, r\}$ on the vertex set of G. It is easy to see that the above data structure can be constructed in $O(n)$ time.

Using this data structure, we can compute the three disjoint paths between u and v similarly to the planar case. However, by Lemma 18, $p_r(v)$ may cross $p_b(u)$ and $p_g(u)$ multiple times, or $p_r(u)$ may cross $p_b(v)$ and $p_g(v)$ multiple times. In the proof of Lemma 20, we have seen that only the first crossing, if any, found traversing $p_r(u)$ from u or $p_r(v)$ from v need be considered. This imply that, differently from the planar case, the first traversed path is $p_r(u)$; the halting events for the traversal are the same of the planar case; if a crossing with $p_b(v)$ or $p_g(v)$ is found, then we continue by traversing $p_b(u)$ or $p_g(u)$, otherwise we switch to v and first traverse $p_r(v)$, and then traverse $p_b(v)$ and $p_g(v)$.

At the end of the process, we have all the necessary information to decide which of the cases of Lemma 19 applies. We have only visited vertices and edges which are contained in the three disjoint paths between u and v. The report of these paths can now be completed by suitably traversing the remaining paths from u or v and by possibly continuing the traversal of some of the already traversed paths.

Theorem 3 *Let G be a triconnected graph with n vertices. There exists an $O(n)$-space data structure for G that can be constructed in $O(n^2)$ time and supports 3-path queries in $O(\ell)$ time, where ℓ is the size of the reported paths.*

6 Graphs of Arbitrary Connectivity

In this section we extended the results of Theorems 2 and 3 to graphs of arbitrary connectivity $k < 3$.

We first consider biconnected (non-triconnected) graphs. We use a suitably augmented version of the *SPQR-tree* data structure for 3-connectivity queries [14]. A description of the SPQR-tree is contained in Appendix B for the reader's convenience. An example of SPQR-tree is shown in Fig. 9.

Let G be a biconnected graph with n vertices and m edges, and let T be an SPQR-tree of G. Each R-node μ of T is equipped with a realizer of $skeleton(\mu)$. If G is nonplanar, then, for each R-node μ of T, instead of storing $skeleton(\mu)$, we store a sparse triconnected spanning subgraph of

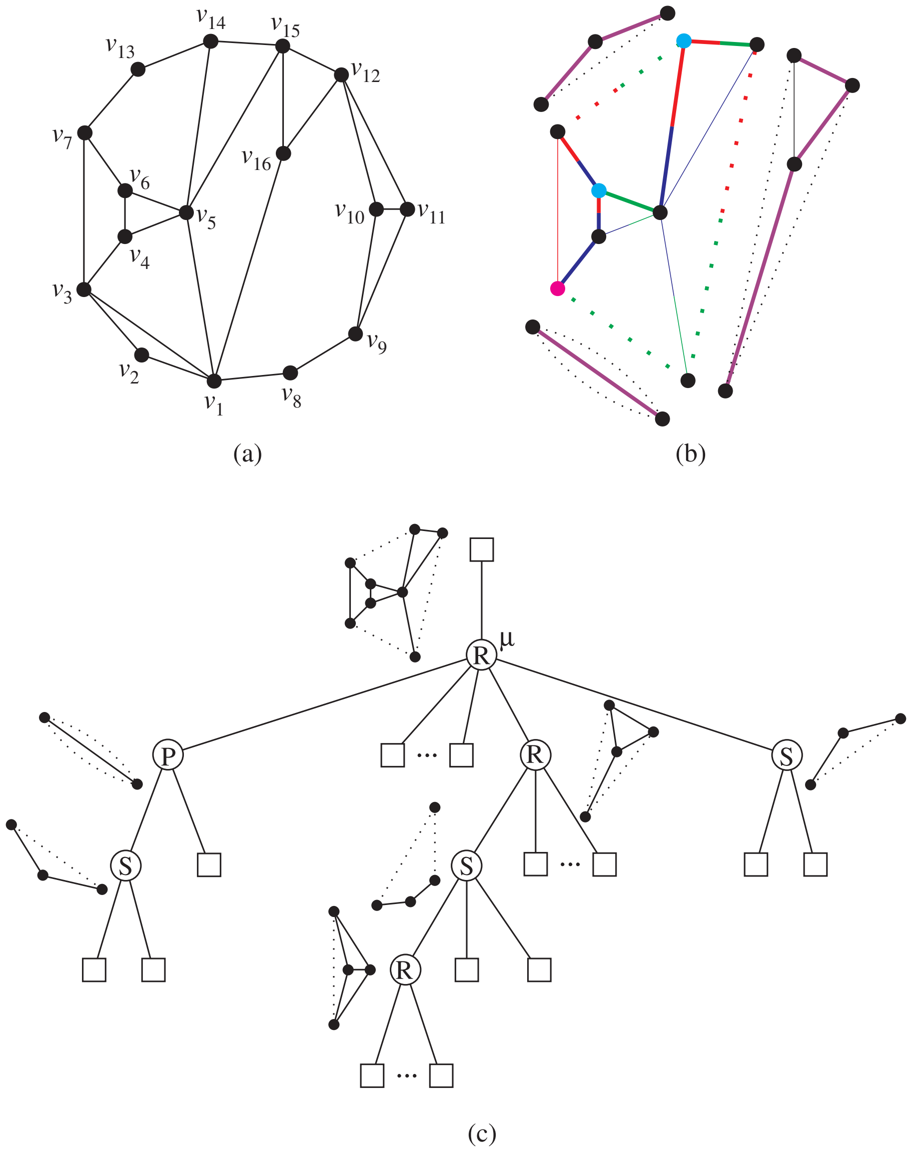

Figure 9: (a) A biconnected graph G. (b) The split components used in the reporting of three disjoint paths between vertices v_6 and v_{14} of G. (c) The SPQR-tree of G with respect to reference edge (v_3, v_7) and the skeletons of its nodes.

skeleton(μ) [34]; this reduces the space requirements to $O(n)$. Computing the spanning subgraphs requires an $O(m)$ total time.

