Algorithmica (2005) 41: 289–308
DOI: 10.1007/s00453-004-1128-8

Algorithmica

Inserting an Edge into a Planar Graph[1]

Carsten Gutwenger,[2] Petra Mutzel,[3] and René Weiskircher[3]

Abstract. Computing a crossing minimum drawing of a given planar graph G augmented by an additional edge e where all crossings involve e, has been a long standing open problem in graph drawing. Alternatively, the problem can be stated as finding a combinatorial embedding of a planar graph G where the given edge e can be inserted with the minimum number of crossings. Many problems concerned with the optimization over the set of all combinatorial embeddings of a planar graph turned out to be NP-hard. Surprisingly, we found a conceptually simple linear time algorithm based on SPQR-trees, that is able to find a solution with the minimum number of crossings.

Key Words. Crossing minimization, Combinatorial embeddings, Planarization, Graph drawing.

1. Introduction. Crossing minimization is among the most challenging problems in graph theory and graph drawing. Although there is a vast amount of literature on the problem, so far no practically efficient exact algorithm for crossing minimization is known. Currently, the best known approach for crossing minimization is based on planarization. Here, in a first step, a preferably small number of edges is deleted so that the resulting graph is planar. Then the edges are iteratively re-inserted into the planar subgraph so that the number of crossings is minimized. Usually, the second step is done in the following way: Fix an arbitrary combinatorial embedding Π of the planar subgraph P and re-insert the first deleted edge e_1. This is done by solving a shortest path problem in the augmented (geometrical) dual graph of P associated with Π, since every crossing of e_1 corresponds to using an edge in the dual graph. Then the crossings are substituted by artificial vertices, yielding a planar subgraph again. Now, the next edge can be inserted, and so on.

One criticism of the planarization method was that when choosing a "bad" embedding in the edge re-insertion phase, the number of crossings may get much higher than necessary [HS]. Hence, the question arose if there is a polynomial time algorithm for inserting an edge into the planar subgraph P so that the number of crossings is minimized. Therefore, the task is to optimize over the set of all possible combinatorial embeddings of P.

A graph is p*lanar* if it can be drawn in the plane without any edge crossings. (*Combinatorial*) *embeddings* are equivalence classes of planar drawings which can be defined by the sequence of the incident edges around each vertex in a drawing. We consider two

[1] A preliminary version of this paper was published in [GMW].

[2] Research Center caesar, Ludwig-Erhard-Allee 2, D-53175 Bonn, Germany. gutwenger@caesar.de.
[3] Vienna University of Technology, Favoritenstrasse 9–11 E186, A-1040 Vienna, Austria. {mutzel, weiskircher}@ads.tuwien.ac.at.

Received June 30, 2003; revised September 15, 2004. Communicated by S. Albers.
Online publication January 21, 2005.

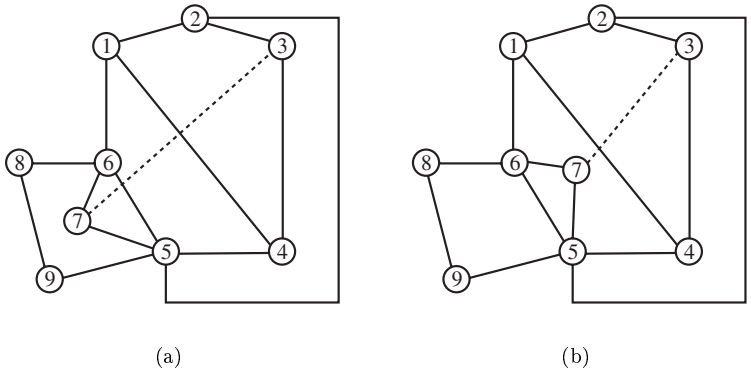

Fig. 1. The number of crossings required when inserting an edge highly depends on the chosen embedding.

drawings of the same graph equivalent if the circular sequences of the incident edges around each vertex in clockwise order is the same. In this case they realize the same combinatorial embedding.

While it is possible to compute an arbitrary combinatorial embedding for a planar graph in linear time [MM], [CNAO], it is often hard to optimize over the set of all possible combinatorial embeddings. For example, the problem of bend minimization can be solved in polynomial time for a fixed combinatorial embedding [T], while it is NP-hard over the set of all combinatorial embeddings [GT]. When a linear function of polynomial size is defined on the cycles of a graph, it is NP-hard to find the embedding that maximizes the value of the cycles that are face cycles in the embedding [MW2], [MW1]. Note that the number of combinatorial embeddings of a planar graph may be exponential.

Figure 1 shows a simple case in which the choice of the combinatorial embedding of the planar subgraph has an impact on the number of crossings produced when inserting the dashed edge. When choosing the embedding of Figure 1(a) for the planar subgraph (without the dashed edge), we get two crossings, while the optimal crossing number over the set of all combinatorial embeddings is one (see Figure 1(b)).

Formally, we define the *edge insertion problem* as follows: Given a planar graph $G = (V, E)$ and a pair of vertices (v_1, v_2) in G, find an embedding Π of G such that we can add the edge $e = (v_1, v_2)$ to Π with the minimum possible number of crossings among all embeddings of G.

This paper shows that the edge insertion problem can be solved in polynomial time, thus solving a long standing open problem in graph drawing. We present a conceptually simple linear time algorithm based on SPQR-trees which is able to solve the edge insertion problem to optimality. Note that an optimal solution of the edge insertion problem does not necessarily lead to a drawing of the graph $G' = (V, E \cup \{e\})$ with the minimum number of crossings. This is due to the fact that there may not always be a drawing with the minimum number of crossings such that $G = (V, E)$ is drawn without crossings.

The rest of the paper is organized as follows. In Section 2 we give a short introduction to SPQR-trees and introduce the concept of traversing costs. Section 3 contains the algorithm for solving the edge insertion problem for planar biconnected graphs. We also prove the correctness and discuss the running time. In Section 4 we present a gener-

alization of the algorithm for arbitrary planar graphs. Again we prove the correctness of the algorithm and analyze the running time. Section 5 gives computational results on a set of benchmark graphs. They show that the new method reduces the number of crossings in drawings computed using the planarization approach significantly. The last section addresses the difference between finding a drawing with the minimum number of crossings and the problem discussed in this paper.

2. Preliminaries

2.1. *SPQR-Trees*. In this section we give a brief introduction to the SPQR-tree data structure for biconnected planar graphs. A *cut vertex* of a graph is a vertex whose removal increases the number of connected components. A connected graph that has no cut vertex is called *biconnected*. A set of two vertices whose removal increases the number of connected components is called a *separation pair*. A biconnected graph without a separation pair is called *triconnected*.

SPQR-trees were introduced by Di Battista and Tamassia [BT] and since then have been used in various graph drawing applications like, e.g., minimizing the number of bends in an orthogonal drawing [BBD], [MW1] or finding planar embeddings with minimum depth and maximum external face [GM3]. Recently, SPQR-trees have also been applied in the area of circuit design [FOO].

SPQR-trees represent the decomposition of a planar biconnected graph according to its split pairs. Let G be a planar biconnected graph. A *split pair* of G is either a separation pair or a pair of adjacent vertices. A *split component* of a split pair $\{u, v\}$ is either an edge (u, v) or a maximal subgraph C of G such that $\{u, v\}$ is not a split pair of C. Let $\{s, t\}$ be a split pair of G. A *maximal split pair* $\{u, v\}$ of G with respect to $\{s, t\}$ is such that, for any other split pair $\{u', v'\}$, vertices u, v, s, and t are in the same split component.

Let $e = (s, t)$ be an edge of G, called the *reference edge*. The SPQR-tree $\mathcal{T}$ of G with respect to e is a rooted ordered tree whose nodes are of four types: S, P, Q, and R. Each node μ of $\mathcal{T}$ has an associated biconnected multi-graph, called the *skeleton* of μ. Tree $\mathcal{T}$ is recursively defined as follows:

Trivial Case. If G consists of exactly two parallel edges between s and t, then $\mathcal{T}$ consists of a single Q-node whose skeleton is G itself.

