Output-Sensitive Reporting of Disjoint Paths (Extended Abstract)*

Giuseppe Di Battista
Dipartimento di Discipline Scientifiche
Sezione Informatica
Terza Università degli Studi di Roma
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 static data structure for G that uses $O(n)$ space and executes k-path queries in output-sensitive $O(\ell)$ time. We also give a semi-dynamic version of the data structure that supports a sequence of intermixed queries and insertions of vertices and edges in a planar graph, with worst-case query time $O(\ell)$ and amortized insertion time $O(\log n)$. Our results make use of a new combinatorial structure for triconnected planar graphs that plays the same role as bipolar (st) orientations for biconnected planar graphs. This combinatorial structure also yields an alternative construction of convex 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 Italian National Research Council, by the Progetto Bilaterale 94.23.CT07 Italia-USA of the Italian National Research Council, 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. Recently, there has been increasing interest in the development of dynamic algorithms for connectivity problems, which have often become the testbed for the development of general dynamization techniques (see, e.g., [14, 12, 16, 15, 20, 21, 24, 27, 28, 29, 33, 37, 38, 47]).

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 preprocessing or dynamically maintaining a 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 $f(n) = O(1)$.

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 [46].

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, 47] for $k = 2$, [12, 13] for $k = 3$, [29] for $k = 4$, and [7] for $k > 4$).

For $k \leq 4$, several dynamic data structures have been devised for performing k-connectivity queries in a dynamic environment, where the queries are intermixed with insertions and/or deletions of vertices and edges [12, 16, 15, 20, 22, 24, 29, 33, 37, 38, 47]. E.g., for $k \leq 3$, there are $O(n)$-space data structures that support a sequence of t intermixed insertions and k-connectivity queries in a general graph in $O(t\,\alpha(t, n))$ time [33], and a sequence of t intermixed insertions, deletions and k-connectivity queries in a planar graph in $O(\sqrt{n})$ time [16]. However, none of the previously known dynamic data structures is able to perform output-sensitive k-path queries.

1.2 Previous Results on Orientations and Orderings of Planar Graphs

Orientations and orderings are powerful combinatorial structures that have been successfully applied to solving problems on planar graphs. 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, 34], and were later used for a variety of topological and geometric graph problems, such as embedding (see, e.g., [5, 11, 42]), visibility (see, e.g., [39, 43, 48]), drawing (see, e.g., [1, 10, 44]), point location (see, e.g., [36, 45]), and floorplanning (see, e.g., [31]). One of the notable properties of planar bipolar orientations is that they induce a 2-dimensional lattice [32] on the vertices of the graph. See [8] for a recent comprehensive study of bipolar orientations.

Canonical orderings were first defined by de Fraysseix, Pach and Pollack [9] for maximal planar graphs and later extended by Kant [30] 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., [6, 9, 30]).

Schnyder [40] 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 [41]. The construction of realizers of maximal planar graphs can also be efficiently parallelized [19]. Brightwell and Trotter [2, 3] define normal families of paths for triconnected planar graphs, which 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 planar maps and convex polytopes.

Graph drawing methods based on orientations, numberings and realizers are surveyed in [23].

Cheriyan and Maheshwari [4], and Zehavi and Itai [49] proved the existence of three independent spanning trees in triconnected graphs. While the proof of Zehavi and Itai is existential, 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.

1.3 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 [40] 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 [2, 3] for G in $O(n)$ time. Brightwell and Trotter [2, 3] 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 [40], and gives an alternative proof of Kant's result [30].
- Based on realizers, we show how to construct a linear-space data structure that supports output-sensitive 3-path queries in 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 [4] and by defining new topological properties of general graphs, 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.
- We show how to incrementally update the realizers of the triconnected components of a planar graph G in a semi-dynamic environment where insertions of vertices and edges that preserve planarity are performed. Our semi-dynamic data structure for G has the following performance:
  - The space requirement is $O(n)$, where n is the current number of vertices of G.
  - A k-path query, with $k \leq 3$, takes (worst-case) time $O(\ell)$, where ℓ is the total size of the paths reported by the query.
  - A sequence of t intermixed vertex insertions, edge insertions and k-path queries ($k \leq 3$), starting from the empty graph, takes total time $O(n \log n + t + L)$, where n is the number of vertex insertions, and L denotes the total size of the paths reported by the queries.

The contributions of this paper can be summarized as follows:

- We have defined, analyzed, and shown how to efficiently compute realizers of planar triconnected 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 shown how to efficiently dynamize the above data structure when queries are intermixed with insertions of vertices and edges, with logarithmic-time overhead per query or update operation.

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 static data structure and the output-sensitive algorithm for 3-path queries in planar graphs are given in Section 4. The extension to general graphs is in Section 5. In Section 6 we show how to dynamize our data structure. Finally, we mention open problems motivated by our work in Section 7.

Due to space limitations, many technical details are omitted in this extended abstract and the proofs are in Appendix A.

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

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 edge or one vertex distinct from their endpoints. The set of vertices and edges shared by two crossing paths is called a *crossing*.

We use the term *plane graph* to denote a planar graph with a given planar embedding and external face. 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. We recall that a triconnected planar graph has a unique planar embedding.

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 a graph with an edge (s,t). A bipolar orientation (also called st-orientation) [8, 34] of G with respect to an edge (s,t) is an acyclic digraph D obtained by orienting the edges of G such that 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].

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 [25, 26]. Tree T_s is obtained by selecting for every vertex $v \neq s$ an edge of G incident on v that is directed towards v in the orientation. Tree T_t is similarly obtained by selecting an outgoing edge for every vertex distinct from t. Clearly, for every vertex v of G, the path $p_s(v)$ of T_s between v and s and the path $p_t(v)$ of 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 the edge (s,t) and the four paths $p_s(u)$, $p_s(v)$, $p_t(u)$, and $p_t(v)$, contains two disjoint paths between u and v.*

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 time $O(\ell)$, where ℓ is the size of the reported paths.*

In a dynamic environment, where a sequence of intermixed 2-path queries and insertions of vertices and edges is performed on a graph G (of arbitrary connectivity), we augment a data structure for on-line 2-connectivity queries [42, 47] in G by storing with each block (biconnected component) the data structure of Theorem 1.

Theorem 2 *There exists a data structure for general graphs that supports a sequence of t operations, each a 2-path query or the insertion of a vertex or an edge, starting from the empty graph, in time $O(n \log n + t + L)$, where n is the total number of vertices inserted and L is the total size of the paths reported by the queries performed. At any time, the space requirement is linear in the current number of vertices, and a 2-path query is answered in time $O(\ell)$ (worst-case), where ℓ is the size of the reported paths.*

2.3 Canonical Orderings

In this section we recall the definition of canonical orderings of triconnected plane graphs, as given by Kant [30], and prove some preliminary results.

