Graph Embedding with Minimum Depth and Maximum External Face

(Extended Abstract)

Carsten Gutwenger¹ and Petra Mutzel²

¹ Stiftung caesar

Ludwig-Erhard-Allee 2, D-53175 Bonn, Germany

² Vienna University of Technology

Favoritenstr. 9–11 E186, A-1040 Vienna, Austria

Abstract. We present new linear time algorithms using the SPQR-tree data structure for computing planar embeddings of planar graphs optimizing certain distance measures. Experience with orthogonal drawings generated by the topology-shape-metrics approach shows that planar embeddings following these distance measures lead to improved quality of the final drawing in terms of bends, edge length, and drawing area.

Given a planar graph, the algorithms compute the planar embedding with

1. the minimum depth among the set of all planar embeddings of G ,
2. the external face of maximum size among the set of all planar embeddings of G ,
3. the external face of maximum size among the set of all embeddings of G with minimum depth.

1 Introduction

A *combinatorial embedding* Γ of a planar graph G is defined as a clockwise ordering of the incident edges for every vertex with respect to a crossing-free drawing of G in the plane. An equivalent definition of a combinatorial embedding is an ordered list of the boundary edges for each face. A combinatorial embedding Γ together with a given *external face* $f \in \Gamma$ is called a *planar embedding* (Γ, f) .

A successful approach for generating orthogonal drawings of general graphs is the *topology-shape-metrics approach* [18,2,6]. Here, in a first step, the crossing structure of the graph is computed. The second step takes this topology as input and produces an orthogonal representation fixing the shape of the drawing. Finally, the third step determines the lengths of the horizontal and vertical edge segments. The goal of the three steps is to minimize the number of crossings, the number of bends, and the total edge length, respectively.

Algorithms for the second step typically deal with planar graphs (i.e., the planarized graphs arising in step 1). Such algorithms, in general, need as input a planar embedding (e.g., the bend minimization algorithm by Tamassia [17]). Fortunately, a planar embedding of a planar graph can be computed in linear

time (see [14,5]). However, the choice of the embedding typically has a big impact on the quality of the resulting drawing, and the number of possible embeddings of a planar graph may be exponential in the size of the graph. Since optimizing aesthetic criteria like the number of bends is NP-hard in general [10], finding criteria for planar embeddings that lead to good drawings in practice, as well as efficient algorithms to compute such planar embeddings is desired.

Several authors have studied the problem of computing planar embeddings which are optimal with respect to some distance measures. Bienstock and Monma [4] have suggested polynomial time algorithms for computing planar embeddings that minimize the distance measures *radius*, *width*, *outerplanarity*, and *face depth*. They also showed that it is NP-complete to test whether a planar graph has an embedding with dual diameter bounded above by an input number.

Exact algorithms for the NP-hard problem of computing planar embeddings that minimize the number of bends of orthogonal planar graphs have been suggested by Liotta et al. [13] and by Mutzel and Weiskircher [15]. Mutzel and Weiskircher [15] have suggested an exact algorithm for the NP-hard problem of computing a linear cost function on the face cycles. However, all the exact algorithms only work well for graphs with up to 80 edges.

In this paper we give linear time algorithms based on the SPQR-tree data structure for computing planar embeddings of planar graphs optimizing various distance measures. Experience in the graph drawing community has shown that planar embeddings following these distance measures lead to improved quality of the final drawing in terms of bends, edge length, and drawing area [1,3,9,16,20].

The first distance measure is the depth of a planar embedding introduced by Pizzonia and Tamassia [16]. They have suggested an algorithm for a restricted version of computing a minimum depth embedding of a planar graph in which the embeddings of the biconnected components are given and fixed. Our algorithm finds the minimum depth embedding over the class of all planar embeddings without any restriction.