Parallel Case. If the split pair $\{s, t\}$ has k split components $G_1, \ldots, G_k$ with $k \geq 3$, the root of $\mathcal{T}$ is a P-node μ, whose skeleton consists of k parallel edges $e = e_1, \ldots, e_k$ between s and t.

Series Case. Otherwise, the split pair $\{s, t\}$ has exactly two split components, one of them is e, and the other one is denoted with G'. 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 $\mathcal{T}$ is an S-node μ, whose skeleton is a cycle $e_0, e_1, \ldots, e_k$, where $e_0 = e$, $c_0 = s$, $c_k = t$, and $e_i = (c_{i-1}, c_i)$ ($i = 1, \ldots, k$).

Rigid Case. If none of the above cases 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 $\{s_i, t_i\}$ but the one containing e. The root of $\mathcal{T}$ is an R-node, whose skeleton is obtained from G by replacing each subgraph G_i with the edge $e_i = (s_i, t_i)$.

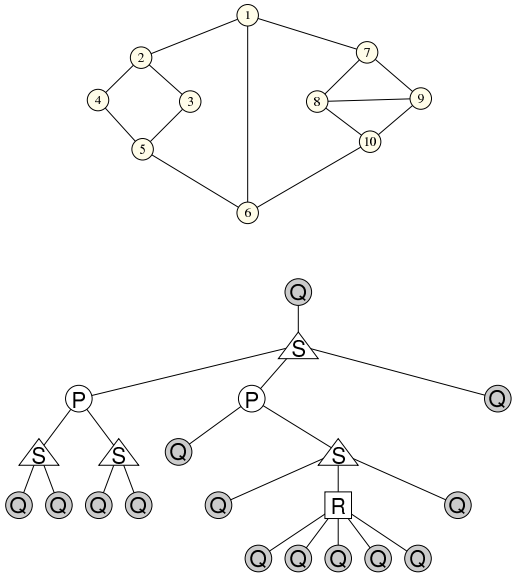

Fig. 2. A biconnected planar graph and its SPQR-tree.

Except for the trivial case, μ has children $\mu_1, \ldots, \mu_k$, such that μ_i is the root of the SPQR-tree of $G_i \cup e_i$ with respect to e_i ($i = 1, \ldots, k$). The endpoints of the edge e_i are called the *poles* of node μ_i. Edge e_i is said to be the *virtual edge* of node μ_i in the skeleton of μ and of node μ in the skeleton of μ_i. We call node μ the *pertinent node* of e_i in the skeleton of μ_i, and μ_i the *pertinent node* of e_i in the skeleton of μ. The virtual edge of μ in the skeleton of μ_i is called the reference edge of μ_i.

Let μ_r be the root of $\mathcal{T}$ in the decomposition given above. We add a Q-node representing the reference edge e and make it the parent of μ_r so that it becomes the new root. Figure 2 shows an example of a graph and its SPQR-tree.

Let e be an edge in *skeleton*(μ) and let ν be the pertinent node of e. Deleting edge $\{\mu, \nu\}$ in $\mathcal{T}$ splits $\mathcal{T}$ into two connected components. Let $\mathcal{T}_\nu$ be the connected component containing ν. The *expansion graph* of e (denoted with *expansion*(e)) is the graph induced by the edges of G contained in the skeletons of the Q-nodes in $\mathcal{T}_\nu$. We further introduce the notation *expansion*$^+(e)$ for the graph *expansion*(e) $\cup\, e$. Figure 3 gives an example for the expansion graph of an edge. The *pertinent graph* of a tree node μ is obtained from the skeleton of μ by replacing each skeleton edge except for the reference edge of μ with its expansion graph. Examples for the pertinent and skeleton graphs of the different node types are shown in Figure 4. If v is a vertex in G, a node in $\mathcal{T}$ whose skeleton contains v is called an *allocation node* of v.

SPQR-trees can be constructed in linear time and their size including the skeleton graphs is linear in the size of the original graph [BT, GM1]. Choosing a different reference edge e' is equivalent to rooting the tree $\mathcal{T}$ at the Q-node whose skeleton contains e'. In particular, the unrooted version of the SPQR-tree of a planar biconnected graph (including the skeleton graphs) is unique.

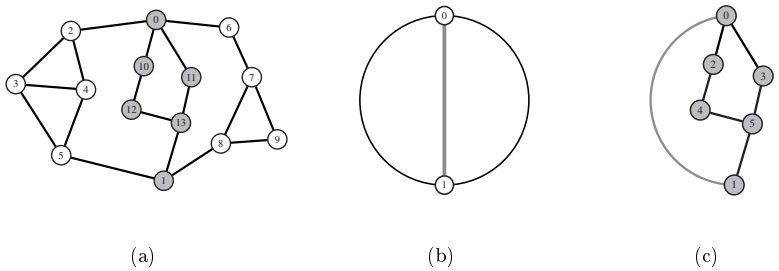

Fig. 3. Example for the expansion graph of a skeleton edge: (a) a biconnected planar graph G, (b) the skeleton μ of a P-node in the SPQR-tree of G, and (c) the graph $expansion^+(e)$ for the gray edge e in $skeleton(\mu)$.

As described in [BT], SPQR-trees can be used to represent all combinatorial embeddings of a biconnected planar graph. This is done by choosing combinatorial embeddings for the skeletons of the nodes in the tree. The skeletons of S- and Q-nodes are simple cycles, so they have only one embedding. The skeletons of R-nodes are always triconnected graphs. According to the definition of combinatorial embeddings, a triconnected graph has two embeddings which are mirror-images of each other, i.e., the order of the edges around each vertex is reversed in the mirror embedding. The number of embeddings of a P-node skeleton with k edges is $(k-1)!$.

Every embedding of the original graph defines a unique embedding for each skeleton of a node in the SPQR-tree. Conversely, when we define an embedding for each skeleton of a node in the SPQR-tree, we define a unique embedding for the original graph. Thus,

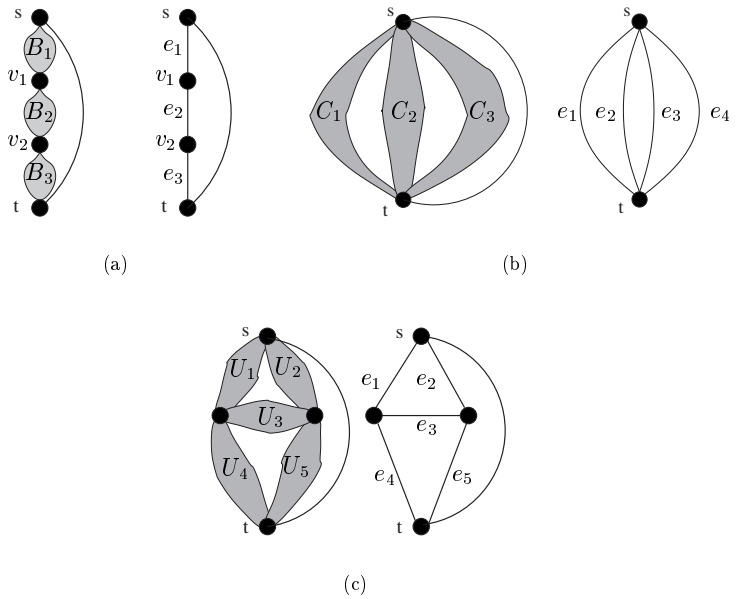

Fig. 4. Pertinent and skeleton graphs of the different node types of an SPQR-tree. The shaded regions represent subgraphs. (a) An S-node, (b) a P-node, and (c) an R-node.

if the SPQR-tree of G has r R-nodes and P-nodes P_1 to P_k where the skeleton of P_i has p_i edges, then the total number of combinatorial embeddings of G is

$$2^r \prod_{i=1}^{k} (p_i - 1)!.$$

In [BBD] SPQR-trees are used to enumerate all combinatorial embeddings of a biconnected planar graph within a branch-and-bound algorithm for finding a combinatorial embedding and an external face for a graph such that the drawing computed by Tamassia's algorithm [T] has the minimum number of bends among all possible orthogonal drawings of the graph.