Let G be a triconnected plane graph, and u_0, u_1, u_2 three consecutive external vertices of G. A *canonical ordering* of G (see Fig. 1) is an ordering $v_1, \cdots, v_n$ of the vertices of G that can be partitioned into subsequences $V_1, \cdots, V_H$, where $V_i = \{v_{s_i}, \cdots, v_{s_{i+1}-1}\}$ $(1 = s_1 < s_2 < \cdots < s_H \leq n)$, 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 $2 \leq k \leq H-1$ exactly 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, each v_i, $s_k \leq i \leq s_k + d_k$, has at least one neighbor in $G - G_k$, and each v_i, $s_k < i < s_k + d_k$, has 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 $V_H = \{v_n\}$.

Lemma 2 [30] *Each triconnected plane graph has a canonical ordering that 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.
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 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:
   (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 *ingoing edges* that belong to the same spanning tree appear consecutively between the outgoing edges of the two remaining spanning trees (the first and last ingoing edges are possibly coincident with the outgoing edges).

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.d). 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_6, v_{11}) is 1-colored, while edge (v_7, v_{12}) is 2-colored.

Lemma 3 *Every triconnected plane graph G admits a realizer, which can be computed in linear time.*

3.2 Properties

In this section, we consider a triconnected plane graph G equipped with a realizer T_b, T_g, T_r. We use the notation v_b, v_g, and v_r instead of v_1, v_2, and v_n, 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 v_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 v_g and the *red path* $p_r(v)$ as the path of G along T_r with endpoints v and v_r. In the following, 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 v_g and v_r and not containing v_b is denoted by $ext(v_g, v_r)$; the subpath of the external face with endpoints v_r and v_b and not containing v_g is denoted by $ext(v_r, v_b)$; the subpath of the external face with endpoints v_b and v_g and not containing v_r is denoted by $ext(v_b, v_g)$. 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 sake.

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 $ext(v_g, v_r)$. 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 $ext(v_r, v_b)$ and the *red region* $R_r(v)$ is the subgraph of G with external face formed by $p_b(v)$, $p_g(v)$ and $ext(v_b, v_g)$. See Fig. 3.

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. In the rest of the paper, set $\{b, g, r\}$ will be considered accordingly ordered.

The following lemmas give fundamental properties of a triconnected plane graph G equipped with a realizer T_b, T_g, T_r.

Lemma 4 *Let v be a vertex of G and i, j, k be three consecutive colors in the circularly ordered set $\{b, g, r\}$. Let x 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 i-colored (k-colored) ingoing edge of x different from (y, x) is on the left (right) of $p_j(v)$.*

Lemma 5 *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 $p_i(u, v) = p_j(v, u)$.*

Lemma 6 *The following properties hold for G and its realizer:*

1. *for each vertex v of G, $p_b(v)$, $p_g(v)$, and $p_r(v)$ have only vertex v in common;*
2. *$p_r(v_g) = p_g(v_r) = ext(v_g, v_r)$; $p_b(v_r) = p_r(v_b) = ext(v_r, v_b)$; $p_g(v_b) = p_b(v_g) = ext(v_b, v_g)$;*
3. *for each pair of vertices u and v of G, two cases are possible:*
   (a) *there are exactly two colors $i, j \in \{b, g, r\}$, $i \neq j$, such that $p_i(v)$ and $p_j(u)$ cross (see Fig. 4.a); three subcases are possible:*
      i. *$u \notin p_j(u)$ and $v \notin p_i(v)$;*
      ii. *either $u \in p_j(u)$ or $v \in p_i(v)$;*
      iii. *$u \in p_j(u)$ and $v \in p_i(v)$;*
   (b) *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 $p_k(u) \subset p_k(v)$ or $p_k(v) \subset p_k(u)$ (see Fig. 4.b);*
4. *for each pair of vertices u and v of G and each color $i \in \{b, g, r\}$, if $u \in R_i(v)$ then $R_i(u) \subseteq R_i(v)$.*

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 $v \in R_i(u)$.*

The properties of a *normal family of paths* [3] for a plane graph and three distinguished external vertices, are similar to the properties of the paths of Lemma 6. Brightwell and Trotter [3] proved that each triconnected plane graph has a normal family of paths for any three external vertices. Using the terminology of [3], we can say that Lemma 6 shows that the set $\{p_i(v) | i \in \{b, g, r\},\ v \in V\}$ is a normal family of paths for the three vertices v_b, v_g, and v_r.

3.3 Applications to Graph Drawing

The realizers we have defined for triconnected graphs naturally extend those defined by Schnyder [40, 41] for maximal planar graphs, where each internal edge has exactly one color. In particular, our realizers support the construction of *barycentric drawings* [40, 41], which yields an alternative proof of the following theorem:

Theorem 3 [30] *Let G be triconnected plane graph with n vertices. A convex grid drawing of G with $O(n^2)$ area can be computed in $O(n)$ time.*

4 Planar 3-Path Queries in a Static Environment

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 and straightforward to implement. This allows us to provide a detailed pseudo-code description.

4.1 Preprocessing

In order to simplify the algorithm, we use a *common-sink realizer* for triconnected planar graph G, i.e. a realizer in which a common vertex of degree three is chosen as v_b, v_g and v_r. If G has no vertex of degree three, we first apply the algorithm of Nagamochi and Ibaraki [35] to obtain a sparse triconnected spanning subgraph G' of G; G' is guaranteed to have a vertex x of degree three (see Lemma 2.6 of [35]). Otherwise we let $G' = G$ Then, a realizer of G' is computed, as shown in the proof of Lemma 3, with $v_1 = x$; a realizer of G' is also a realizer of G. Finally, a common-sink realizer of G' is obtained in the following way: let (v_g, y) be the edge following (v_g, v_b) in the clockwise order around v_g, and let (v_r, z) be the edge preceding (v_r, v_b) in the clockwise order around v_r; (v_g, v_b) is colored green, (v_g, y) is colored blue and green; (v_r, v_b) is colored red; (v_r, z) is colored green and red; v_g and v_r are identified with v_b.

4.2 Query Algorithm

Let G be a triconnected plane graph equipped with a common-sink realizer. To answer a 3-path query for vertices u and v, we assemble three paths between u and v by suitably following 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. By Lemma 6, for each pair of vertices u and v of a triconnected plane graph G two cases are possible.