The *depth* of a planar embedding is a measure of the topological nesting of the biconnected components of G in (Γ, f) . For the formal definition, we need to introduce some more terms. A *block* of G is a biconnected component of G . Given a connected graph, the *BC-tree* $\mathcal{B}$ of G has vertices for each block and each cut vertex of G . There is an edge between a block vertex b associated with a block B and a cut vertex v if $v \in B$. BC-trees are rooted at an arbitrary block of G , and the edges are directed from parent to child (e.g., an edge (v, w)); sometimes they will be considered as undirected (e.g., denoted as $\{v, w\}$). Given a planar embedding (Γ, f) of a planar graph $G = (V, E)$ with face set F , the *dual graph* $G^* = (V^*, E^*)$ is constructed as follows: $V^* = F$ and E^* contains an edge $\{f_i, f_j\}$ for each $e \in E$ such that e is on the boundary of both f_i and f_j . An *extended dual BC-tree* of a given planar embedding (Γ, f) is either the BC-tree of the dual graph G^* in the case that the edges on the external face do not belong to a single block in G ; then, we define r as the (cut) vertex $v_f \in V^*$ associated

Fig. 1. Two drawings of the same graph given in [16] (computed by the GDToolkit system [11]). The planar embedding in (a) has depth 5, while the one in (b) has depth 1.

with f in G . Otherwise, all edges bordering the external face belong to a unique block B in the dual BC-tree containing the external face vertex $v_f \in V^*$. Then, the *extended dual BC-tree* is the dual BC-tree extended by an additional vertex r and the edge (r, b) , where b is the block vertex associated with B . Now we are ready to introduce the formal definition of the *depth* of a planar embedding:

For a planar embedding (Γ, f) of a connected planar graph G , the *depth* is defined as the height of the extended dual BC-tree with respect to (Γ, f) rooted at r .

The example shown in Figure 1 has already been provided by Pizzonia and Tamassia in order to justify their approach for minimizing the depth. Each of the two drawings has bends and area optimized for its embedding (computed by the GDToolkit system [11]). The planar embedding in Figure 1(a) has depth 5, while the one in Figure 1(b) has depth 1. Obviously, the drawing in Figure 1(b) is much easier to read and to understand than the one in Figure 1(a).

Figure 2 shows two drawings of the same graph realizing different planar embeddings. Again, both drawings have the minimal number of bends with respect to their planar embeddings. The drawing in Figure 2(b) looks much more compact than the drawing in Figure 2(a). We observe that the graph is biconnected and hence the depth of any planar embedding is 1. On the other hand, the external face in Figure 2(b) is bordered by 9 edges and is much larger than in Figure 2(a) with only 3 edges.

Also the two embeddings in Figure 1 differ in the number of edges contained in the external face. Figure 1(a) has 3 edges bordering the external face, while the better drawing shown in Figure 1(b) has 15. This goes along with our observation that a higher number of edges on the external face leads to improved layout quality.

Hence, the second distance measure investigated in this paper is the length of the external face cycle (also called, the *size of a face*) in the planar embed-

Fig. 2. Two bend-minimal orthogonal drawings of a graph with different planar embeddings.

ding. Our algorithm computes the planar embedding with the external face of maximum size among the set of all possible planar embeddings.

The third distance measure considered in this paper is a combination of the first two distance measures: we search the planar embedding with the external face of maximum size among all planar embeddings with minimum depth. We conjecture that this measure provides, in general, the *best* planar embeddings among the considered distance measures in our paper, and in the literature, leading to improved planar layouts.

The rest of the paper is organized as follows. Section 2 provides the required graph-theoretical background. For technical reasons, we describe the algorithm for computing a planar embedding with maximum external face first (see Section 3), before we discuss the two remaining algorithms in Section 4. Because of space restrictions, we have omitted some of the proofs.

2 Preliminaries

SPQR-trees [8,7] basically represent the decomposition of a biconnected graph into its triconnected components. Let G be a 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 at least three split components $G_1, \dots, G_k$, the root of $\mathcal{T}$ is a P-node μ , whose skeleton consists of k parallel edges $e = e_1, \dots, e_k$ 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 $c_1, \dots, c_{k-1}$ ($k \geq 2$) that partition G into its blocks $G_1, \dots, G_k$, in this order from s to t , the root of $\mathcal{T}$ is an S-node μ , whose skeleton is the cycle $e_0, e_1, \dots, e_k$, where $e_0 = e$, $c_0 = s$, $c_k = t$, and $e_i = (c_{i-1}, c_i)$ ($i = 1, \dots, k$).