As usual, let u and v be the two vertices on which we want to perform a 3-paths query. We first perform a 3-connectivity query on u and v as shown in [14].

Lemma 21 [14] *A 3-connectivity query on vertices u and v returns true if and only if there is a P-node or an R-node χ of T such that u and v are both allocated at χ. Node χ can be determined in $O(1)$ time.*

If the 3-connectivity query on vertices u and v returns true, the 3-path query can be answered as follows.

If χ is a P-node, u and v are the poles of χ and the endpoints of at least three virtual edges in *skeleton*(μ). Three disjoint paths between u and v in *skeleton*(μ) are obtained by taking three of these virtual edges. Note that, since we are considering simple graphs, at least two of these three virtual edges are non-trivial.

If χ is an R-node, we determine three disjoint paths between u and v in *skeleton*(μ) as shown in Sections 4 and 5. In general, these three paths contain some non-trivial virtual edges (see Fig. 9b).

In both cases, let $p_{\mu 1}$, $p_{\mu 2}$, and $p_{\mu 3}$ be the three disjoint paths between u and v in *skeleton*(μ). Three disjoint paths between u and v in G can be obtained from $p_{\mu 1}$, $p_{\mu 2}$, and $p_{\mu 3}$ by recursively replacing each non-trivial virtual edge e_ν, corresponding to a node ν, with a path p_ν between the poles of *skeleton*(ν). The graph can be preprocessed so that for each node ν of T a path p_ν in *skeleton*(ν) between its poles (different from the virtual edge corresponding to the parent of ν) is stored. In the example of Fig. 9b, paths p_ν are represented with the purple color.

It remains to be proved that this recursive process requires (ℓ) time. We need the following lemma.

Lemma 22 [14] *Two S-nodes cannot be adjacent in T. Two P-nodes cannot be adjacent in T.*

During the recursive process, each virtual edge e_ν contained in one of the three paths is replaced with a path p_ν. Path p_ν contains exactly one edge and this edge is non-trivial virtual only if ν is a P-node. In all other cases, p_ν is either a trivial virtual edge or contains more than one edge. Thus, by Lemma 22, the total number of virtual edges substituted with a path during the recursive process, i.e., the total number of nodes of T visited, is $O(\ell)$.

If, on the contrary, the 3-connectivity query on vertices u and v returns false, we can answer a 2-path query using the data structure of Theorem 1.

We now consider connected (non-biconnected) graphs. We use a suitably augmented version of the *BC-tree* data structure for 3-connectivity queries [14]. A description of the BC-tree is contained in Appendix C for the reader's convenience.

Let G be a connected graph and let T be a BC-tree of G. Each B-node of T is equipped with an augmented SPQR-tree described above.

Let again u and v be the two vertices on which we want to perform a 3-paths query. We first perform a 2-connectivity query on u and v as shown in [14].

Lemma 23 [14] *A 2-connectivity query on vertices u and v returns true if and only if there is a B-node χ of T such that u and v are both allocated at χ. Node χ can be determined in $O(1)$ time.*

If the 2-connectivity query on vertices u and v returns true, then we can apply the methods described above for answering a 3-path or a 2-path query.

If, on the contrary, the 2-connectivity query on vertices u and v returns false, we can easily answer a 1-path query using a spanning tree of G.

Finally, we consider non-connected graphs. We use the *BC-forest* data structure, which is a forest of the BC-trees of the connected components of G.

We first perform a 1-connectivity query on u and v simply testing if u and v are both allocated in the same BC-tree of the BC-forest; this can be done in $O(1)$ time. If the 1-connectivity query on vertices u and v returns true, then we can apply the methods described above for answering a 3-path, 2-path, or 1-path query.

The results described in this section can be summarized in the following two theorems.

Theorem 4 *Let G be a planar graph with n vertices. There exists an $O(n)$-space data structure for G that can be constructed in $O(n+m)$ time and supports 1- 2- and 3-path queries in $O(\ell)$ time, where ℓ is the size of the reportedpaths.*

Theorem 5 *Let G be a graph with n vertices. There exists an $O(n)$-space data structure for G that can be constructed in $O(n^2)$ time and supports 1- 2- and 3-path queries in $O(\ell)$ time, where ℓ is the size of the reported paths.*

7 Applications of Realizers to Graph Drawing

In this section we show a graph drawing application for the realizers of triconnected planar graphs.

A *straight-line* drawing is a drawing in which each edge is mapped to a straight-line segment. Planar straight-line drawings of planar graphs are a classical topic in graph drawing (a survey on graph drawing can be found in [13]).

A classical result independently established by Steinitz and Rademacher [41], Wagner [50], Fary [19], and Stein [40] shows that every planar graph has a planar straight-line drawing.