- If Case 3a of Lemma 6 applies, then there exist two paths $p_i(v)$ and $p_j(u)$ that cross (see Fig. 5.a–5.c; the portions of paths that are used to compose the three disjoint paths are thicker). These two paths are exploited to determine a first path with endpoints u and v. A second path is determined using $p_i(v)$, $p_j(u)$ and part of the external face of G. The third path is the one along T_k, $k \neq i, j$. Notice that, if $i, j \neq b$, $p_i(v)$ and $p_j(u)$ may have an external crossing.
- If Case 3b of Lemma 6 applies, the three paths are the non-common part of $p_i(u)$ and $p_i(v)$, and the paths along T_j, $j \neq i$ and along T_k, $k \neq i, j$, respectively (see Fig. 5.d–5.f).

A detailed description of the 3-path query algorithm is given below, where the following notation is used. The concatenation of two paths $p_i(v, u)$ and $p_j(w, u)$, $i, j \in \{b, g, r\}$, $i \neq j$, having only vertex u in common is denoted by $p_i(v, u) + p_j(w, u)$. Furthermore, 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 sake; in the figures we will use sv_{ij} instead of $stopvertex_{ij}$.

Algorithm ThreePaths
Input: a triconnected plane graph G, equipped with a common-sink realizer
a pair of vertices u and v
Output: three disjoint paths p_1, p_2, and p_3 with endpoints u and v

```
begin
  if there are exactly two colors i, j ∈ {b, g, r}, i ≠ j, such that p_i(v) and p_j(u) cross then
    case u, v of
{a}   p_b(v) crosses p_g(u)                          {internal blue-green crossing}    {see Fig. 5.a}
      (or analogously p_b(u) crosses p_g(v))
        compute lca_r and stopvertex_bg;             {lockstep}
        p1 := p_r(u, lca_r) + p_r(v, lca_r);   p2 := p_b(u, v_b) + p_g(v, v_g);
        p3 := p_g(u, stopvertex_bg) + p_b(v, stopvertex_bg)
{b}   p_r(v) crosses p_b(u)                          {internal red-blue crossing}      {see Fig. 5.b}
      (or analogously p_r(u) crosses p_b(v))
        compute lca_g and stopvertex_rb;             {lockstep}
        p1 := p_g(u, lca_g) + p_g(v, lca_g);   p2 := p_b(u, v_b) + p_r(v, v_r);
        p3 := p_r(u, stopvertex_rb) + p_b(v, stopvertex_rb)
{c}   p_g(v) crosses p_r(u)                          {green-red crossing}              {see Fig. 5.c}
      (or analogously p_g(u) crosses p_r(v))
        compute lca_b and stopvertex_gr;             {lockstep}
        p1 := p_b(u, lca_b) + p_b(v, lca_b);
        p2 := p_g(u, v_g) + p_r(v, v_r);
        p3 := p_r(u, stopvertex_gr) + p_g(v, stopvertex_gr)
    endcase
  else
  {there is exactly one color k ∈ {b, g, r}, such that either p_k(u) ⊂ p_k(v) or p_k(v) ⊂ p_k(u)}
    case u, v of
{d}   p_b(u) ⊂ p_b(v) (or analogously p_b(v) ⊂ p_b(u))                                 {see Fig. 5.d}
        compute lca_g and lca_r;                     {lockstep}
        p1 := p_g(u, lca_g) + p_g(v, lca_g);   p2 := p_r(u, lca_r) + p_r(v, lca_r);   p3 := p_b(u, v)
{e}   p_g(u) ⊂ p_g(v) (or analogously p_g(v) ⊂ p_g(u))                                 {see Fig. 5.e}
        compute lca_r and lca_b;                     {lockstep}
        p1 := p_b(u, lca_b) + p_b(v, lca_b);   p2 := p_r(u, lca_r) + p_r(v, lca_r);   p3 := p_g(u, v)
{f}   p_r(u) ⊂ p_r(v) (or analogously p_r(v) ⊂ p_r(u))                                 {see Fig. 5.f}
        compute lca_b and lca_g;                     {lockstep}
        p1 := p_b(u, lca_b) + p_b(v, lca_b);   p2 := p_g(u, lca_g) + p_g(v, lca_g);   p3 := p_r(u, v)
    endcase
  endif
end.
```

Theorem 4 *Algorithm ThreePaths correctly returns three vertex-disjoint paths between two given vertices of a triconnected plane graph G equipped with a common-sink realizer.*

4.3 Data Structure and Complexity

In this section we present a data structure for performing 3-path queries in a triconnected planar graph G with n vertices. By Lemma 3 and by Theorem 2.1 of [35], we assume that G has been embedded and a common-sink realizer T_b, T_g, T_r of G has been constructed, which can be done in $O(n)$ time.

A first version of the data structure consists of trees T_b, T_g, and T_r, implemented with parent pointers. It is easy to see that the above data structure can be constructed in $O(n)$ time.

Using this data structure, we perform Algorithm ThreePaths by activating 6 concurrent computational threads, corresponding to the various cases of the algorithm. By Theorem 4, one of the threads will successfully answer the query. Hence, we add to each thread a postprocessing step that checks whether the final result is indeed an answer to the query. In a sequential machine, the threads are executed in "lockstep" mode, by performing in turn a constant amount of computation for each of them. As soon as one thread successfully completes the computation, the remaining threads are terminated.

Let ℓ be the size of the paths returned by Algorithm ThreePaths. Within a thread, once $stopvertex_{ij}$ and lca_k, $i, j, k \in \{b, g, r\}$ are known, reporting p_1, p_2, and p_3 can be done in $O(\ell)$ time by simply traversing trees T_i, T_j, T_k. Thus, it remains to prove that the computation of $stopvertex_{ij}$ or lca_k can be carried out in $O(\ell)$ time. This is done by performing in "lockstep" mode a scan of $p_i(v)$ and $p_j(u)$, or $p_k(u)$ and $p_k(v)$. The $O(\ell)$ time bound is guaranteed by the "lockstep" technique and by the fact that, in all the cases in which $stopvertex_{ij}$ or lca_k are computed, at least one of the paths explored to compute them is reused for constructing p_1, p_2 or p_3.

Theorem 5 *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 time $O(\ell)$, where ℓ is the size of the reported paths.*

4.4 A Practical and Optimal Data Structure

While theoretically optimal, the practical performance of the above data structure is negatively affected by the overhead caused by running 6 concurrent threads, only one of which will be successful. In order to make Algorithm ThreePaths more practical, we add to the above data structure a component for determining which case of the the algorithm applies.

For this purpose, we define for each tree T_i, $i \in \{b, g, r\}$, two binary relations $\equiv_i$ and $\prec_i$ on the vertex set of G. For a pair of vertices $\{u, v\}$, such relations determine the relative positions of u and v in the embedding of G. Namely, $u \equiv_i v$ if the lowest common ancestor lca_i of u and v in T_i is either u or v; $u \prec_i v$ if, with respect to lca_i, the subtree containing u is on the left of the subtree containing v.