Rigid Case: If none of the above cases applies, let $\{s_1, t_1\}, \dots, \{s_k, t_k\}$ be the maximal split pairs of G with respect to $\{s, t\}$ ($k \geq 1$), and, for $i = 1, \dots, k$, 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 $e_i = (s_i, t_i)$.

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

Let e be an edge in $\text{skeleton}(\mu)$ and ν 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 $\text{expansion}(e)$) is the graph induced by the edges that are represented by the Q-nodes in $\mathcal{T}_\nu$. We further introduce the notation $\text{expansion}^+(e)$ for the graph $\text{expansion}(e) \cup e$.

Replacing a skeleton edge e by its expansion graph is called *expanding* e . The *pertinent graph* of a tree node μ results from expanding all edges in $\text{skeleton}(\mu)$ except for the reference edge of μ and is denoted with $\text{pertinent}(\mu)$. Hence, if e is a skeleton edge and ν its pertinent node, then $\text{expansion}^+(e)$ equals $\text{pertinent}(\nu)$. If v is a vertex in G , a node in $\mathcal{T}$ whose skeleton contains v is called an *allocation node* of v .

BC-trees and SPQR-trees can be constructed in linear time and use only linear space (see [19,12,8]). They are used to encode all embeddings of a planar connected graph G . Denote with $\mathcal{T}_B$ the SPQR-tree of a block $B \in G$. The embeddings of B are in one-to-one correspondence with the embeddings of the skeletons of $\mathcal{T}_B$. An embedding of B is basically constructed by replacing skeleton edges with their expansion graphs while preserving the embedding. Once all blocks are embedded, an embedding of G is constructed by several applications of the following procedure. Let Γ' be an embedding consisting of a connected union of some blocks. We want to insert a further embedded block Γ_B with vertex

Fig. 3. An embedding of $\text{expansion}^+(e)$ with given vertex lengths. Assuming that e, v, w have length 0, face f_1 has size 8 and f_2 has size 9, therefore edge e has component length ≥ 9 .

$c \in B \cap \Gamma'$ into Γ' . Then, we have to choose a face containing c as external face of Γ_B and insert Γ_B into a face of Γ' containing c .

3 Planar Embeddings with Maximum External Face

Let $G = (V, E)$ be a planar connected graph without self-loops. We first consider biconnected graphs and present linear-time algorithms for finding an embedding with maximum external face and for computing the size of a maximum external face containing a prescribed vertex v for every vertex $v \in V$.

3.1 Biconnected Graphs

Let $B = (V_B, E_B)$ be a block of G and $\mathcal{T}_B$ its SPQR-tree. We associate a non-negative length with each vertex $v \in B$ and each skeleton edge in $\mathcal{T}_B$. The length of an edge in B is simply 1. The size of a face f is defined as $\sum_{e \in f} \text{length}(e) + \sum_{v \in f} \text{length}(v)$.

Consider an edge $e = (v, w)$ in a skeleton graph and let Γ_e be an embedding of $\text{expansion}^+(e)$ such that Γ_e has a face f^* containing e of maximum size among all embeddings of $\text{expansion}^+(e)$. We call such an embedding an embedding of $\text{expansion}^+(e)$ with *maximum length* and define the *component length* of e to be $\text{size}(f^*) - \text{length}(e) - \text{length}(v) - \text{length}(w)$ (compare also Figure 3).