A grid drawing is a drawing such that the vertices have integer coordinates. Independently, de Fraysseix, Pach, and Pollack [11] and Schnyder [37, 38] have shown that every planar graph with n vertices has a planar straight-line grid drawing with $O(n^2)$ area. In particular, they presented algorithms for computing a planar straight-line grid drawing of a maximal planar graph. de Fraysseix, Pach, and Pollack define the canonical ordering for maximal planar graphs; the drawing is constructed by assigning integer coordinates to the vertices according to this canonical ordering. Schnyder defines the realizers for maximal planar graphs, and, based on such realizer, the vertices are assigned integer coordinates in 3D space which have a purely combinatorial meaning and such that all the vertices lie on a plane. A drawing in the plane is then obtained by projection.

Planar straight-line drawings have also been studied with the constraint that all faces be represented by convex polygons (*convex* drawings). Tutte [47, 48] has shown that for a triconnected planar graph a convex drawing can be constructed by solving a system of linear equations. More recently, Kant has presented an algorithm for constructing grid convex drawings with quadratic area [29]. His approach can be seen as the natural extension to triconnected planar graphs of the result by de Fraysseix, Pach, and Pollack for maximal planar graphs. He defines the canonical ordering for triconnected planar graphs recalled in Section 2.3 and the drawing is constructed assigning integer coordinates to the vertices according to this canonical ordering. Recent results on convex grid drawings in the plane and in 3D space are presented in [7].

The realizers we have defined for triconnected planar graphs in Section 3 naturally extend those defined by Schnyder [37, 38] for maximal planar graphs, and can be used to devise a new algorithm for constructing grid convex drawings of triconnected planar graphs with quadratic area, as shown below.

We recall here the definition of weak barycentric representation of a graph given by Schnyder [37, 38]. A *weak barycentric representation* of a graph G is a mapping of each vertex v of G to a distinct point (v_b, v_g, v_r) in 3D space such that the following conditions are satisfied:

1. for each vertex v of G, $v_b + v_g + v_r = c$, where c is a constant dependent on G;

2. for each edge (u, w) and each vertex $v \neq u, w$ of G, there exist coordinates $i, j \in \{b, g, r\}$ such that $(u_i, u_j) <_{lex} (v_i, v_j)$ and $(w_i, w_j) <_{lex} (v_i, v_j)$.

Following Schnyder [37, 38], we can obtain a weak barycentric representation of a triconnected planar graph by using a realizer to assign coordinates to the vertices; these coordinates have a purely combinatorial meaning.

Lemma 24 *Let G be a triconnected planar graph equipped with a realizer. For each vertex v of G, let $l_b(v)$, $l_g(v)$, and $l_r(v)$ be the number of faces in $R_b(v)$, $R_g(v)$, and $R_r(v)$, respectively. The mapping $(v_b, v_g, v_r) = (l_b(v), l_g(v), l_r(v))$ is a weak barycentric representation.*

Proof: Condition 1 of weak barycentric representations is trivially satisfied, since, for each vertex v, $v_b + v_g + v_r = l - 1$, where l is the number of faces of G.

As for Condition 2, let i, j and k be three consecutive colors in the circularly ordered set $\{b, g, r\}$; let (u, w) be an edge of G and $v \neq u, w$ be a vertex of G. In order to simplify the exposition of the proof, we define $\bar{p}_i(v) = p_i(v) - \{v\}$, $\bar{p}_j(v) = p_j(v) - \{v\}$, $\bar{p}_k(v) = p_k(v) - \{v\}$, $\bar{R}_i(v) = R_i(v) - \{p_j(v) \cup p_k(v)\}$, $\bar{R}_j(v) = R_j(v) - \{p_k(v) \cup p_i(v)\}$, and $\bar{R}_k(v) = R_k(v) - \{p_i(v) \cup p_j(v)\}$. W.l.o.g., let $u \in R_k(v)$. If $u \in \bar{R}_k(v)$, then, by planarity of G, $w \notin \bar{R}_i(v)$ and $w \notin \bar{R}_j(v)$. Thus, the following five cases are possible:

1. $u, w \in \bar{R}_k(v)$; by Lemma 10, $R_k(u) \subset R_k(v)$ and $R_k(w) \subset R_k(v)$; hence $u_k < v_k$ and $w_k < v_k$;

2. $u \in \bar{R}_k(v)$ and $w \in p_i(v)$; by Lemma 10, $R_k(u) \subset R_k(v)$, hence $u_k < v_k$; two subcases are possible:

   (a) $v \notin p_j(w)$; by Lemma 10, $R_k(w) \subset R_k(v)$, hence $w_k < v_k$;

   (b) $v \in p_j(w)$; by Lemma 10, $R_k(w) = R_k(v)$, hence $w_k = v_k$; however, still by Lemma 10, $R_j(w) \subset R_j(v)$, hence $w_j < v_j$;

3. $u \in \bar{R}_k(v)$ and $w \in p_j(v)$; analogous to Case 2;

4. $u, w \in p_i(v)$; w.l.o.g., let w be an ancestor of u in T_i; four subcases are possible:

   (a) $u \notin p_j(w)$ and $v \notin p_j(u)$; by Lemma 10, $R_k(w) \subset R_k(u) \subset R_k(v)$ hence $w_k < u_k < v_k$;