Lemma 7 *It is not possible that $u \equiv_b v$ and $u \equiv_g v$ and $u \equiv_r v$. It is not possible that $u \prec_b v$ and $u \prec_g v$ and $u \prec_r v$.*

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

Lemma 8 *Let u and v be two vertices of G, and i, j, k be three consecutive colors in the circularly ordered set $\{b, g, r\}$. Paths $p_i(v)$ and $p_j(u)$ cross if and only if one of the following cases applies:*

1. *$v \prec_i u$ and $v \prec_j u$ and $u \prec_k v$;*
2. *let w be the parent of v along T_i; $u \equiv_j v \equiv_j w$ and $v \prec_i u$ and $u \prec_k v$;*
3. *let w be the parent of u along T_j; $u \equiv_i v \equiv_i w$ and $v \prec_j u$ and $u \prec_k v$;*
4. *$u \equiv_i v$ and $u \equiv_j v$ and $u \prec_k v$.*

Lemma 9 *Let u and v be two vertices of G. Let i, j, k be three consecutive colors in the circularly ordered set $\{b, g, r\}$, and let x and y be the parents of v along T_j and T_k, respectively. Path $p_i(u)$ is a proper subpath of $p_i(v)$, and $p_i(v)$ cross neither $p_j(u)$ nor $p_k(u)$ if and only if $u \equiv_i v$ and $v \prec_j u$ and $u \prec_k v$ and $x \prec_i v$ and $v \prec_i x$.*

To determine which of Cases a–f of Algorithm ThreePaths applies, we have to evaluate whether one crossing exists or not; in the positive case we have to determine which colors are involved. This is done by exploiting Lemmas 8 and 9.

We implement relations $\prec_i$ and $\equiv_i$, $i \in \{b, g, r\}$ by associating to each vertex v of T_i two positive integers: $min_i(v)$ and $max_i(v)$. We consider the leaves from left to right with respect to the embedding of G: for the leftmost leaf, we set $min_i(v) = max_i(v) = 1$; for the second leftmost leaf, we set $min_i(v) = max_i(v) = 2$, etc. For any other vertex of T_i, we set $min_i(v)$ equal to the value of its leftmost descendant leaf, and $max_i(v)$ equal to the value of its rightmost descendant leaf. The two binary relations are easily tested in the following way. For each pair of vertices u and v of G:

- $u \prec_i v$ if and only if $max_i(u) < min_i(v)$
- $u \equiv_i v$ if and only if $max_i(u) \geq min_i(v)$ and $min_i(u) \leq max_i(v)$

Thus, determining which case of Algorithm ThreePaths applies can be done in $O(1)$ time. Clearly, the computation of $min_i(x)$ and $max_i(x)$, $i \in \{b, g, r\}$, $1 \le x \le n$, can be done in $O(n)$ time. This completes the description of our theoretically optimal and practically efficient data structure for 3-path queries.

4.5 General Planar Graphs

The result of Theorem 5 can be extended to general (i.e., of arbitrary connectivity) planar graphs by suitably augmenting the static data structure of [12, 13] for 3-connectivity queries. This is essentially done by storing at each triconnected component the data structure of Theorem 5. Also, we can combine this data structure with the one of Theorem 1 and the known methods for 1-path queries, and obtain the following result:

Theorem 6 *Let G be a planar graph with n vertices. There exists an $O(n)$-space data structure for G that can be constructed in $O(n)$ time and supports 1- 2- and 3-path queries in time $O(\ell)$, where ℓ is the size of the reported paths.*

5 General 3-Path Queries in a Static Environment

In this section we extend the results on triconnected planar graphs of Section 4 to general triconnected graphs. Let G be a triconnected graph with n vertices and m edges:

- three spanning trees satisfying Properties 1–4 of the realizers can be constructed using the nonseparating ear decomposition of Cheriyan and Maheshwari [4]; by applying the algorithm on a sparse triconnected spanning subgraph of G [4, 35], the time complexity of the problem can be reduced from $O(nm)$ to $O(n^2)$;
- the adjacency lists of the vertices of G are sorted according to the numbering of the ears, so that Properties 5a and 5b of the realizers are satisfied;
- visiting the sorted spanning trees T_i, $i \in \{b, g, r\}$, the binary relations $\equiv_i$ and $\prec_i$ on the vertex set of G are computed;
- for each vertex v of G, $R_b(v)$ is defined as the subgraph of G induced by the vertex set $p_g(v) \cup p_r(v) \cup \{w \mid w \prec_g v\}$; $R_g(v)$ and $R_r(v)$ are defined in a similar way;
- one case is added to those of Property 3 of Lemma 6, since there may be neither two colors $i, j \in \{b, g, r\}$, $i \neq j$, such that $p_i(v)$ and $p_j(u)$ cross, nor one color $k \in \{b, g, r\}$ such that either $p_k(u) \subset p_k(v)$ or $p_k(v) \subset p_k(u)$;
- in the algorithm, the role of the external face is now played by the first ear (a cycle), and a new thread is added to take into account the new case: if this case applies, the three disjoint paths with endpoints u and v are the paths along T_i, $i \in \{b, g, r\}$.

Theorem 7 *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 time $O(\ell)$, where ℓ is the size of the reported paths.*

6 Planar 3-Path Queries in a Dynamic Environment

In this section, we present a semi-dynamic version of the data structure of Theorem 6, which supports a sequence of intermixed queries and insertions of vertices and edges.

6.1 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 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* analogously. The following lemmas characterize the chromatic structure of a face induced by the realizer.

Lemma 10 *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} such that:*

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

Lemma 11 *The external face of G can be decomposed into three counterclockwise consecutive paths p_{gb}, p_{rg}, p_{br} such that:*

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

6.2 Updating Realizers

The following lemmas show how to update a realizer of a triconnected plane graph after certain standard operations that add an edge or a subgraph are performed. They can be proved applying Lemmas 10 and 11.

Lemma 12 *Let G' be a triconnected plane graph, and let x and y be two nonadjacent vertices of G' lying on the same face f. Suppose a realizer is given for G'. Then a realizer for $G = G' \cup (x, y)$ can be constructed by coloring (x, y) and:*

- *if f is internal, by changing the colors of a constant number of edges of f;*
- *if f is external, by possibly changing the colors of the edges of a subpath f' of f, the color of one internal edge for each vertex of f', and the choice of the sink of one tree.*

Lemma 13 *Let G' be a triconnected plane graph, and let (x, y) and z be an edge and a vertex of G' on the same face f. Replace (x, y) with a plane graph G'' and insert a new edge (w, z), such that w is a vertex of G'' and the resulting graph G is a triconnected plane graph. Suppose a realizer is given for G'. Then a realizer for G can be constructed by coloring (w, z) and the edges of G'', and:*

- *if f is internal, by changing the colors of a constant number of edges of f;*
- *if f is external, by possibly changing the colors of the edges of a subpath f' of f, the color of one internal edge for each vertex of f', and the choice of the sink of one tree.*

Lemma 14 *Let G' be a triconnected plane graph and let (x'', y'') and (x''', y''') be two edges of G on the same face f. Replace (x'', y'') with a plane graph G'' and (x''', y''') with a plane graph G'''. Also, insert a new edge (w'', w'''), such that w'' is a vertex of G'', w''' is a vertex of G''', and the resulting graph G is a triconnected plane graph. Suppose a realizer is given for G'. Then a realizer for G can be constructed by coloring (w'', w''') and the edges of G'' and G''', and:*

- *if f is external, by changing the colors of the edges of a subpath f' of f, the color of one internal edge for each vertex of f', and the choice of the sink of one tree.*

An example of the replacement mechanism of Lemma 14 is shown in Fig. 6. Observe that in the construction of Lemmas 12–14 the inmost and outmost paths may change.