The general idea of the algorithm is as follows: Let S be a skeleton for which we have chosen an embedding Γ_S . In order to extend Γ_S to an embedding Γ of B , we have to choose embeddings of the graphs $\text{expansion}^+(e)$ for all edges $e \in S$. Each face f_S in Γ_S corresponds to a face f_Γ in Γ in which each skeleton edge $e \in f_S$ is replaced by a path p_e on the external face of its expansion graph (see Figure 4). We call this *expanding face* f_S to face f_Γ . Vice versa, for each face f'_Γ in Γ , we can find a face in a skeleton that can be expanded to f'_Γ . Face f_Γ is made as large as possible by embedding each expansion graph of an edge in f_S , such that the *length* of the path p_e , which we define as the number of edges

Fig. 4. A face f_s in the skeleton of an R-node (left) and the resulting face f_r after expanding all edges in f_s (right).

in p_e plus the lengths of the interior vertices on p_e , is the component length of e . We get the following Lemma:

Lemma 1. *Let B be a planar biconnected graph with vertex lengths and $\mathcal{T}_B$ its SPQR-tree. Then the following holds:*

- (i) *Each skeleton face f can be expanded to a face f' such that the size of f' is the sum of the lengths of the vertices in f plus the component lengths of all edges in f .*
- (ii) *There exists an internal node $\mu \in \mathcal{T}_B$ and an embedding Γ_μ of $\text{skeleton}(\mu)$ such that there is a face f in Γ_μ that can be expanded to a maximum face f^* of B . All edges in f are expanded to expansion graphs with maximum length.*

We can compute the component length of all non-reference skeleton edges by a bottom-up traversal of $\mathcal{T}_B$. Let e be an edge in $\text{skeleton}(\mu)$ and ν the pertinent node of e . Denote with e_r the reference edge of ν and with ℓ the sum of the length of the two poles of μ . If we assume that the length of all edges in $\text{skeleton}(\nu)$ except for e_r are set to their component length, the component length of e is

Q-node: 1

S-node: size of a face in $\text{skeleton}(\nu)$ minus ℓ

P-node: length of the longest edge different from e_r in $\text{skeleton}(\nu)$

R-node: size of the largest face containing e_r in $\text{skeleton}(\nu)$ minus ℓ

Once we have set the length of all non-reference skeleton edges to their component lengths, we can compute the component length of the reference edges by a top-down traversal of $\mathcal{T}_B$. Since $\mathcal{T}_B$ is rooted at a Q-node, the component length of the reference edge of its only son is 1. Consider now a node $\mu \in \mathcal{T}_B$ and let S be the skeleton of μ . We assume that the length of each edge in S is set to its component length. We want to compute the component length of the reference edge of each child ν of μ . We distinguish three cases according to the type of μ :

S-node: Let L be the sum of the length of all vertices and edges in S . The component length of the reference edge of ν is L minus the length of $e_{S,\nu}$ minus the lengths of the two vertices incident to $e_{S,\nu}$.

P-node: The component length of the reference edge of ν is the length of the longest edge in S different from $e_{S,\nu}$.

R-node: Let f be the largest face in S containing $e_{S,\nu}$. The component length of the reference edge of ν is the size of f minus the length of $e_{S,\nu}$ minus the lengths of the two vertices incident to $e_{S,\nu}$.

According to Lemma 1, we need to find an embedding Γ_μ of a skeleton with a face f of maximum size among all possible embeddings of skeletons. This can be achieved by inspecting all skeletons S . If S is the skeleton of an R-node, S has only two embeddings which are a mirror of each other. The largest face we can produce is simply a largest face in any embedding of S . If S is the skeleton of a P-node, say a bundle of parallel edges $e_1, \dots, e_k$, we can form any face consisting of two edges e_i and e_j with $i \neq j$. Hence, the largest face we can produce consists of the two longest edges in S . If S is the skeleton of an S-node, S has only a single embedding consisting of two equally sized faces.

This shows that we can find a tree node μ and produce a skeleton face $f \in \Gamma_\mu$ that can be expanded to a maximum face of B . According to Lemma 1, all edges in f have to be expanded to expansion graphs with maximum length, which can be achieved by recursively traversing the tree nodes involved.