2.2. *Traversing Costs*. Traversing costs of skeleton edges are a fundamental concept used in the edge insertion algorithm. First, we give formal definitions of the two terms *dual graph* and *edge insertion path*.

DEFINITION 1 (Dual Graph). Let G be a connected planar graph and let Π be a combinatorial embedding of G. Then the *dual graph* $\Pi^* = (F, E^*)$ of G with respect to Π is defined as follows:

1. For each face $f \in \Pi$ there is one vertex f^* in F.
2. For each edge $e \in E$, there is one edge e^* in E^*. If e is on the boundary of two different faces $f_1, f_2 \in \Pi$, then $e^* = (f_1^*, f_2^*)$; otherwise e is on the boundary of only one face $f \in \Pi$ and $e^* = (f^*, f^*)$.

Throughout the paper we use the notations Π^* for the dual graph of G with respect to Π, e^* for the dual edge of e, and f^* for the dual vertex of face f.

An edge insertion path is basically associated with a path in the dual graph and determines the edges that are crossed when inserting an edge into a given embedding.

DEFINITION 2 (Edge Insertion Path). Let $G = (V, E)$ be a connected planar graph, and let Π be an embedding of G. Let v_1 and v_2 be two non-adjacent vertices in G. Then $e_1, \ldots, e_k$ is an *edge insertion path* for v_1 and v_2 in G with respect to Π if either $k = 0$ and v_1 and v_2 are contained in a common face in Π or the following conditions are satisfied:

1. $e_1, \ldots, e_k \in E$.
2. There is a face in Π with e_1 and v_1 on its boundary.
3. There is a face in Π with e_k and v_2 on its boundary.
4. $e_1^*, \ldots, e_k^*$ is a path in Π^*.

If $p = e_1, \ldots, e_k$ is an edge insertion path for v_1 and v_2 with respect to Π, then it is possible to insert the edge (v_1, v_2) into Π with k crossings, where the ith crossing involves edge (v_1, v_2) and edge e_i for $1 \leq i \leq k$. The length of p, denoted by $|p|$, is k. We call p an *optimal edge insertion path* for v_1 and v_2 in G, if there is no shorter edge insertion path for v_1 and v_2 in G with respect to any embedding of G.

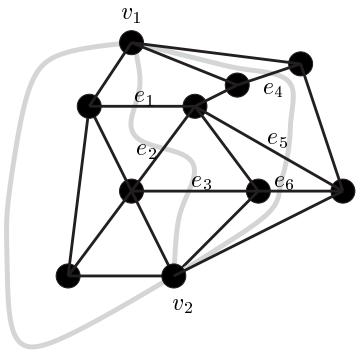

Fig. 5. Three different edge insertion paths for v_1 and v_2.

Figure 5 shows three different edge insertion paths for v_1 and v_2 with respect to the embedding realized by the drawing. The three paths are the empty path, the path e_1, e_2, e_3, and the path e_4, e_5, e_6. In this case the empty path is the optimal edge insertion path for v_1 and v_2.

The *traversing costs* $c(e)$ of a skeleton edge e are defined as follows. Consider an arbitrary embedding Π of the graph *expansion*$^+(e)$ and its dual graph Π^*. Let f_1 and f_2 be the two faces in Π that are separated by e, and let f_1^* and f_2^* be the corresponding vertices in the dual graph. We denote with $P(\Pi^*, e)$ the shortest path in Π^* that connects f_1^* and f_2^* and does not use edge e^*. Lemma 1 below shows that the length of this path is independent of the embedding Π chosen for *expansion*$^+(e)$. Thus we define the *traversing costs* $c(e)$ simply as

$$c(e) = \text{length of the path } P(\Pi^*, e) \text{ for any embedding } \Pi \text{ of } \textit{expansion}^+(e).$$

LEMMA 1. *Let μ be a node in $\mathcal{T}$ and let e be an edge in skeleton(μ). Then the length of the path $P(\Pi^*, e)$ is independent of the embedding Π of expansion$^+(e)$.*

PROOF. The expansion graph of e is defined using a subtree of the SPQR-tree $\mathcal{T}$. Let ν be the pertinent node of e. We denote with $\mathcal{T}_e$ the subtree of $\mathcal{T}$ which is the connected component containing ν of the graph $\mathcal{T}\backslash\{(\mu, \nu)\}$. The root of $\mathcal{T}_e$ is the node ν which is not necessarily a Q-node. We prove the lemma by induction over the height of $\mathcal{T}_e$.

If the height of $\mathcal{T}_e$ is 1, then ν is a Q-node and *expansion*$^+(e)$ is a circle of two edges. Thus, *expansion*$^+(e)$ has only a single embedding Π and the length of the path $P(\Pi^*, e)$ is simply 1.

Assume now that the height of $\mathcal{T}_e$ is $k > 1$ and that the lemma holds for all skeleton edges $\tilde{e}$ for which the height of $\mathcal{T}_{\tilde{e}}$ is less than k. Since $k > 1$, the root ν of $\mathcal{T}_e$ is either an S-, P-, or R-node. We denote with e' the virtual edge of μ in *skeleton*(ν). An embedding Π of *expansion*$^+(e)$ induces an embedding Π_ν of *skeleton*(ν) and an embedding $\Pi_{\bar{e}}$ of *expansion*$^+(\bar{e})$ for each edge $\bar{e} \neq e'$ in *skeleton*(ν). Since the height of $\mathcal{T}_{\bar{e}}$ is less

than k, $c_{\bar{e}} = P(\Pi^*_{\bar{e}}, \bar{e})$ is independent of $\Pi_{\bar{e}}$. We consider the three possible types of node v:

S-node. The skeleton of v is a circle $e', e_1, \ldots, e_\ell$ with $\ell \geq 2$. The length of the path $P(\Pi^*, e)$ is $\min_{i=1}^{\ell} P(\Pi^*_{e_i}, e_i) = \min_{i=1}^{\ell} c_{e_i}$ which is independent of Π.

P-node. The skeleton of v consists of $\ell + 1$ parallel edges $e', e_1, \ldots, e_\ell$ with $\ell \geq 2$ and the length of the path $P(\Pi^*, e)$ is $\sum_{i=1}^{\ell} P(\Pi^*_{e_i}, e_i) = \sum_{i=1}^{\ell} c_{e_i}$ which is also independent of Π.

R-node. The skeleton of v is a triconnected planar graph $S = (V_S, E_S)$. The length of the path $P(\Pi^*, e)$ is the length of a shortest path in Π^*_v connecting the two faces separated by e' without using the dual edge of e' where each edge $\bar{e} \in E_S \backslash \{e'\}$ has cost $c_{\bar{e}}$. Since a triconnected planar graph has only two embeddings which are mirror-images of each other, the length of this path is independent of the embedding Π_v and thus independent of Π. □

According to Lemma 1, the traversal costs of a skeleton edge e can be computed by finding a shortest path in the dual graph of an arbitrary embedding of $expansion^+(e)$. This can be done in time $\mathcal{O}(|expansion^+(e)|)$ using a breadth first search approach.

3. Inserting an Edge into a Biconnected Graph. In this section we present an algorithm for inserting an edge into a biconnected planar graph. First, we define the augmented dual graph, which is used for finding a shortest edge insertion path in case of a fixed embedding.