6.3 Data Structure and Complexity

We answer 3-path queries in a semi-dynamic environment where queries are intermixed with insertions of vertices and edges by suitably augmenting the on-line data structure of [12, 13] for 3-connectivity queries. This is essentially done by storing with each triconnected component the data structure of Theorem 5.

The insertion of an edge in the graph may cause two or more "old" triconnected components to be merged into a "new" triconnected component G. Let G' be the largest of the old components. We construct a realizer of G by:

1. choosing the external face of G as the one containing at least two of the sinks of the realizers of G';
2. recoloring the internal edges of G that were external in G'.
3. extending to G the existing realizer G' by recoloring all the edges of the other old components.

By Lemmas 13–14, for each edge of a triconnected component, the recoloring in Step 2 can be executed only once, while the recoloring of Step 3 can be executed $O(\log n)$ times.

Omitting numerous details, we conclude.

Theorem 8 *There exists a data structure for planar graphs that supports a sequence of t operations, each a k-path query with $k \leq 3$ or the insertion of a vertex or an edge, starting from the empty graph, in time $O(n \log n + t + L)$, where n is the total number of vertices inserted and L is the total size of the paths reported by the queries performed. At any time, the space requirement is linear in the current number of vertices, and a k-path query ($k \leq 3$) is answered in time $O(\ell)$ (worst-case), where ℓ is the size of the reported paths.*

7 Open Problems

Our results motivate the study of the following open problems:

- Improve the performance of our semi-dynamic data structure.
- Exploit the presented techniques to devise fully-dynamic data structures that allow output-sensitive reporting of paths.
- Devise a semi-dynamic data structure for maintaining convex drawings of graphs.
- Extend the semi-dynamic results to nonplanar 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] 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.

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

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

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

[6] 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.

[7] 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.

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

[9] H. de Fraysseix, J. Pach, and R. Pollack. Small sets supporting Fary embeddings of planar graphs. In *Proc. 20th Annu. ACM Sympos. Theory Comput.*, pages 426–433, 1988.

[10] 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.

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

[12] G. Di Battista and R. Tamassia. On-line graph algorithms with SPQR-trees. In *Automata, Languages and Programming (Proc. ICALP '90)*, volume 442 of *Lecture Notes in Computer Science*, pages 598–611, 1990.

[13] G. Di Battista and R. Tamassia. On-line maintenance of triconnected components with SPQR-trees. *Algorithmica*, to appear. Preprint: Technical Report CS-92-40, Comput. Sci. Dept., Brown Univ., 1992.

[14] Y. Dinitz and A. Vainshtein. The connectivity carcass of a vertex subset in a graph and its incremental maintenance. In *Proc. 26th Annu. ACM Sympos. Theory Comput.*, pages 716–725, 1994.

[15] D. Eppstein, Z. Galil, G. F. Italiano, and A. Nissenzweig. Sparsification: A technique for speeding up dynamic graph algorithms. In *Proc. 33rd Annu. IEEE Sympos. Found. Comput. Sci.*, pages 60–69, 1992.

[16] D. Eppstein, Z. Galil, G. F. Italiano, and T. H. Spencer. Separator based sparsification for dynamic planar graph algorithms. In *Proc. 25th Annu. ACM Sympos. Theory Comput.*, pages 208–217, 1993.

[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] 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.

[20] Z. Galil and G. F. Italiano. Maintaining biconnected components of dynamic planar graphs. In *Automata, Languages and Programming (Proc. ICALP '91)*, volume 510 of *Lecture Notes in Computer Science*, pages 339–350. Springer-Verlag, 1991.

[21] Z. Galil, G. F. Italiano, and N. Sarnak. Fully dynamic planarity testing. In *Proc. 24th Annu. ACM Sympos. Theory Comput.*, pages 495–506, 1992.

[22] D. Giammarresi and G. F. Italiano. Dynamic 2- and 3-connectivity on planar graphs. In *Proc. 3rd Scand. Workshop Algorithm Theory*, volume 621 of *Lecture Notes in Computer Science*, pages 221–232. Springer-Verlag, 1992.

[23] 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.

[24] J. Hershberger, M. Rauch, and S. Suri. Fully dynamic 2-edge-connectivity in planar graphs. In *Proc. 3rd Scand. Workshop Algorithm Theory*, volume 621 of *Lecture Notes in Computer Science*, pages 233–244. Springer-Verlag, 1992.

[25] A. Itai and M. Rodeh. The multi-tree approach to reliability in distributed networks. In *Proc. 25th Annu. IEEE Sympos. Found. Comput. Sci.*, pages 137–147, 1984.

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

[27] G. F. Italiano and Z. Galil. Fully dynamic algorithms for edge connectivity problems. In *Proc. 23rd Annu. ACM Sympos. Theory Comput.*, pages 317–327, 1991.

[28] G. F. Italiano, J. A. La Poutré, and M. H. Rauch. Fully dynamic planarity testing in planar embedded graphs. In *Algorithms (Proc. ESA '93)*, volume 726 of *Lecture Notes in Computer Science*, pages 212–223. Springer-Verlag, 1993.

[29] 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.

[30] G. Kant. Drawing planar graphs using the *lmc*-ordering. In *Proc. 33rd Annu. IEEE Sympos. Found. Comput. Sci.*, pages 101–110, 1992.

[31] 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.

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

[33] J. A. La Poutré. Maintenance of triconnected components of graphs. In *Automata, Languages and Programming (Proc. ICALP '92)*, volume 623 of *Lecture Notes in Computer Science*, 1992.

[34] 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.

[35] 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.

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

[37] M. Rauch. Fully dynamic biconnectivity in graphs. *Proc. 33rd Annu. IEEE Sympos. Found. Comput. Sci.*, pages 50–59, 1992.