Theorem 1. *Let $B = (V_B, E_B)$ be a biconnected planar graph. Then, there exists an $O(|V_B| + |E_B|)$ -time algorithm that computes a planar embedding of B with maximum external face.*

In order to compute the size of a maximum external face containing a prescribed vertex v , we have to consider all allocation nodes of v in $\mathcal{T}_B$. Suppose we have precomputed the size of all skeleton faces in S- and R-nodes, as well as the length of the two longest edges within a P-node. Then, we can compute the size of a maximum face containing v very efficiently, i.e. in time $O(n_v + m_v)$, where n_v is the number of allocation nodes of v in $\mathcal{T}_B$ and m_v is the number of skeleton edges in R-nodes incident to representatives of v . Since the size of $\mathcal{T}_B$ including all skeleton graphs is linear in the size of B , we know that $\sum_{v \in B} (n_v + m_v)$ is linear in the size of B and we get the following lemma.

Lemma 2. *There exists an $O(|V_B| + |E_B|)$ -time algorithm that computes the size of a maximum face containing v for every vertex $v \in V_B$.*

3.2 Connected Graphs

Let $\mathcal{B}$ be the BC-tree of G . Consider a block B and a cut-vertex c in G with $c \in B$. If we delete edge $\{c, B\}$ in $\mathcal{B}$, $\mathcal{B}$ is split into two connected components, one component $\mathcal{B}_B$ containing B . We denote the graph induced by the edges in all blocks contained in $\mathcal{B}_B$ with $G_{c,B}$.

The idea of the algorithm is as follows (compare Figure 5): Suppose we have fixed an embedding Γ_0 of a block B_0 . In order to extend Γ_0 to an embedding of G , we have to find an embedding $\Gamma_{c,B}$ with c on its external face for each graph $G_{c,B}$ with $c \in B$, $B \neq B_0$, and place $\Gamma_{c,B}$ into an adjacent face of c in

Fig. 5. A fixed embedding Γ_0 of B_0 (thick edges), where the graphs G_{c_1, B_1} , G_{c_1, B_2} and G_{c_2, B_5} have been placed into face f_0 .

Γ_0 . For a fixed face f_0 in Γ_0 , we can enlarge f_0 as much as possible if we place all $\Gamma_{c, B}$ with $c \in f_0$ into f_0 . We denote with $\text{smf}_B(c)$ the sum of the sizes of the maximum faces containing c of all $G_{c, B'}$ with $c \in B'$, $B' \neq B$. If c is not a cut-vertex in G , then $\text{smf}_B(c)$ is 0. We will use the value $\text{smf}_B(c)$ as the vertex length of c in B_0 and apply the procedures developed in the previous section. The following lemma states the crucial results on which the correctness of our algorithm is based.

- Lemma 3.** (i) Let B be a block of G and define the length of a vertex v in B as $\text{smf}_B(v)$. If f_B is a face in an arbitrary embedding Γ_B of B , then there exists an embedding Γ of G with a face f , such that the size of f (disregarding vertex lengths) equals the size of f_B . Face f results from embedding each $G_{c, B'}$ with $c \in B$, $B' \neq B$ with a maximum external face containing c and placing it into f_B .
- (ii) Let f be a face in an embedding Γ of G . Then, there exists a block B of G , a length ℓ_v for each vertex $v \in B$ with $\ell_v \leq \text{smf}_B(v)$, and an embedding Γ_B of B with a face f_B such that the size of f (disregarding vertex lengths) equals the size of f_B .

Theorem 2. For each block B of G , let $\text{smf}_B(v)$ be the length of a vertex $v \in B$. Let $B_{\max}$ be the block having the embedding $\Gamma_{\max}$ with the largest maximum face $f_{\max}$ among all blocks. Then, we can extend $\Gamma_{\max}$ to a planar embedding of G with maximum external face f . Face f results from embedding each $G_{c, B'}$ with $c \in B_{\max}$ and $B' \neq B_{\max}$ with a maximum external face containing c and placing it into $f_{\max}$.