DEFINITION 3 (Augmented Dual Graph). Let G be a planar graph and let Π be an embedding of G. Let v_1 and v_2 be two vertices in G. For $i = 1, 2$, let F_i be the set $\{f^* \mid f \text{ is a face with } v_i \text{ on its boundary}\}$. The *augmented dual graph* of Π, v_1, v_2 denotes the graph obtained from the dual graph Π^* by adding the vertices v_1 and v_2 and inserting the edges (v_1, f_1) for all $f_1 \in F_1$ and (v_2, f_2) for all $f_2 \in F_2$.

We further say a skeleton edge e *represents* a vertex v of G if v is contained in $expansion(e)$ and v is not an endpoint of e, and we introduce the following notation for list concatenation. If $L_1 = a_1, \ldots, a_k$ and $L_2 = b_1, \ldots, b_\ell$ are two lists, we denote with $L_1 + L_2$ the list $a_1, \ldots, a_k, b_1, \ldots, b_\ell$. The algorithm for computing an optimal edge insertion path for a biconnected planar graph G and two non-adjacent vertices v_1 and v_2 of G is shown in Algorithm 1. We remark that it is not necessary actually to construct graph G_i if μ_i is not an R-node. It is part of the algorithm, since we refer to G_i in the correctness proof.

In order to prove the correctness of Algorithm 1, we first show that the path computed by the algorithm is indeed an edge insertion path with respect to some embedding.

LEMMA 2. *Let $p_1 + \cdots + p_k$ be the path computed by Algorithm* 1. *Then there exists an embedding Π of G such that $p_1 + \cdots + p_k$ is an edge insertion path for v_1 and v_2 in G with respect to Π.*

Algorithm 1. Computes an optimal edge insertion path for a pair of non-adjacent vertices v_1, v_2 in a biconnected planar graph G.

```
OptimalBlockInserter (graph G, vertex v1, vertex v2)
    Compute the SPQR-tree T of G;
    Find the shortest path μ1, ..., μk in T between an allocation node μ1
    of v1 and μk of v2;
    for i = 1, ..., k do
        Si := skeleton(μi);
        if v1 is in Si then
            xi^1 := v1;
        else
            Split the edge representing v1 in Si by inserting a new vertex yi^1;
            Mark the two edges produced by the split;
            xi^1 := yi^1;
        end
        if v2 is in Si then
            xi^2 := v2;
        else
            Split the edge representing v2 in Si by inserting a new vertex yi^2;
            Mark the two edges produced by the split;
            xi^2 := yi^2;
        end
        let Gi be the graph obtained from Si by replacing each unmarked
        edge with its expansion graph;
        if μi is not an R-node then
            set pi to the empty path;
        else
            Compute an arbitrary embedding Πi of Gi;
            let Ai be the augmented dual graph of Πi, xi^1, xi^2;
            Compute the shortest path e0*, ..., eℓ+1* in Ai between xi^1 and
            xi^2;
            pi := e1, ..., eℓ , where ej is the primal edge of ej*;
        end
    end
    return p1 + ··· + pk;
end
```

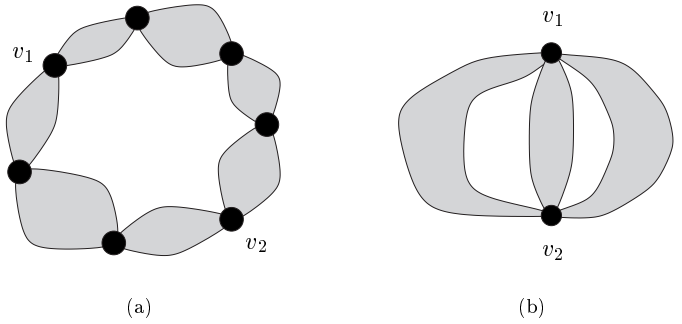

Fig. 6. Path Λ consists of a single node μ_1. (a) μ_1 is an S-node and (b) μ_1 is a P-node.

PROOF. Consider the path $\Lambda = \mu_1, \ldots, \mu_k$ computed by the algorithm. By construction of Λ, the skeleton of μ_1 contains v_1, the skeleton of μ_k contains v_2 (note that $k = 1$ is possible), and, for each $j = 2, \ldots, k-1$, the skeleton of μ_j contains neither v_1 nor v_2. Moreover, Λ does not contain a Q-node.

First, we prove the lemma for the case that Λ consists of a single node μ_1. In this case the skeleton of μ_1 contains both v_1 and v_2. There are three possible cases for the type of μ_1:

1. μ_1 **is an S-node.** Then v_1 and v_2 form a separation pair of G, see Figure 6(a). Let Π_1 be any embedding of G. Since (v_1, v_2) is a separation pair, v_1 and v_2 lie in a common face of Π_1. Thus, the empty path returned by the algorithm is an edge insertion path for v_1 and v_2 in G with respect to Π_1.
2. μ_1 **is a P-node.** Again, v_1 and v_2 are a separation pair of G and similar arguing as for the first case holds, see Figure 6(b).
3. μ_1 **is an R-node.** In this case the graph G_1 constructed by the algorithm is the original graph G, since all skeleton edges are expanded, and Π_1 computed by the algorithm is an embedding of G. Thus, the algorithm computes an edge insertion path in G for v_1 and v_2 with respect to embedding Π_1 of G.

Assume now that $k > 1$. We define graphs $H_1, \ldots, H_k$ as follows. H_i is obtained from $skeleton(\mu_i)$ by replacing all skeleton edges that do not represent vertex v_2 by their expansion graph, and, if $i < k$, splitting the skeleton edge that represents vertex v_2 introducing a new vertex r_i. The skeleton of μ_k contains vertex v_2 itself and we denote with r_k this vertex in $skeleton(\mu_k)$. We show by induction over i that there is an embedding Γ_i of H_i such that $p_1 + \cdots + p_i$ is an edge insertion path for v_1 and r_i in H_i with respect to Γ_i. The embeddings $\Gamma_1, \ldots, \Gamma_k$ are iteratively constructed during the proof.

$i = 1$. Consider the different types for node μ_1:
   1. μ_1 **is a P-node.** This case does not apply, since μ_2 is not an allocation node of v_1.
   2. μ_1 **is an S-node.** In this case, v_1 and r_1 form a separation pair in H_1 and v_1 and r_1 lie in a common face in any embedding of H_1 (see Figure 7(a)). Thus, Γ_1 is set to an arbitrary embedding of H_1 and the empty path p_1 computed by the algorithm is an edge insertion path for v_1 and r_1 in H_1 with respect to Γ_1.

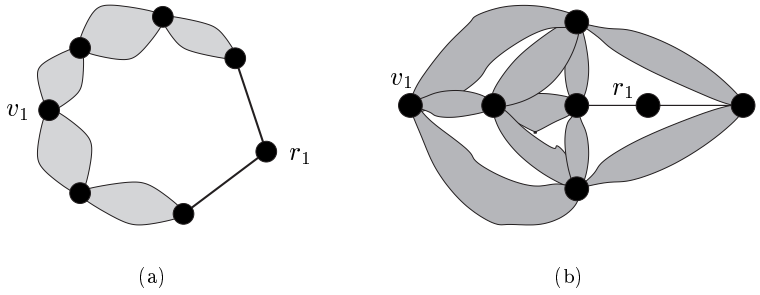

Fig. 7. The different node types for the case $i = 1$. μ_i is (a) an S-node and (b) an R-node.

3. μ_1 **is an R-node.** The graph G_1 constructed by the algorithm is the graph H_1 if r_1 is identified with vertex y_1^2 in the algorithm (see Figure 7(b)). Hence, Π_1 is also an embedding of H_1 and we define $\Gamma_1 := \Pi_1$. Since p_1 is an edge insertion path for v_1 and y_1^2 in G_1 with respect to Π_1 by construction, p_1 is also an edge insertion path for v_1 and r_1 in H_1 with respect to Γ_1.