[38] M. Rauch. Improved data structures for fully dynamic biconnectivity. In *Proc. 26th Annu. ACM Sympos. Theory Comput.*, pages 686–695, 1994.

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

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

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

[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] 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.

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

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

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

A Proofs

Sketch of Proof of Theorem 1: The data structure simply stores rooted trees T_s and T_t with parent pointers. Query 2-paths for vertices u and v is performed by an algorithm that traces one edge at at time paths $p_s(u)$, $p_s(v)$, $p_t(u)$, and $p_t(v)$, alternating between them, and stops when two disjoint paths between u and v have been formed, which is guaranteed to eventually happen by Lemma 1. $\square$

Sketch of Proof of Theorem 2: Whenever an edge insertion causes two or more "old" blocks to be merged into a "new" block, we construct the spanning trees of the new block by rerooting the spanning trees of all but the largest block. Hence, each vertex is involved in a rerooting $O(\log n)$ times. $\square$

Sketch of Proof of Lemma 3: We can construct a realizer by coloring the edges of G as follows:

- a canonical ordering of the vertices of G is computed;
- v_1, v_2, and v_n are the sinks of the blue, green, and red tree, respectively;
- (v_2, v_1) is colored blue and green.
- for each $2 \leq k \leq H$:
  1. 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 colored blue, and possibly red if c_l has no neighbor in $G - G_k$; (v_{s_k}, c_r) is colored green, and possibly red if c_r has no neighbor in $G - G_k$; edges (v_{s_k}, c_h), $r < h < l$ are colored red (see Fig. 2.a; from now on we represent a 2-colored edge half with one color and half with the other; dashes represent optionality);
  2. 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 colored blue, and possibly red if c_l has no neighbor in $G - G_k$; (v_{s_k}, c_r) is colored green and possibly red if c_r has no neighbor in $G - G_k$; edges (v_h, v_{h+1}), $s_k \leq h < s_k + d_k$ are colored blue and green (see Fig. 2.b).

Notice that, for each $2 \leq k \leq H$, the following invariants hold:

- every vertex of G_k has exactly one outgoing blue edge (but v_1), exactly one outgoing green edge (but v_2), and at most one outgoing red edge (but v_n);
- G_k contains no cycle such that a common color is assigned to all its edges.

By Lemma 2, the above construction can be carried out in linear time. $\square$

Sketch of Proof of Lemma 5: Suppose, for a contradiction, that $p_i(u, v)$ and $p_j(v, u)$ have only vertices u and v in common, and w.l.o.g. suppose that $p_j(v, u) \in R_k(u)$, $k \neq i, j$. It is easy to see that for u Property 5b of the realizers cannot be satisfied. Thus, $p_i(u, v)$ and $p_j(v, u)$ have a third vertex w in common besides u and v. Recursively repeating the argument on $p_i(u, w)$ and $p_j(w, u)$, and on $p_i(w, v)$ and $p_j(v, w)$ completes the proof. $\square$

Sketch of Proof of Lemma 6: *Properties 1 and 2.* Easily follow from the construction of a realizer of G.

Property 3. We consider the structure of path $p_r(u)$ (the structures of $p_b(u)$ and of $p_g(u)$ are analogous). We also suppose that u and v do not coincide with v_b, v_g, and v_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 4 and 5, 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 v_r. The vertices of path $p_{r1}(u)$ belong to $\bar{R}_r(v)$; either the vertices of $p_{r2}(u)$ belong to $\bar{p}_b(v)$ and the vertices of $p_{r3}(u)$ belong to $\bar{R}_g(v)$, or 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 $\bar{p}_r(v)$.

- if u is 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 is in $\bar{p}_b(v)$ or 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 is in $\bar{R}_b(v)$ or 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 is 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 Property 3.

Property 4. Can be proved by using Lemmas 4 and 5. □

Sketch of Proof of Theorem 4: The completeness of the algorithm in analyzing all the possible cases is guaranteed by the exhaustiveness of the two cases of Property 3 of Lemma 6 and by the easily checkable completeness of the if and case statements.

In proving the disjointness of the paths we refer to the labels a–f that the cases have in the algorithm.

Lemma 8 (Lemma 9) is used to determine if one of Cases a–c (d–f) applies. In the rest of the proof we concentrate on Case c and Case d; the other cases are similar and are omitted for brevity sake.

Case c (see Fig. 5.c). First, we prove that p_1 and p_2 are disjoint. By Property 1 of Lemma 6, $p_b(u, lca_b)$ and $p_g(u, v_g)$ ($p_b(v, lca_b)$ and $p_r(v, v_r)$) are disjoint. By Case 3a of Lemma 6, and since $p_r(u)$ and $p_g(v)$ cross, $p_b(u, lca_b)$ and $p_r(v, v_r)$ ($p_b(v, lca_b)$ and $p_g(u, v_g)$) are disjoint.

Second, we prove that p_1 and p_3 are disjoint. By Property 1 of Lemma 6, $p_b(u, lca_b)$ and $p_r(u, stopvertex_{gr})$ ($p_b(v, lca_b)$ and $p_g(v, stopvertex_{gr})$) are disjoint. By Case 3a of Lemma 6 and since $p_r(u)$ and $p_g(v)$ cross, $p_b(u, lca_b)$ and $p_g(v, stopvertex_{gr})$ ($p_b(v, lca_b)$ and $p_r(u, stopvertex_{gr})$) are disjoint.

Third, we prove that p_2 and p_3 are disjoint. By Property 1 of Lemma 6, $p_r(u, stopvertex_{gr})$ and $p_g(u, v_g)$ ($p_g(v, stopvertex_{gr})$ and $p_r(v, v_r)$) are disjoint. Since $p_r(u)$ and $p_g(v)$ cross, it follows from Corollary 1 that $p_r(u, stopvertex_{gr}) \subset R_r(v)$ ($p_g(v, stopvertex_{gr}) \subset R_g(u)$), while, by Property 1 of Lemma 6, $p_r(v) \cap R_r(v) = \{v\}$ ($p_g(u) \cap R_g(u) = \{u\}$); hence, $p_r(u, stopvertex_{gr})$ and $p_r(v, v_r)$ ($p_g(v, stopvertex_{gr})$ and $p_g(u, v_g)$) 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 by Property 3a of Lemma 6. Path p_3 is simple by being $p_r(u)$ and $p_g(v)$ simple and by Lemma 5.

Case d (see Fig. 5.d). First, we prove that p_1 and p_2 are disjoint. By Property 1 of Lemma 6, $p_g(u, lca_g)$ and $p_r(u, lca_r)$ ($p_g(v, lca_g)$ and $p_r(v, lca_r)$) are disjoint. By Case 3b of Lemma 6, and by being $p_b(u)$ a proper subpath of $p_b(v)$, $p_g(u, lca_g)$ and $p_r(v, lca_r)$ ($p_r(u, lca_r)$ and $p_g(v, lca_g)$) are disjoint.