Proof. According to Lemma 3(i), we will find face f with the required size in an embedding Γ of G . Lemma 3(ii) shows that a maximum face in G cannot be larger than f , hence Γ is an embedding of G with a face f of maximum size. $\square$

Algorithm 1: Planar embedding with maximum external face.

```

MaximumFace (graph  $G$ )
  Compute BC-tree  $\mathcal{B}$  of  $G$  (rooted at any block) and SPQR-tree  $\mathcal{T}_{\mathcal{B}}$  for each block  $B \in \mathcal{B}$ .
  let  $B_r$  be the root block of  $\mathcal{B}$ 
  forall  $v \in B_r$  do
     $length_{B_r}(v) := \sum_{(v,B) \in \mathcal{B}} \text{ConstraintMaxFace}(B, v)$ 
  od
   $(B, \ell) := \text{MaximumFaceRec}(B_r)$ 
  Embed  $G$  by expanding a maximum face in  $B$ 
end

ConstraintMaxFace (block  $B$ , vertex  $c$ )
  forall  $v \in B, v \neq c$  do
     $length_B(v) := \sum_{(v,B') \in \mathcal{B}} \text{ConstraintMaxFace}(B', v)$ 
  od
   $length_B(c) := 0$ 
   $ctrLength(B, c) := \text{size of a maximum face in } B \text{ containing } c$ 
  return  $ctrLength(B, c)$ 
end

MaximumFaceRec (block  $B$ )
   $(B^*, \ell^*) := (B, \text{size of a maximum face in } B)$ 
  forall  $(B, c) \in \mathcal{B}$  with  $ex. (c, B') \in \mathcal{B}$  do
     $ctrLength(B, c) := \text{size of a maximum face in } B \text{ containing } c$ 
     $L := \sum_{\{B', c\} \in \mathcal{B}} ctrLength(B', c)$ 
    forall  $c \rightarrow B' \in \mathcal{B}$  do
       $length_{B'}(c) := L - ctrLength(B', c)$ 
       $\ell' := \text{MaximumFaceRec}(B')$ 
      if  $\ell' > \ell^*$  then  $(B^*, \ell^*) := (B', \ell')$ 
    od
  od
  return  $(B^*, \ell^*)$ 
end

```

Algorithm 3.2 shows how to compute a planar embedding with maximum external face. Variable $length_B(c)$ stores the length of vertex c in block B , and $ctrLength(B, c)$ is set to the size of a maximum face of $G_{c,B}$ containing c . Function $\text{ConstraintMaxFace}(B, c)$ computes a maximum face in $G_{c,B}$ containing c and is called for each edge $(c, B) \in \mathcal{B}$. Function MaximumFaceRec recursively traverses $\mathcal{B}$ from top to bottom. When MaximumFaceRec is called for block B , the length of each vertex $v \in B$ is already set to $smf_B(v)$. Since the function finally returns the block B^* with the largest maximum face of size ℓ^* , this is the block B_{max} in Theorem 2. For embedding each $G_{c,B}$ with $c \in B^*$ and $B \neq B^*$ with a maximum external face containing c , $\mathcal{B}$ has to be recursively traversed starting at block B^* .

Theorem 3. *Let $G = (V, E)$ be a planar connected graph. $\text{MaximumFace}(G)$ computes a planar embedding of G with maximum external face in time $O(|V| + |E|)$.*

All algorithms presented in this section can easily be generalized to graphs with predefined non-negative edge lengths, in particular to edges with length 0.

Fig. 6. Two different extensions of an embedding Γ_0 of a block B_0 with their corresponding block-cutface trees. The left extension has depth 3 whereas the right one has depth 5.

4 Planar Embeddings with Minimum Depth

Let $G = (V, E)$ be a planar graph. Consider a block B with planar embedding Γ_B . An *extension* of Γ_B denotes a planar embedding of G which results from embedding all graphs $G_{c,B'}$ with $c \in B$ and $B' \neq B$ so that c lies on the external face and placing them into a face of Γ_B .