$i > 1$. Assume now that $\Gamma_1, \ldots, \Gamma_{i-1}$ are already constructed and $p_1 + \cdots + p_{i-1}$ is an edge insertion path for v_1 and r_{i-1} in H_{i-1} with respect to Γ_{i-1}.

Graph G_i constructed in the algorithm contains a vertex x_i^1 adjacent to exactly two vertices, say a and b, and H_{i-1} contains vertex r_{i-1} adjacent to exactly two vertices, say a' and b'. By construction, both a and a', as well as b and b', represent the same vertex of G, and the graph H_i is obtained from G_i and H_{i-1} by identifying a and a', b and b', and removing the vertices x_i^1 and r_{i-1} (including their adjacent edges).

An embedding of H_i can be determined in the following way. Chose one of the two faces containing r_{i-1} as the external face of Γ_{i-1}. This leads to an embedding in which either the last edge of $p_1 + \cdots + p_{i-1}$ and r_{i-1} lie in a common face, or $p_1 + \cdots + p_{i-1}$ is empty and v_1 and r_{i-1} lie in a common face. Then determine an embedding of G_i, insert Γ_{i-1} into the embedding of G_i and remove the vertices r_{i-1} and x_i^1. It is also possible to mirror the embedding Γ_{i-1} before inserting, since $p_1 + \cdots + p_{i-1}$ is still an edge insertion path in H_{i-1} with respect to the mirror embedding of Γ_{i-1}.

We distinguish the possible cases for the type of node μ_i:

1. μ_i **is an S-node.** There is just one embedding of G_i and inserting H_{i-1} into this embedding as described above leads to an embedding Γ_i of H_i such that $p_1 + \cdots + p_{i-1} = p_1 + \cdots + p_i$ is an edge insertion path in H_i with respect to Γ_i (see Figure 8(a)).
2. μ_i **is a P-node.** Let Π_i be an embedding of G_i such that x_i^1 and x_i^2 lie in a common face. Obtain Γ_i by inserting Γ_{i-1} into Π_i in such a way that r_{i-1} and x_i^2 lie in a common face (see Figure 8(b)). This can be achieved by mirroring Γ_{i-1} if necessary. Then $p_1 + \cdots + p_{i-1} = p_1 + \cdots + p_i$ is an edge insertion path in H_i with respect to Γ_i.
3. μ_i **is an R-node.** Let Π_i be the embedding computed by the algorithm. The list $p_i = e_1, \ldots, e_\ell$ is an edge insertion path for x_i^1 and x_i^2 in G_i with respect to

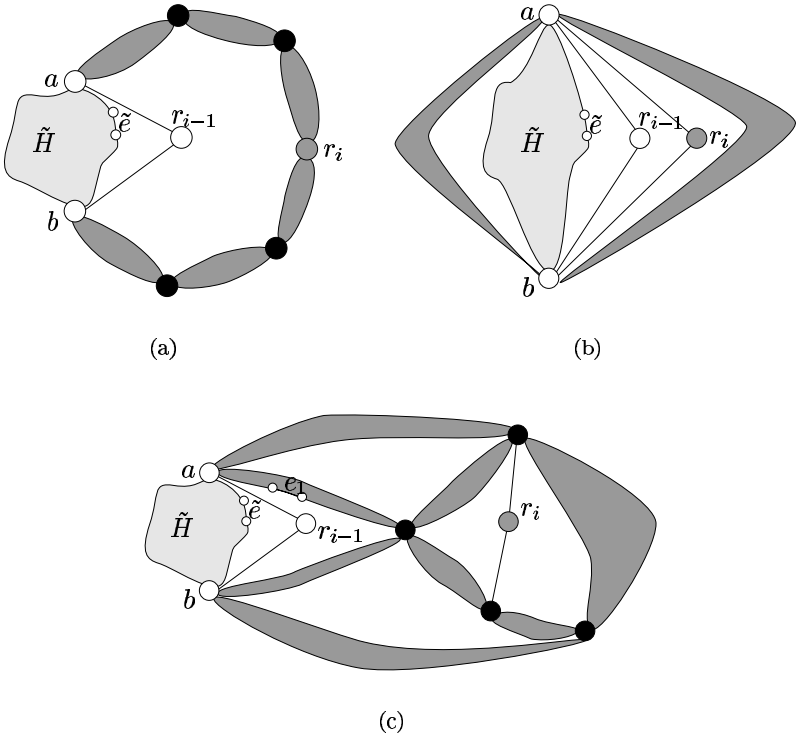

Fig. 8. The different types for node μ_i. $\tilde{H}$ denotes the graph $H_{i-1}\backslash\{r_{i-1}\}$ and $\tilde{e}$ denotes the last edge in $p_1 + \cdots + p_{i-1}$. μ is (a) an S-node, (b) a P-node, and (c) an R-node.

Π_i. We obtain the embedding Γ_i by inserting Γ_{i-1} into Π_i in such a way that r_{i-1} and e_1 (or r_{i-1} and r_i if p_i is empty) lie in a common face, see Figure 8(c). This is possible by mirroring Γ_i if necessary, since x_i^1 and e_1 (or x_i^1 and x_i^2 if p_i is empty) lie in a common face in Π_i.

Since $r_k = v_2$ and $H_k = G$, it follows that $p_1 + \cdots + p_k$ is an edge insertion path for v_1 and v_2 in G with respect to $\Pi := \Gamma_k$ and the lemma holds. □

Algorithm 1 computes only an edge insertion path $p = e_1, \ldots, e_\ell$ for the vertices v_1 and v_2 in G, but not the corresponding embedding of G. However, there is a simple way for finding an embedding Π such that p is an edge insertion path for v_1 and v_2 in G with respect to Π. Construct a graph G' by splitting each edge e_i in p introducing a new vertex w_i and insert new edges forming a path $v_1, w_1, \ldots, w_\ell, v_2$. Since p is an edge insertion path, the graph G' is planar and an embedding Π' for G' can be computed in linear time (see, e.g., [HT] and [MM]). Replacing all split edges in Π' by original edges (thus removing the vertices $w_1, \ldots, w_\ell$ and their adjacent edges again) results in an embedding Π for G such that p is an edge insertion path for v_1 and v_2 in G with respect to Π.

In order to prove the optimality of the edge insertion path p for v_1 and v_2 computed by the algorithm, we show that any edge insertion path for v_1 and v_2 is at least as long as p. It is sufficient to consider a shortest edge insertion path for an arbitrary, fixed embedding.

LEMMA 3. *Let Π' be an arbitrary embedding of G and let p' be a shortest edge insertion path for v_1 and v_2 in G with respect to Π'. Then $|p'| \geq |p|$ holds.*

PROOF. If the path $\Lambda = \mu_1, \ldots, \mu_k$ computed by the algorithm contains no R-node, p is empty and $|p'| \geq |p| = 0$ obviously holds. Assume now that Λ contains at least one R-node.

Let μ_i be an R-node in Λ. Denote with S_i the modified skeleton of μ_i constructed in the algorithm which contains the vertices x_i^1 and x_i^2 as representatives of v_1 and v_2, respectively. Let $G_i = (V_i, E_i \cup M_i)$ be the graph constructed in the algorithm such that E_i is the set of edges that results from expanding the unmarked edges. Since p' is a shortest edge insertion path for the embedding Π', the edges in p' that are also contained in E_i form a subsequence $p_i' = e_1', \ldots, e_{\ell_i}'$ of p' and p_i' is an edge insertion path for x_i^1 and x_i^2 in G_i with respect to the embedding of G_i induced by Π'. We will show that $|p_i'| \geq |p_i|$, where p_i is the subsequence of p computed by the algorithm.

For each unmarked edge e in S_i, set the costs of e to the traversing costs $c(e)$ of e and define the length of an edge insertion path to be the sum of the costs of the edges in the path. All marked edges are adjacent to either x_i^1 or x_i^2 and will not appear in an edge insertion path we consider. Each edge insertion path for x_i^1 and x_i^2 in G_i induces an edge insertion path for x_i^1 and x_i^2 in S_i which contains all the skeleton edges whose expansion graph is crossed in G_i.