Second, we prove that p_1 and p_3 are disjoint. By Property 1 of Lemma 6, $p_g(v, lca_g)$ and $p_b(u, v)$ are disjoint. By Case 3b of Lemma 6 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 Property 1 of Lemma 6, $p_r(v, lca_r)$ and $p_b(u, v)$ are disjoint. By Case 3b of Lemma 6 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 (p_2) is composed by the two paths along T_g (T_r) between u or v and their lowest common ancestor, and that p_3 is a subpath of simple path $p_b(v)$. □

Sketch of Proof of Lemma 7: The first sentence is proved by contradiction as follows. Since $u \equiv_b v$ and $u \equiv_g v$, we have that u and v are vertices of a path along T_b and of a path along T_g. By Lemma 5, $p_b(u, v)$ and $p_g(v, u)$ coincide. With the same argument, we prove that $p_r(u, v)$ exists and that $p_r(u, v)$ and $p_g(v, u)$ coincide. It follows that there is an edge of the realizer with three colors, which contradicts Property 3 of the realizers. The second sentence is proved by contradiction observing that the circular order of the colors of the outgoing edges around u and v and the fact that $u \prec_b v$ and $u \prec_g v$ and $u \prec_r v$ imply that $p_b(v)$ crosses both $p_g(u)$ and $p_r(u)$. This contradicts Case 3a of Lemma 6. □

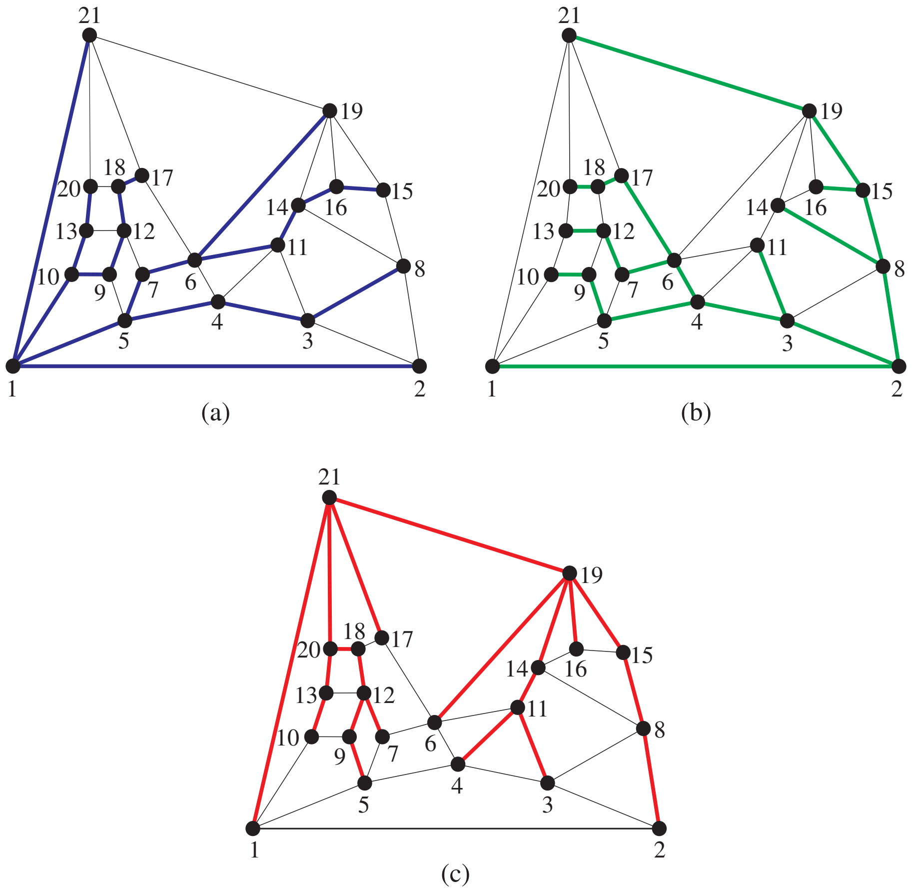

Figure 1: (a) The blue tree of a triconnected plane graph G. (b) The green tree of G. (c) The red tree of G.

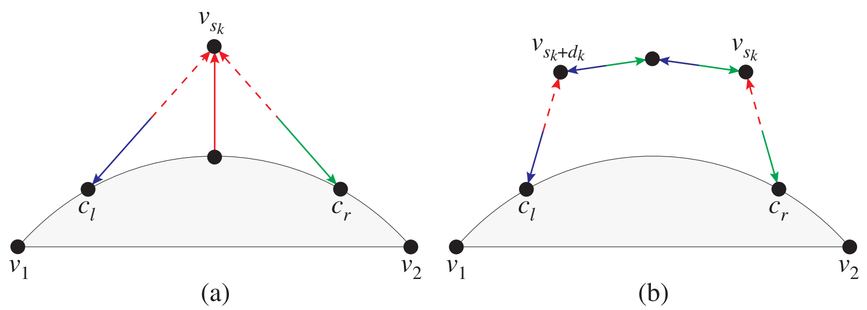

Figure 2: The coloring of the edges in the construction of a realizer. (a) $V_k = \{v_{s_k}\}$. (b) $V_k = \{v_{s_k}, \ldots, v_{s_k+d_k}\}$.

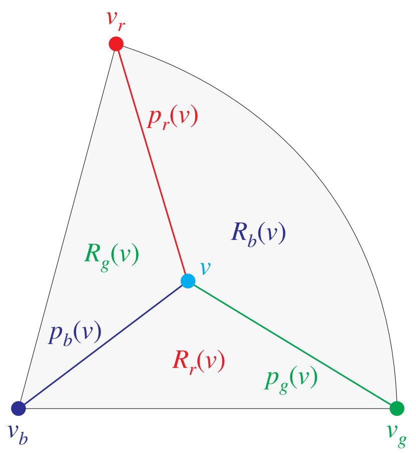

Figure 3: The blue, green, red paths and regions of a vertex of a triconnected plane graph.