   (b) $u \in p_j(w)$ and $v \notin p_j(u)$; by Lemma 10, $R_k(w) = R_k(u) \subset R_k(v)$, hence $w_k = u_k < v_k$;

   (c) $u \notin p_j(w)$ and $v \in p_j(u)$; by Lemma 10, $R_k(w) \subset R_k(u) = R_k(v)$, hence $w_k < u_k = v_k$; however, still by Lemma 10, $R_j(u) \subset R_j(v)$, hence $u_j < v_j$;

   (d) $u \in p_j(w)$ and $v \in p_j(u)$; by Lemma 10, $R_k(w) = R_k(u) = R_k(v)$, hence $w_k = u_k = v_k$; however, still by Lemma 10, $R_j(w) \subset R_j(u) \subset R_j(v)$, hence $w_j < u_j < v_j$;

5. $u, w \in p_j(v)$; analogous to Case 4.

In all five cases, Condition 2 is satisfied. □

Theorem 6 *Let G be a triconnected plane graph with n vertices and l faces. A convex grid drawing of G with height $l-2$ and width $l-2$ can be computed in $O(n)$ time and $O(n)$ space.*

Proof: Let Γ be the straight-line drawing of G resulting from the weak barycentric representation of Lemma 24.

First, note that, by Condition 1 of weak barycentric representations, all the points representing vertices of G lie on plane π in 3D space defined by equation $b+g+r=l-1$; in particular, vertices s_b, s_g, and s_r are mapped to points $(l-1,0,0)$, $(0,l-1,0)$, $(0,0,l-1)$, respectively.

The planarity of Γ follows from Lemma 2.1 in [38].

The convexity of Γ can be proved as follows. Let v be an internal vertex of G. By Condition 1 of weak barycentric representations, if we fix coordinate v_b, then the point representing v lies on line l_b of π which is the projection on π of line $g+r=c_b$ of the g-r plane, where $c_b=l-1-v_b$. Lines l_g and l_r, and constants c_g and c_r are defined in a similar way. Since π intersects the b-, g-, and r-axis at the same coordinate $l-1$, lines l_b, l_g and l_r cross at (v_b,v_g,v_r) and form six $60°$ angles (see Fig. 10). For each line l_i, $i\in\{b,g,r\}$, let the positive (negative) halfplane be the open halfspace containing (not containing) s_i. Let the blue positive (negative) wedge be the portion of the positive (negative) halfspace of l_b delimited by l_g and l_r; the green and red positive (negative) wedges are defined in a similar way. Let x, y and z be the parents of v in T_b, T_g, and T_r, respectively. Thus $v\in R_b(x)$, and by Lemma 10 (Case 1, or 2, or 3 with $k=b$, $i=g$, and $j=r$) $R_b(v)\subset R_b(x)$, hence $v_b<x_b$; still by Lemma 10, $R_g(x)\subseteq R_g(v)$ and $R_r(x)\subseteq R_r(v)$ (where only one equality may hold), hence $x_g\le v_g$ and $x_r\le v_r$ (where only one equality may hold). If we fix coordinate x_b, then the point representing x lies on line l'_b of π which is the projection on π of line $g+r=c'_b$ of the g-r plane, where $c'_b=l-1-x_b<c_b$. Hence, l'_b lies in the positive halfspace of l_b. In a similar way, l'_g lies in the negative halfspace of l_g or $l'_g=l_g$, and l'_r lies in the negative halfspace of l_r or $l'_r=l_r$ (where only one equality may hold). It follows that the point representing x must lie in the positive blue wedge of v. Similarly it can be proved that the point representing y and z must lie in the positive green and red wedges of v, respectively. Hence, no angle incident on v can be greater then $180°$.

As for the external face, the points representing s_b, s_g and s_r are the vertices of an equilateral triangle. Let i, j, and k be three consecutive colors in $\{b,g,r\}$. For each two consecutive vertices u and v of $ext(s_i,s_j)$, $u_k=v_k=0$. It follows that, by Condition 1 of weak barycentric representations, all the vertices of $ext(s_i,s_j)$ are collinear. Thus, also the external face is represented as a convex polygon.

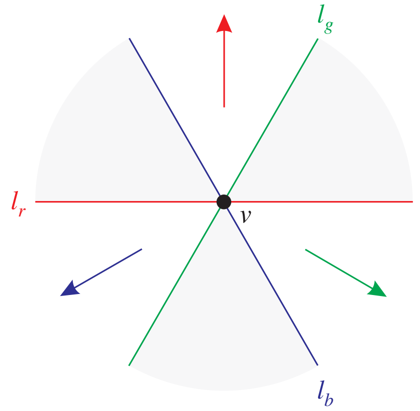

Figure 10: The blue, green, and red wedges of a vertex.