Let $\tilde{p}_i$ be the edge insertion path in S_i induced by p_i, and let $\tilde{p}_i'$ be the edge insertion path in S_i induced by p_i'. Then

$$|p_i'| \geq |\tilde{p}_i'| = \sum_{e \in \tilde{p}_i'} c(e)$$

since crossing an expansion graph involves at least $c(e)$ edge crossings (see Lemma 1). Since p_i is a shortest edge insertion path for x_i^1 and x_i^2 in G_i with respect to Π_i, $\tilde{p}_i$ is a shortest edge insertion path for x_i^1 and x_i^2 in S_i, which implies that $|\tilde{p}_i'| \geq |\tilde{p}_i| = |p_i|$ and thus $|p_i'| \geq |p_i|$.

Let $I = \{i \mid \mu_i \text{ is an R-node}\}$. Since all E_i, $i \in I$, are pairwise disjoint, it follows that

$$|p'| \geq \sum_{i \in I} |p_i'| \geq \sum_{i \in I} |p_i| = |p|. \qquad \square$$

Lemmas 2 and 3 show that Algorithm 1 computes an optimal edge insertion path for v_1 and v_2 in $G = (V, E)$. It remains to prove that its running time is linear in the size of G.

The SPQR-tree $\mathcal{T}$ of G can be computed in time $\mathcal{O}(|V| + |E|)$ (see [GM1]). A path between two arbitrary allocation nodes of v_1 and v_2 can be found by inspecting

each skeleton graph and using depth first search in the tree $\mathcal{T}$. The shortest path $\Lambda = \mu_1, \ldots, \mu_k$ is obtained from this path by removing nodes from the start and the end of the path until it contains exactly one allocation node of v_1 and one allocation node of v_2. Hence, finding path Λ takes time $\mathcal{O}(|V| + |E|)$, since the size of $\mathcal{T}$ including all skeleton graphs is linear in the size of G.

The construction of the modified skeleton S_i which results from possibly splitting at most two edges in $skeleton(\mu_i)$ takes time linear in the size of $skeleton(\mu_i)$. Since the total size of all skeleton graphs in $\mathcal{T}$ is linear in the size of G, the total time for constructing $S_1, \ldots, S_k$ is $\mathcal{O}(|V| + |E|)$.

Finally, consider graph $G_i = (V_i, E_i \cup M_i)$ for $1 \leq i \leq k$, where E_i is the set of edges that results from expanding the unmarked edges in S_i and M_i is the set of marked edges in S_i. Since $|M_i| \leq 4$ (and $|V_i| \leq |E_i|$), an arbitrary embedding of G_i is computed in time $\mathcal{O}(|E_i|)$ (see [MM]), and the size of the augmented dual graph A_i is $\mathcal{O}(|E_i|)$. Hence, a shortest path between x_i^1 and x_i^2 in A_i can be found in time $\mathcal{O}(|E_i|)$ using breadth first search. Since all the sets E_i are pairwise disjoint, the total time for constructing $G_1, \ldots, G_k$ and for finding $p_1, \ldots, p_k$ is $\sum_{i=1}^{k} \mathcal{O}(|E_i|) = \mathcal{O}(|E|)$. Thus, the following theorem holds:

THEOREM 1. *Let $G = (V, E)$ be a biconnected planar graph and let v_1 and v_2 be two non-adjacent vertices in V. Then Algorithm* 1 *computes an optimal edge insertion path for v_1 and v_2 in G in time $\mathcal{O}(|V| + |E|)$.*

4. Inserting an Edge into a Connected Graph. We first introduce two definitions. Let G be a connected graph. The *block-vertex tree* $\mathcal{B}$ of G represents the relationships between the biconnected components (blocks) of G. It contains a B-node for each block of G and a V-node for each vertex of G. A V-node v and a B-node B are connected by an edge in $\mathcal{B}$ if vertex v is contained in block B. The *representative* of a vertex $v \in G$ in a block B is either v itself if $v \in B$, or the first cut-vertex c on the unique path from B to v in $\mathcal{B}$.

The algorithm for computing an optimal edge insertion path for a connected planar graph G and two non-adjacent vertices v_1 and v_2 is given in Algorithm 2. The algorithm constructs the block-vertex tree $\mathcal{B}$ of G and considers only the blocks on the path from v_1 to v_2 in $\mathcal{B}$. For each block B_i, an optimal edge insertion path p_i for the representatives of v_1 and v_2 in B_i is computed using Algorithm 1, and these paths are then concatenated. The following lemma shows that the resulting path $p_1 + \cdots + p_k$ is indeed an optimal edge insertion path for v_1 and v_2 in G.

LEMMA 4. *Let $p_1 + \cdots + p_k$ be the path computed by Algorithm* 2. *Then there exists an embedding Π of G such that $p_1 + \cdots + p_k$ is an optimal edge insertion path for v_1 and v_2 in G with respect to Π.*

PROOF. Let H_i be the union of the blocks B_1 to B_i. We show by induction that there is an embedding Γ_i of H_i such that $\Lambda_i := p_1 + \cdots + p_i$ is an optimal edge insertion path in H_i for v_1 and y_i.

Algorithm 2. Computes an optimal edge insertion path for a pair of non-adjacent vertices v_1, v_2 in a connected graph G.

```
OptimalInserter (graph G, vertex v1, vertex v2)
    Compute the block-vertex tree B of G;
    Find the path v1, B1, c1, ..., Bk−1, ck−1, Bk, v2 from v1 to v2 in B;
    for i = 1, ..., k do
        Let xi and yi be the representatives of v1 and v2 in Bi;
        pi := OptimalBlockInserter(Bi, xi, yi);
    end
    return p1 + ··· + pk;
end
```

$i = 1$. In this case, H_1 equals B_1 and, by Theorem 1, there is an embedding Γ_1 such that $\Lambda_1 = p_1$ is an optimal edge insertion path for $x_1 = v_1$ and y_1 in H_1 with respect to Γ_1.

$i > 1$. Assume now that $\Gamma_1, \ldots, \Gamma_{i-1}$ are already constructed such that Λ_{i-1} is an optimal edge insertion path for v_1 and y_{i-1} in H_{i-1} with respect to Γ_{i-1}.

By Theorem 1, there exists an embedding Π_i of B_i such that p_i as constructed in the algorithm is an optimal edge insertion path for x_i and y_i in B_i with respect to Π_i. Since y_{i-1} and x_i denote the same vertex in G, the embedding Γ_i of H_i can be constructed as follows. Since Λ_{i-1} is an edge insertion path for v_1 and y_{i-1}, there is face $f \in \Gamma_{i-1}$ that contains y_{i-1} and either v_1 if Λ_{i-1} is empty, or the last edge in Λ_{i-1}. Analogously, there is a face $f' \in \Pi_i$ that contains x_i and either y_i if p_i is empty, or the first edge in p_i. The embedding Γ_i is constructed by choosing f as the external face of Γ_{i-1} and placing this planar embedding of H_{i-1} into face f' of Π_i. This is possible, since $B_1 \cup \cdots \cup B_{i-1}$ and B_i have only the vertex $y_{i-1} = x_i$ in common. Thus, $\Lambda_i = p_1 + \cdots + p_i$ is an edge insertion path for v_1 and y_i in H_i with respect to Γ_i.