Let $m_{c,B'}$ be the minimum depth of a planar embedding of $G_{c,B'}$ with c on the external face. We define further

$$\begin{aligned} m_B(c) &:= \max\{0\} \cup \{m_{c,B'} \mid c \in B', B' \neq B\} \\ m_B &:= \max_{c \in B} m_B(c) \\ M_B &:= \{c \in B \mid m_B(c) = m_B\} \end{aligned}$$

Then, the minimum depth of an extension of an embedding of B is m_B if there is an embedding of B in which all vertices in M_B lie in a common face, and $m_B + 2$ otherwise (compare Figure 6). The following lemma shows the relationship between minimizing the depth and maximizing the external face.

Lemma 4. *Set the length of all edges in B to 0 and the length of a vertex v in B to 1 if $v \in M_B$, and 0 otherwise. Let (Γ^*, f^*) be a planar embedding of B with maximum external face f^* . Then, the minimum depth of an extension of an embedding of B is*

$$m_B \quad \text{if } \text{size}(f^*) = |M_B| \\ m_B + 2 \quad \text{otherwise}$$

The problem of finding a planar embedding of a graph $G_{c,B}$ with minimum depth such that c is on the external face can be tackled the same way. Based on this result, algorithm **MinimumDepth** proceeds similar to algorithm **MaximumFace** for connected graphs.

First, the values $m_{c,B}$ for all edges (c, B) in the BC-tree $\mathcal{B}$ are computed by a bottom-up traversal of $\mathcal{B}$. The length of each vertex $v \neq c$ in B is set according to Lemma 4 and a maximum face in B is computed that contains c .

Afterwards, $\mathcal{B}$ is traversed from top to bottom. When a block $B = (V_B, E_B)$ is visited, the values $m_{c,B'}$ are already computed for each cut-vertex c in G that is contained in B . According to Lemma 4, we set the vertex lengths in B and compute the minimum depth of an extension of B by finding a maximum face in B . Before we proceed with the descendants of B in tree $\mathcal{B}$, we have to compute the values $m_{c,B}$ for all edges $(B, c) \in \mathcal{B}$. We distinguish two cases:

$M_B = \{c_1, \dots, c_k\}$ with $k \geq 2$: We compute a maximum face in block B containing c using the vertex lengths we have already assigned.

$M_B = \{c\}$: In this case, $m_2 = \max_{v \in V_B, v \neq c} m_B(v)$ is less than m_B and we cannot reuse the vertex lengths. However, this case can occur at most once for a block which allows us to invest $O(|B|)$ running time. We calculate new vertex lengths according to $M_2 = \{c \in V_B \setminus \{v\} \mid m_B(c) = m_2\}$ and find a maximum face in B containing c using these vertex lengths.

Since **MaximumFace** and **MinimumDepth** proceed in a similar fashion, it is possible to combine both algorithms to a new algorithm **MinDepthMaxFace** that computes a planar embedding with maximum external face among all planar embeddings with minimum depths. This can be achieved by using a pair (d, ℓ) as length attribute, where the first component d refers to the 0 – 1 length attribute used in **MinimumDepth** and ℓ refers to the length attribute used in **MaximumFace**. The linear order on these tuples is defined componentwise, i.e.

$$(d, \ell) > (d', \ell') \iff d > d' \text{ or } (d = d' \text{ and } \ell > \ell')$$

Each time a maximum face in a block has to be computed, we first determine a maximum face according to the linear order defined above. If the resulting maximum face has size (d^*, ℓ^*) and d^* is the number of cut-vertices we want to place in a common face (e.g. $d^* = |M_B|$ as in Lemma 4), we also have found a planar embedding with a maximum face among all planar embeddings with minimum depth. Otherwise, the value d^* is irrelevant since all extensions will have the same depth and there might be planar embeddings with a larger external face. Therefore, we call the procedure for finding a maximum face in a block again, this time respecting only the second component ℓ of the length attribute.

Theorem 4. *Let $G = (V, E)$ be a planar connected graph.*

- i) *Algorithm MinimumDepth(G)* computes a planar embedding of G with minimum depth.
- ii) *Algorithm MinDepthMaxFace(G)* computes a planar embedding of G with maximum external face among all planar embeddings of G with minimum depth.