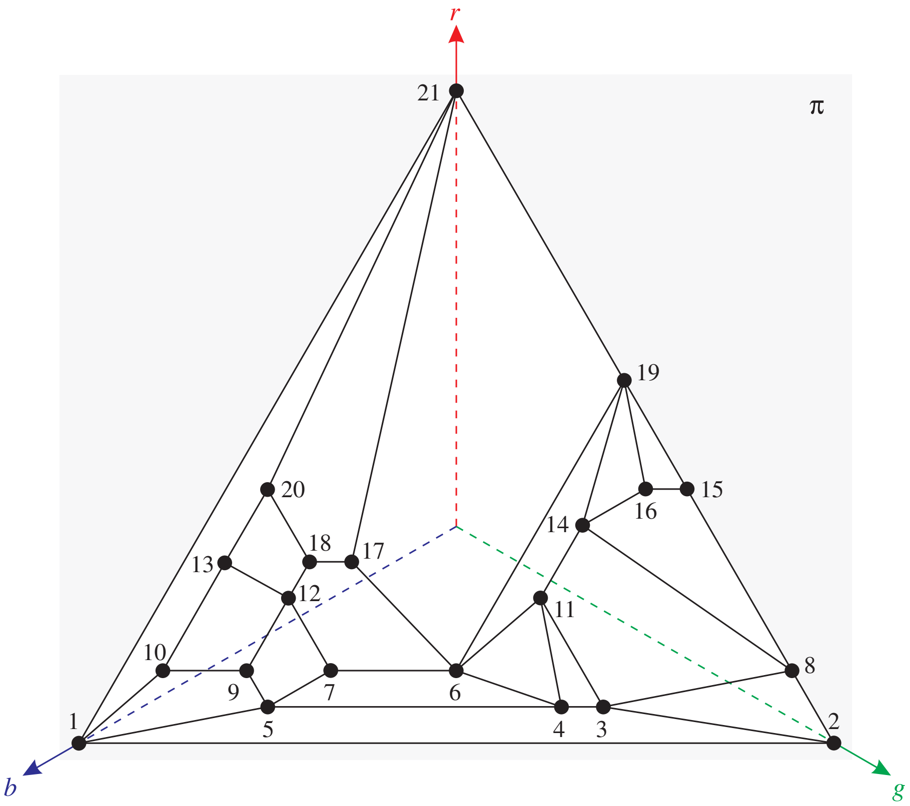

Figure 11: A convex grid drawing of the triconnected plane graph of Fig. 1.

A convex grid drawing with height $l-2$ and width $l-2$ in the plane can be obtained by projecting Γ, e.g., by dropping the red coordinate.

Finally, we prove the time and space complexity. To compute the coordinates we will use both a realizer T_b, T_g, T_r of G, and the induced realizer T_b^*, T_g^*, T_r^* of the extended dual graph G^* of G. The extended dual graph of G^* can be easily constructed in linear time. By Lemma 3, a realizer of G and the induced realizer of G^* can be constructed in linear time and space. Thus we only have to prove that the coordinates for the vertices of G can be computed in linear time. In particular, we will prove that, the number of faces in $R_k(v)$, for each vertex v of G, can be computed by visiting T_i and T_j.

For each vertex v of G we initialize coordinate v_k to $l-1$, i.e., to the number of internal faces of G. We will then subctract from v_k the number of the faces which are not contained in $R_k(v)$; this can be done by visiting T_i and T_j as follows. First we compute, for each vertex v^* of T_k^*, the number of its descendants, including v^* itself, and store it in variable $numdescendants_k(v^*)$; this can be done by a postorder visit of T_k^*. Second, we perform a preorder visit of T_i; we use an auxiliary variable $sumdescendants_i$ initialized to 0. For each edge (u,v) traversed during the visit, let (u^*,v^*) be the dual edge of (u,v), where v^* is the vertex of G^* corresponding to the face of G on the left of (u,v); if $(u^*,v^*) \in T_k^*$, we sum $numdescendants_k(v^*)$ to $sumdescendants_i$ and then subtract $sumdescendants_i$ from coordinate v_k. Third, we perform a similar preorder visit of T_j. The only difference with the previous visit of T_i is that now, for each edge (u,v) traversed during the visit, v^* is the vertex of G^* corresponding to the face of G on the right of (u,v).

It is easy to see that, after the visits of T_i and T_j, for each vertex v of G coordinate v_k is equal to the number of faces in $R_k(v)$. □

A similar result was claimed by Schnyder and Trotter [39], but since then, to the best of our knowledge, no proof has been published.

A convex grid drawing of the triconnected plane graph of Fig. 1 produced by the above algorithm is shown in Fig. 11.

8 Conclusions

The contributions of this paper can be summarized as follows:

- We have defined, analyzed, and shown how to efficiently compute realizers of triconnected planar graphs, a combinatorial structure that unifies and extends various previous constructions. Realizers play for triconnected planar graphs a similar role as bipolar orientations for biconnected planar graphs.
- We have presented the first data structure that supports output-sensitive 2- and 3-path queries in general graphs. The previous best methods for performing queries do not exploit preprocessing and have $O(n)$ time complexity, irrespectively of the output size. Our data structure and query algorithm are both theoretically optimal and practically useful.
- We have presented a new $O(n)$-time algorithm for constructing a convex grid drawing of G with $O(n^2)$ area, which extends to triconnected planar graphs the barycentric drawing method for maximal planar graphs.

References

[1] T. Biedl and G. Kant. A better heuristic for orthogonal graph drawings. In *Algorithms (Proc. ESA '94)*, volume 855 of *Lecture Notes in Computer Science*, pages 24–35. Springer-Verlag, 1994.

[2] J. A. Bondy and U. S. R. Murty. *Graph Theory with Applications.* North-Holland, Amsterdam, 1976.

[3] G. Brightwell and W. T. Trotter. The order dimension of planar maps. Technical Report LSE-MPS-37, Dept. of Statistical and Mathematical Sciences, London School of Economics and Political Science, 1992.