It remains to show the optimality of Λ_i. Let $\hat{p}$ be an arbitrary edge insertion path for v_1 and y_i in H_{i-1} with respect to some embedding $\hat{\Gamma}$. Obviously, $\hat{p}$ can be partitioned into two subpaths $\hat{p}_A$ and $\hat{p}_B$ such that $\hat{p}_B$ contains only the edges in B_i. Then $\hat{p}_B$ is an edge insertion path for x_i and y_i in B_i with respect to the embedding of B_i induced by $\hat{\Gamma}$, and $\hat{p}_A$ is an edge insertion path for v_1 and y_{i-1} in H_{i-1} with respect to the embedding of H_{i-1} induced by $\hat{\Gamma}$. Since $|p_i| \leq |\hat{p}_B|$ by Theorem 1 and $|\Lambda_{i-1}| \leq |\hat{p}_A|$ by the induction hypothesis, it follows that $|\Lambda_i| \leq |\hat{p}|$.

Since a block shares only a single vertex with the rest of the graph, it is easy to see that Λ_k is still an edge insertion path for v_1 and $y_k = v_2$ in G with respect to an embedding Π that results from inserting the remaining blocks not contained in $B_1, \ldots, B_k$ arbitrarily into Γ_k.

The optimality of Λ_k in G can be shown using a similar argument as in the induction step. Let $\hat{p}$ be an arbitrary edge insertion path for v_1 and v_2 in G. The subpath $\hat{p}_B$ of $\hat{p}$ which contains only the edges in H_k is an edge insertion path for v_1 and y_k in H_k. Thus, $|\hat{p}| \geq |\hat{p}_B| \geq |\Lambda_k|$. □

The block-vertex tree of $G = (V, E)$ can be computed in time $\mathcal{O}(|V| + |E|)$ by finding the blocks of G (see, e.g., [H]). Since all blocks are pairwise edge-disjoint, the size of $\mathcal{B}$ is $\mathcal{O}(|V| + |E|)$ and the path from v_1 to v_2 in $\mathcal{B}$ is found in time $\mathcal{O}(|V| + |E|)$ using depth first search. Algorithm 1 is called for each block $B_i = (V_i, E_i)$ which takes time $\mathcal{O}(|V_i| + |E_i|)$ according to Theorem 1. Since all blocks are pairwise edge-disjoint, Algorithm 2 takes time $\mathcal{O}(|V| + |E|)$.

The algorithm presented in this section can easily be generalized to arbitrary planar graphs. If v_1 and v_2 belong to the same connected component, simply apply Algorithm 2. Otherwise, the graph $G \cup \{(v_1, v_2)\}$ is obviously planar and the empty path is the optimal edge insertion path. Hence, we get the following result.

THEOREM 2. *Let $G = (V, E)$ be a planar graph and let v_1 and v_2 be two non-adjacent vertices in V. Then there exists an algorithm that computes an optimal edge insertion path for v_1 and v_2 in G in time $\mathcal{O}(|V| + |E|)$.*

5. Computational Experiments. We have implemented the algorithm presented in this paper within the graph drawing library AGD [AGD]. AGD contains a state-of-the-art framework of the planarization method as described in the Introduction and a linear time implementation of SPQR-trees as presented in [GM1].

We compare the standard edge insertion technique which uses an arbitrary fixed embedding with the method presented in this paper. The edge insertion procedure is applied iteratively for all edges not contained in the planar subgraph and the total number of crossings is compared.

We use the 8249 non-planar graphs from a benchmark set collected by Di Battista et al. [BGL$^+$] ranging from 11 to 100 vertices to test the implementations. The planar subgraph is computed using the AGD implementation of the heuristics described in [JLM]. Since all edges not contained in the subgraph are inserted successively in the second step of the planarization method, the impact of the new approach on the total number of crossings is not obvious.

Figure 9 shows the relative improvement achieved by the new edge insertion technique compared with the standard method. The relative improvement is defined as follows. Let c_s be the number of crossings produced by the standard method, and let c_n be the number of crossings produced using the new technique. Then the relative improvement is $(c_s - c_n)/c_s$. The chart shows the average value for all graphs with the same number of vertices, ranging from 11 to 100. For graphs up to 40 vertices, the improvement shows a large variance since there are only few small graphs in the benchmark set. For larger graphs, the relative improvement is about 15%.

For 68% out of the 8249 tested non-planar graphs, the number of crossings achieved by the new method was smaller and for only 8% was it larger than in the standard approach. The average relative improvement was 14.42% in total. The maximum improvement of 85.71% was for a graph with 39 vertices and 56 edges. The standard method produced 7 crossings whereas the new edge insertion technique led to only 1 crossing. The average number of crossings produced by the standard method grows from 1.29 for graphs with 11 vertices to 57.86 for graphs with 100 vertices, whereas the average number of crossings produced by the new approach grows from 1.29 to only 49.83.

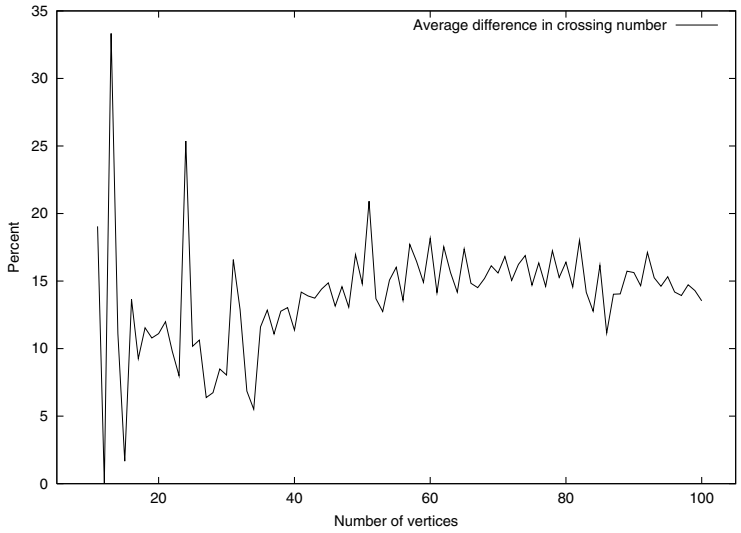

Fig. 9. Relative improvement of the new method compared with the standard approach.

A comprehensive experimental study of crossing minimization heuristics applying the planarization approach is presented in [GM2]. The authors study various techniques for computing a planar subgraph and edge insertion, including the standard and the new edge insertion method with postprocessing and randomization strategies. The paper concludes that the edge insertion method applied has a significant impact on the final number of crossings even if additional strategies like postprocessing and randomization are used. Thus, the optimal edge insertion method presented in this paper is a valuable contribution for solving the crossing minimization problem heuristically.

6. Crossing Number and Edge Insertion. The edge insertion problem can also be stated as follows: given a planar graph $G = (V, E)$ and a pair of vertices (v_1, v_2) in V, find a drawing of $G' = (V, E \cup \{(v_1, v_2)\})$ that has the minimum number of crossings among all drawings of G' in which every crossing is a crossing between an edge in E and the edge (v_1, v_2). We show in this section that such a drawing of G' is not necessarily crossing minimal. In particular, we give a class of graphs G_m such that a solution to the edge insertion problem for G_m and (v_1, v_2) results in a drawing with m crossings, whereas a crossing minimal drawing of $G'_m = G_m \cup \{(v_1, v_2)\}$ has only two crossings.

First, we define a wall graph as follows. A *wall* with width k consists of the vertices $x, y, z_1, \ldots, z_k$, the edges (z_i, z_{i+1}) for $1 \leq i < k$, and the edges (x, z_i) and (y, z_i) for $1 \leq i \leq k$ (see Figure 10). The vertices x and y are called the *poles* of the wall. A wall with width ≥ 3 is a triconnected planar graph.