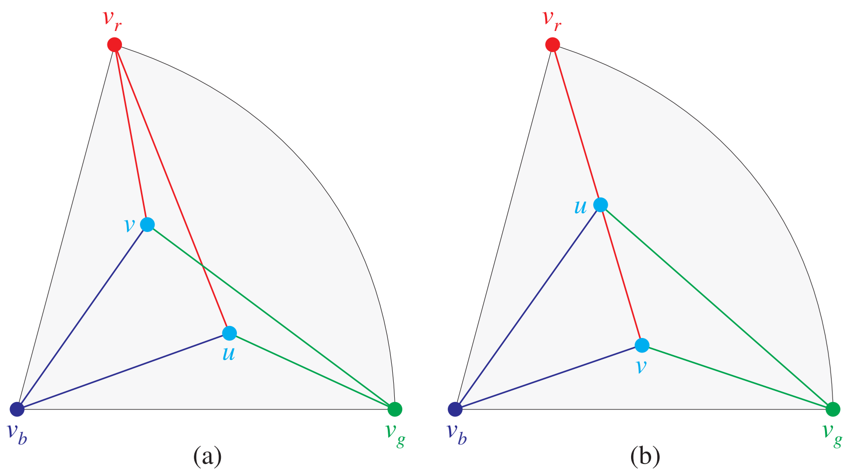

Figure 4: (a) Crossing paths. (b) Non-crossing paths.

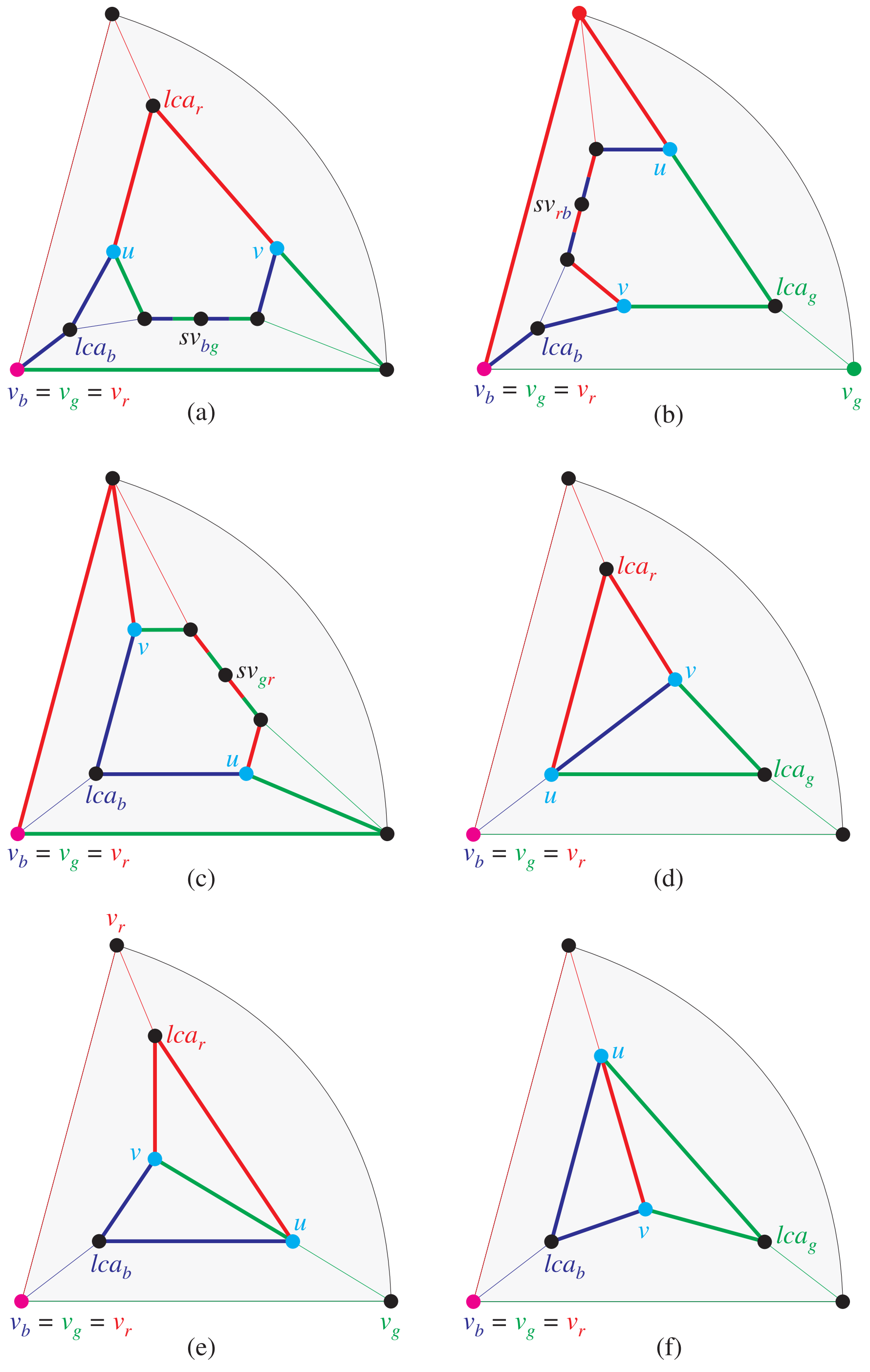

Figure 5: The cases of disjoint paths with endpoints u and v in Algorithm ThreePaths.

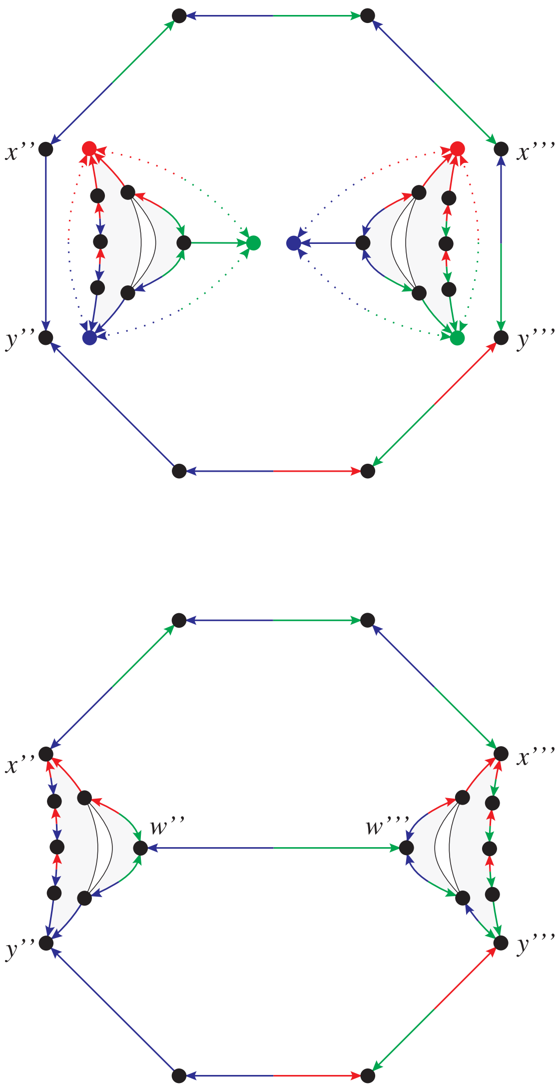

Figure 6: Replacing two edges of a face with two components joined by an edge.