[4] G. Brightwell and W. T. Trotter. The order dimension of convex polytopes. *SIAM J. Discrete Math.*, 6(2):230–245, 1993.

[5] J. Cheriyan and S. N. Maheshwari. Finding nonseparating induced cycles and independent spanning trees in 3-connected graphs. *J. Algorithms*, 9:507–537, 1988.

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

[7] M. Chrobak, M. T. Goodrich, and R. Tamassia. Convex drawings of graphs in two and three dimensions. In *Proc. 12th Annu. ACM Sympos. Comput. Geom.*, pages 319–328, 1996.

[8] M. Chrobak and S. Nakano. Minimum-width grid drawings of plane graphs. In R. Tamassia and I. G. Tollis, editors, *Graph Drawing (Proc. GD '94)*, volume 894 of *Lecture Notes in Computer Science*, pages 104–110. Springer-Verlag, 1995.

[9] R. F. Cohen, G. Di Battista, A. Kanevsky, and R. Tamassia. Reinventing the wheel: an optimal data structure for connectivity queries. In *Proc. 25th Annu. ACM Sympos. Theory Comput.*, pages 194–200, 1993.

[10] H. de Fraysseix, P. O. de Mendez, and P. Rosenstiehl. Bipolar orientations revisited. *Discrete Appl. Math.*, 56:157–179, 1995.

[11] H. de Fraysseix, J. Pach, and R. Pollack. How to draw a planar graph on a grid. *Combinatorica*, 10:41–51, 1990.

[12] H. de Fraysseix and P. Rosenstiehl. Structures combinatoires pour des traces automatiques de reseaux. In *Proc. 3rd European Conf. on CAD/CAM and Computer Graphics*, pages 332–337. Hermes, 1984.

[13] G. Di Battista, P. Eades, R. Tamassia, and I. G. Tollis. Algorithms for drawing graphs: An annotated bibliography. *Comput. Geom. Theory Appl.*, 4:235–282, 1994.

[14] G. Di Battista and R. Tamassia. On-line maintenance of triconnected components with SPQR-trees. *Algorithmica*, 15(3):302–318, 1996.

[15] G. Di Battista and R. Tamassia. On-line planarity testing. *SIAM J. Comput.*, to appear. Preprint: Technical Report CS-92-39, Comput. Sci. Dept., Brown Univ., 1992.

[16] J. Edmonds. Edge-disjoint branchings. In R. Rustin, editor, *Combinatorial Algorithms*, pages 91–96. Algorithmics Press, New York, 1972.

[17] S. Even and R. E. Tarjan. Network flow and testing graph connectivity. *SIAM J. Comput.*, 4:507–518, 1975.

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

[19] I. Fary. On straight lines representation of planar graphs. *Acta Sci. Math. Szeged*, 11:229–233, 1948.

[20] M. Fürer, X. He, M.-Y. Kao, and B. Raghavachari. $O(n \log n)$-work parallel algorithm for straight-line grid embedding of planar graphs. In *Proc. ACM Sympos. Parallel Algorithms Architect.*, pages 100–109, 1992.

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

[22] X. He and M.-Y. Kao. Regular edge labelings and drawings of planar graphs. In R. Tamassia and I. G. Tollis, editors, *Graph Drawing (Proc. GD '94)*, volume 894 of *Lecture Notes in Computer Science*, pages 96–103. Springer-Verlag, 1995.

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

[24] A. Huck. Independent trees in graphs. *Graphs Combin.*, 10(1):29–45, 1994.

[25] A. Huck. Disproof of a conjecture about independent branchings in k-connected directed graphs. *J. Graph Theory*, 20(2):235–239, 1995.

[26] A. Huck. Independent trees in planar graphs, (submitted to *Graphs Combin.*).

[27] A. Itai and M. Rodeh. The multi-tree approach to reliability in distributed networks. *Inform. and Comput.*, 79(1):43–59, 1988.

[28] A. Kanevsky, R. Tamassia, G. Di Battista, and J. Chen. On-line maintenance of the four-connected components of a graph. In *Proc. Annu. IEEE Sympos. Found. Comput. Sci.*, pages 793–801, 1991.

[29] G. Kant. Drawing planar graphs using the canonical ordering. *Algorithmica* (Special Issue on Graph Drawing, G. Di Battista and R. Tamassia, editors), 16(1):4–32, 1996.

[30] G. Kant and X. He. Two algorithms for finding rectangular duals of planar graphs. In *Graph-Theoretic Concepts in Computer Science (Proc. WG '93)*, volume 790 of *Lecture Notes in Computer Science*, pages 396–410. Springer-Verlag, 1993.

[31] D. Kelly and I. Rival. Planar lattices. *Canad. J. Math.*, 27(3):636–665, 1975.

[32] S. Khuller and B. Schieber. On independent spanning trees. *Inform. Process. Lett.*, 42(6):321–323, 1992.

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

[34] H. Nagamochi and T. Ibaraki. A linear-time algorithm for finding a sparse k-connected spanning subgraph of a k-connected graph. *Algorithmica*, 7:583–596, 1992.

[35] F. P. Preparata and R. Tamassia. Fully dynamic point location in a monotone subdivision. *SIAM J. Comput.*, 18:811–830, 1989.

[36] P. Rosenstiehl and R. E. Tarjan. Rectilinear planar layouts and bipolar orientations of planar graphs. *Discrete Comput. Geom.*, 1(4):343–353, 1986.