For an even number $m \geq 2$, the graph G_m is constructed in the following way (see Figure 11). We start with a ring of walls $W_1, \ldots, W_6$ with width $m + 1$, where the poles of adjacent walls in the ring are identified. We denote the pole vertices with $w_1, \ldots, w_6$ such that the poles of W_1 are w_1 and w_2, and so forth. For each wall W_j, the other two

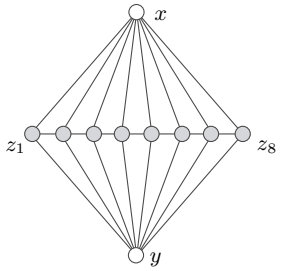

Fig. 10. A wall with width 8.

vertices on the boundary are denoted with u_j^{i} and u_j^e, where u_j^{i} is inside the ring and u_j^{e} is on the external face (see Figure 11). Moreover, the edges $e_1 = (u_1^{\mathrm{e}}, w_3)$, $e_2 = (u_6^{\mathrm{e}}, w_5)$, $e_3 = (u_2^{\mathrm{i}}, u_3^{\mathrm{i}})$, $e_4 = (u_5^{\mathrm{i}}, u_4^{\mathrm{i}})$ are added, $m/2$ vertices are inserted by splitting edge (u_3^{i}, w_4) and $m/2$ vertices are inserted by splitting edge (w_4, u_4^{i}), and every created split vertex is connected with vertex w_1 by an edge h_j, $1 \leq j \leq m$. The two vertices to be connected are $v_1 := u_1^{\mathrm{i}}$ and $v_2 := u_6^{\mathrm{i}}$, i.e., $G'_m = G_m \cup \{(u_1^{\mathrm{i}}, u_6^{\mathrm{i}})\}$.

By construction, G_m is triconnected and planar. In particular, G_m has only two embeddings which are mirror-images of each other. It is easy to see that an optimal edge insertion path for v_1 and v_2 has length m (by crossing the edges $h_1, \ldots, h_m$), since passing through a wall would require at least $m + 1$ crossings. On the other hand, there is a drawing of G'_m with only two crossings as shown in Figure 12. Here, only the two crossings e_1 with e_3 and e_2 with e_4 occur, independent of the choice of m.

Acknowledgements. We thank Graham Farr for finding the example graphs G_m we used in Section 6.

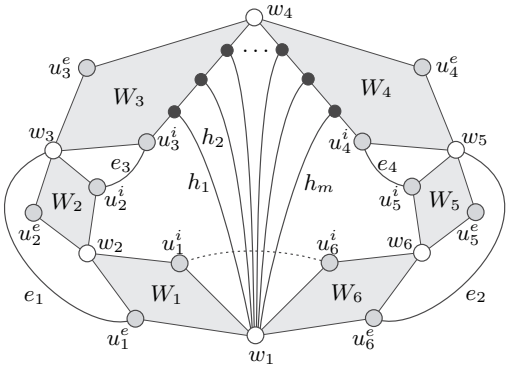

Fig. 11. The graph G_m; each shaded region represents a wall with width $m + 1$. The dashed edge $(u_1^{\mathrm{i}}, u_6^{\mathrm{i}})$ is the edge to be inserted.

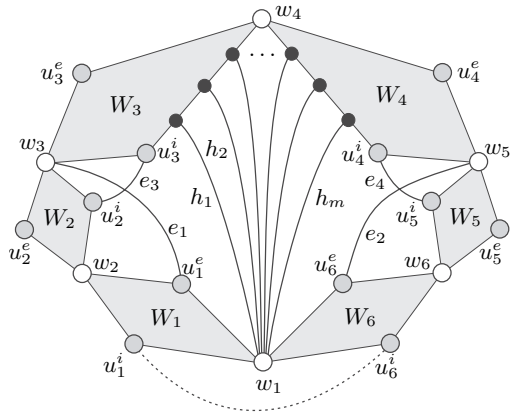

Fig. 12. A drawing of the graph G'_m with only two crossings.

References

[AGD] *AGD User Manual* (*Version* 1.1). Universität Wien, Max-Planck-Institut Saarbrücken, Universität Trier, Universität zu Köln, 1999. See also http://www.mpi-sb.mpg.de/AGD/ .

[BBD] P. Bertolazzi, G. Di Battista, and W. Didimo. Computing orthogonal drawings with the minimum number of bends. *IEEE Transactions on Computers*, 49(8):826–840, 2000.

[BGL+] G. Di Battista, A. Garg, G. Liotta, R. Tamassia, E. Tassinari, and F. Vargiu. An experimental comparison of four graph drawing algorithms. *Computational Geometry Theory and Applications*, 7:303–326, 1997.

[BT] G. Di Battista and R. Tamassia. On-line planarity testing. *SIAM Journal on Computing*, 25(5):956–997, 1996.

[CNAO] N. Chiba, T. Nishizeki, S. Abe, and T. Ozawa. A linear algorithm for embedding planar graphs using PQ-trees. *Journal of Computer and System Sciences*, 30(1):54–76, 1985.

[FOO] D. Fränken, J. Ochs, and K. Ochs. Generation of wave digital structures for connection networks containing ideal transformers. In *Proceedings of the IEEE International Symposium on Circuits and Systems* (*ISCAS*), volume 3, pages 240–243, 2003.

[GM1] C. Gutwenger and P. Mutzel. A linear time implementation of SPQR-trees. In J. Marks, editor, *Graph Drawing* (*Proc*. 2000), pages 77–90. Volume 1984 of LNCS. Springer-Verlag, Berlin, 2001.

[GM2] C. Gutwenger and P. Mutzel. An experimental study of crossing minimization heuristics. In G. Liotta, editor, *Proceedings of the* 11*th Symposium on Graph Drawing* 2003, pages 13–24. Volume 2912 of LNCS. Springer-Verlag, Berlin, 2004.

[GM3] C. Gutwenger and P. Mutzel. Graph embedding with minimum depth and maximum external face. In G. Liotta, editor, *Proceedings of the* 11*th Symposium on Graph Drawing* 2003, pages 259–272. Volume 2912 of LNCS. Springer-Verlag, Berlin, 2004.

[GMW] C. Gutwenger, P. Mutzel, and R. Weiskircher. Inserting an edge into a planar graph. In *Proceedings of the Twelth Annual ACM–SIAM Symposium on Discrete Algorithms*, pages 246–255, 2001.

[GT] A. Garg and R. Tamassia. On the computational complexity of upward and rectilinear planarity testing. In R. Tamassia and I. G. Tollis, editors, *Proceedings Graph Drawing* '94, pages 286–297. Volume 894 of LNCS. Springer-Verlag, Berlin, 1994.

[H] Frank Harary. *Graph Theory*. Addison-Wesley, Reading, MA, 1972, 3rd printing, 1994.

[HS] D. Harel and M. Sardas. Randomized graph drawing with heavy duty preprocessing. *Advanced Visual Interfaces*, pages 19–33, 1994.

[HT] J. Hopcroft and R.E. Tarjan. Efficient planarity testing. *Journal of the ACM*, 21:549–568, 1974.

[JLM] M. Jünger, S. Leipert, and P. Mutzel. A note on computing a maximal planar subgraph using PQ-trees. *IEEE Transactions on Computer-Aided Design*, 17(7):609–612, 1998.

[MM] K. Mehlhorn and P. Mutzel. On the embedding phase of the Hopcroft and Tarjan planarity testing algorithm. *Algorithmica*, 16(2):233–242, 1996.

[MW1] P. Mutzel and R. Weiskircher. Optimizing over all combinatorial embeddings of a planar graph. In G. Cornuéjols, R. Burkard, and G. Wöginger, editors, *Proceedings IPCO* '99, pages 361–376. Volume 1610 of LNCS. Springer-Verlag, Berlin, 1999.

[MW2] P. Mutzel and R. Weiskircher. Computing optimal embeddings for planar graphs. In D. Z. Du, P. Eades, V. Estivill-Castro, X. Lin, and A. Sharma, editors, *Proceedings COCOON* '00, pages 95–104. Volume 1858 of LNCS. Springer-Verlag, Berlin, 2000.

[T] R. Tamassia. On embedding a graph in the grid with the minimum number of bends. *SIAM Journal on Computing*, 16(3):421–444, 1987.