References

- [1] C. Batini, E. Nardelli, and R. Tamassia. A layout algorithm for data-flow diagrams. *IEEE Trans. Soft. Eng.*, SE-12(4):538–546, 1986.
- [2] C. Batini, M. Talamo, and R. Tamassia. Computer aided layout of entity relationship diagrams. *The Journal of Systems and Software*, 4:163–173, 1984.
- [3] C. Batini, M. Talamo, and R. Tamassia. Computer aided layout of entity-relationship diagrams. *Journal of Systems and Software*, 4:163–173, 1984.
- [4] D. Bienstock and C. L. Monma. On the computational complexity of embedding planar graphs to minimize certain distance measures. *Algorithmica*, 5(1):93–109, 1990.
- [5] N. Chiba, T. Nishizeki, S. Abe, and T. Ozawa. A linear algorithm for embedding planar graphs using PQ-trees. *J. Computer and System Sciences*, 30:54–76, 1985.
- [6] G. Di Battista, P. Eades, R. Tamassia, and I. G. Tollis. *Graph Drawing*. Prentice Hall, 1998.
- [7] G. Di Battista and R. Tamassia. On-line maintenance of triconnected components with SPQR-trees. *Algorithmica*, 15:302–318, 1996.
- [8] G. Di Battista and R. Tamassia. On-line planarity testing. *SIAM J. Comput.*, 25(5):956–997, 1996.
- [9] Didimo and Liotta. Computing orthogonal drawings in a variable embedding setting. In K. Chwa, editor, *Algorithms and Computation (Proc. ISAAC '98)*, volume 1533 of *LNCS*, pages 79–88. Springer-Verlag, 1998.
- [10] A. Garg and R. Tamassia. On the computational complexity of upward and rectilinear planarity testing. *SIAM J. Computing*, 31(2):601–625, 2002.
- [11] Graph drawing toolkit: An object-oriented library for handling and drawing graphs. <http://www.dia.uniroma3.it/gdt>.
- [12] C. Gutwenger and P. Mutzel. A linear time implementation of SPQR trees. In J. Marks, editor, *Graph Drawing (Proc. GD 2000)*, volume 1984 of *LNCS*, pages 77–90. Springer-Verlag, 2001.
- [13] G. Liotta, F. Vargiu, and G. Di Battista. Orthogonal drawings with the minimum number of bends. In *Proceedings of the 6th Canadian Conference on Computational Geometry*, pages 281–286. University of Saskatchewan, 1994.
- [14] K. Mehlhorn and P. Mutzel. On the embedding phase of the Hopcroft and Tarjan planarity testing algorithm. *Algorithmica*, 16(2):233–242, 1996.
- [15] P. Mutzel and R. Weiskircher. Bend minimization in orthogonal drawings using integer programming. In O. H. Ibarra and L. Zhang, editors, *Proceedings of the Eighth Annual International Conference on Computing and Combinatorics (COCOON 2002)*, volume 2387 of *LNCS*, pages 484–493. Springer-Verlag, 2002.
- [16] M. Pizzonia and R. Tamassia. Minimum depth graph embedding. In M. Paterson, editor, *Algorithms – ESA 2000*, volume 1879 of *LNCS*, pages 356–367. Springer-Verlag, 2000.

- [17] R. Tamassia. On embedding a graph in the grid with the minimum number of bends. *SIAM J. Comput.*, 16(3):421–444, 1987.
- [18] R. Tamassia, G. Di Battista, and C. Batini. Automatic graph drawing and readability of diagrams. *IEEE Trans. Syst. Man Cybern.*, SMC-18(1):61–79, 1988.
- [19] R. E. Tarjan. Depth-first search and linear graph algorithms. *SIAM J. Comput.*, 1(2):146–160, 1972.
- [20] R. Weiskircher. *New Applications of SPQR-Trees in Graph Drawing*. PhD thesis, Universität des Saarlandes, 2002.