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

[38] W. Schnyder. Embedding planar graphs on the grid. In *Proc. 1st ACM-SIAM Sympos. Discrete Algorithms*, pages 138–148, 1990.

[39] W. Schnyder and W. Trotter. Convex embeddings of 3-connected plane graphs. *Abstracts AMS*, 13(5):502, 1992.

[40] S. K. Stein. Convex maps. *Proc. Amer. Math. Soc.*, 2:464–466, 1951.

[41] E. Steinitz and H. Rademacher. *Vorlesungen über die Theorie der Polyeder.* Julius Springer, Berlin, Germany, 1934.

[42] R. Tamassia. A dynamic data structure for planar graph embedding. In T. Lepisto and A. Salomaa, editors, *Automata, Languages and Programming (Proc. ICALP '88)*, volume 317 of *Lecture Notes in Computer Science*, pages 576–590. Springer-Verlag, 1988.

[43] R. Tamassia and I. G. Tollis. A unified approach to visibility representations of planar graphs. *Discrete Comput. Geom.*, 1(4):321–341, 1986.

[44] R. Tamassia and I. G. Tollis. Planar grid embedding in linear time. *IEEE Trans. on Circuits and Systems*, CAS-36(9):1230–1234, 1989.

[45] R. Tamassia and J. S. Vitter. Parallel transitive closure and point location in planar structures. *SIAM J. Comput.*, 20(4):708–725, 1991.

[46] P. Tong and E. Lawler. A faster algorithm for finding edge-disjoint branchings. *Inform. Process. Lett.*, 17(2):73–76, 1983.

[47] W. T. Tutte. Convex representations of graphs. *Proc. London Math. Soc.*, 10:304–320, 1960.

[48] W. T. Tutte. How to draw a graph. *Proc. London Math. Soc.*, 13:743–768, 1963.

[49] D. Wagner and K. Weihe. A linear-time algorithm for edge-disjoint paths in planar graphs. In *Algorithms (Proc. ESA '93)*, volume 726 of *Lecture Notes in Computer Science*, pages 384–395. Springer-Verlag, 1993.

[50] K. Wagner. Bemerkungen zum vierfarbenproblem. *Jahresbericht der Deutschen Mathematiker-Vereinigung*, 46:26–32, 1936.

[51] J. Westbrook and R. E. Tarjan. Maintaining bridge-connected and biconnected components on-line. *Algorithmica*, 7:433–464, 1992.

[52] R. W. Whitty. Vertex-disjoint paths and egde-disjoint branchings in directed graphs. *J. Graph Theory*, 11(3):349–358, 1987.

[53] S. K. Wismath. Characterizing bar line-of-sight graphs. In *Proc. 1st Annu. ACM Sympos. Comput. Geom.*, pages 147–152, 1985.

[54] A. Zehavi and A. Itai. Three tree-paths. *J. Graph Theory*, 13(2):175–188, 1989.

A Results on k-Path and k-Connectivity Queries

Table 1 summarizes the previous and new results on methods for k-path and k-connectivity queries.

graph	k	space	preprocessing	k-conn.	k-path	references
			Previous Results			
general	any	$O(n+m)$	—	$O(m\sqrt{n})$	$O(m\sqrt{n})$	[17]
general	fixed	$O(n+m)$	—	$O(n+m)$	$O(n+m)$	[17]
planar	any	$O(n)$	—	$O(n)$	$O(n)$	[49]
$(k-1)$-conn.	fixed $k>4$	$O(n)$	$O(n^4 m)$	$O(1)$	$O(n)$	[9]
general	$k \leq 3$	$O(n)$	$O(n+m)$	$O(1)$	$O(n)$	[14, 42, 51]
general	$k=4$	$O(n)$	$O(n\alpha(m,n)+m)$	$O(1)$	$O(n)$	[28]
general	$k=1$	$O(n)$	$O(n+m)$	$O(1)$	$O(\ell)$	—
			New Results			
general	$k=2$	$O(n)$	$O(n)$	$O(1)$	$O(\ell)$	§ 2.2
planar	$k=3$	$O(n)$	$O(n)$	$O(1)$	$O(\ell)$	§ 4
general	$k=3$	$O(n)$	$O(n^2)$	$O(1)$	$O(\ell)$	§ 5
planar	$k \leq 3$	$O(n)$	$O(n)$	$O(1)$	$O(\ell)$	§ 6
general	$k \leq 3$	$O(n)$	$O(n^2)$	$O(1)$	$O(\ell)$	§ 6

Table 1: Summary of results on methods for k-path and k-connectivity queries.

B The SPQR-Tree

In this appendix, the *SPQR-tree* presented in [15, 14] is described. Let G be a biconnected graph. A *split pair* of G is either a pair of adjacent vertices or a separation pair. In the former case the split pair is said *trivial*, in the latter *non-trivial*. 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 said *trivial*, in the latter *non-trivial*. Note that each vertex of G distinct from u and v belongs to exactly one non-trivial 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 example of Fig. 9.a, $\{v_1,v_5\}$ is a trivial split pair, $\{v_9,v_{12}\}$ is a non-trivial 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 non-trivial split component, $\{v_1,v_{15}\}$ is a maximal split pair with respect to $\{v_3,v_7\}$, while $\{v_1,v_{12}\}$ is not.

Let $e=(s,t)$ be an edge of G, called *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 by $skeleton(\mu)$. Also, it is associated with an edge of the skeleton of the parent ν of μ, called the *virtual edge* of μ in $skeleton(\nu)$. Tree T is recursively defined as follows.

Trivial Case: If G consists of exactly two parallel 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 $G_1,\ldots,G_k$ ($k\geq 3$), the root of T is a P-node μ. Graph $skeleton(\mu)$ consists of k parallel edges between s and t, denoted $e_1,\ldots,e_k$, with $e_1=e$.

Series Case: If the split pair $\{s, t\}$ has exactly two split components, one of them is the reference edge e, and we denote with G' the other split component. If G' has cut-vertices $c_1, \ldots, c_{k-1}$ ($k \geq 2$) that partition G into its blocks $G_1, \ldots, G_k$, in this order from s to t, the root of T is an S-node μ. Graph $skeleton(\mu)$ is the cycle $e_0, e_1, \ldots, e_k$, where $e_0 = e$, $c_0 = s$, $c_k = t$, and e_i connects c_{i-1} with c_i ($i = 1, \ldots, k$).

Rigid Case: If none of the cases above applies, let $\{s_1, t_1\}, \ldots, \{s_k, t_k\}$ be the maximal split pairs of G with respect to $\{s, t\}$ ($k \geq 1$), and for $i = 1, \ldots, k$, let G_i be the union of all the split components of $\{s_i, t_i\}$ except the one containing the reference edge e. The root of T is an R-node μ. Graph $skeleton(\mu)$ is obtained from G by replacing each subgraph G_i with the edge e_i between s_i and t_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 $G_i \cup e_i$ with respect to reference edge e_i ($i = 1, \ldots, k$). 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 adding another Q-node, representing the reference edge e, and making it the parent of μ so that it becomes the root. An example of SPQR-tree is shown in Fig. 9.c, where 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_i of $skeleton(\mu)$. A virtual edge is said *trivial* if the corresponding node μ_i is a Q-node, *non-trivial* otherwise. The endpoints of e_i are called the *poles* of μ_i. Graph G_i is called the *pertinent graph* of node μ_i, and the *expansion graph* of edge e_i.

In the example of Fig. 9, the non-trivial virtual edges are represented by dotted lines, the trivial virtual edges are represented by solid lines.

Let μ be a node of T. We have:

- if μ is an R-node, then $skeleton(\mu)$ is a triconnected graph;
- if μ is an S-node, then $skeleton(\mu)$ is a cycle;
- if μ is a P-node, then $skeleton(\mu)$ is a triconnected multigraph consisting of a bundle of multiple edges;
- if μ is a Q-node, then $skeleton(\mu)$ is a biconnected 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 classical decomposition of biconnected graphs into triconnected components [23]. Namely, the triconnected components of a biconnected graph G are in one-to-one correspondence with the internal nodes of the SPQR-tree: the R-nodes correspond to triconnected graphs, the S-nodes to polygons, and the P-nodes to bonds.

Let v be a vertex of G. The *allocation nodes* of v are the nodes of T whose skeleton contains v. The least common ancestor μ of the allocation nodes of v is itself an allocation node of v and is called the *proper* allocation node of v, denoted $\mu = proper(v)$. If $v = s$ or $v = t$ (the endpoints of the reference edge) we conventionally define $proper(v)$ as the unique child of the root of T (recall that the root of T is the Q-node of the reference edge). If $v \neq s, t$, node $\mu = proper(v)$ is either an R-node or an S-node; also, μ is the only allocation node of v such that v is not a pole of μ. The set of vertices v with proper allocation node μ is denoted $properset(\mu)$. If μ is a (proper) allocation node of v, we say that v is (properly) allocated at μ.

The SPQR-tree T of a graph with n vertices and m edges has m Q-nodes and $O(n)$ S-, P-, and R-nodes. The total number of vertices of the skeletons stored at the nodes of T is $O(n)$. It can be constructed in $O(n+m)$ time using a variation of the algorithm given in [23].

C The BC-Tree

In this appendix, the *BC-tree* presented in [15, 14] is described. Let G be a connected graph with n vertices. The *BC-tree* T of G has a B-node for each block (biconnected component) of G, and a C-node for each cutvertex of G. Edges in T connect each B-node μ to the C-nodes associated with the cutvertices in the block of μ. The BC-tree is rooted at an arbitrary B-node. Also the B-node of each nontrivial block B stores the SPQR-tree of B. Observe that the number of blocks of G is $O(n)$, and the total number of vertices in the blocks of G is $O(n)$ as well.

The BC-tree is a variation of the data structures for maintaining biconnected components described in [42, 51]. The main difference is that an SPQR-tree is attached at each B-node.

If vertex v is a cutvertex, $bcproper(v)$ denotes the C-node associated with v. Otherwise, $bcproper(v)$ denotes the B-node of the unique block containing v. It is easy to see that, knowing $\mu_1 = bcproper(v_1)$ and $\mu_2 = bcproper(v_2)$, we can determine in $O(1)$ time whether v_1 and v_2 are in the same block of G [42]: namely the block associated with node μ contains vertices v_1 and v_2 if and only if the undirected path of T between μ_1 and μ_2 contains μ but no other B-node.

The BC-tree of a graph with n vertices and m edges can be constructed in $O(n+m)$ time.