Optimizing Over All Combinatorial Embeddings of a Planar Graph
(Extended Abstract)

Petra Mutzel[⋆1] and René Weiskircher[⋆⋆2]

[1] mutzel@mpi-sb.mpg.de
[2] weiski@mpi-sb.mpg.de

Max–Planck–Institut für Informatik, Saarbrücken

Abstract. We study the problem of optimizing over the set of all combinatorial embeddings of a given planar graph. Our objective function prefers certain cycles of G as face cycles in the embedding. The motivation for studying this problem arises in graph drawing, where the chosen embedding has an important influence on the aesthetics of the drawing. We characterize the set of all possible embeddings of a given biconnected planar graph G by means of a system of linear inequalities with $\{0,1\}$-variables corresponding to the set of those cycles in G which can appear in a combinatorial embedding. This system of linear inequalities can be constructed recursively using SPQR-trees and a new splitting operation. Our computational results on two benchmark sets of graphs are surprising: The number of variables and constraints seems to grow only linearly with the size of the graphs although the number of embeddings grows exponentially. For all tested graphs (up to 500 vertices) and linear objective functions, the resulting integer linear programs could be generated within 10 minutes and solved within two seconds on a Sun Enterprise 10000 using CPLEX.

1 Introduction

A graph is called planar when it admits a drawing into the plane without edge-crossings. There are infinitely many different drawings for every planar graph, but they can be divided into a finite number of equivalence classes. We call two planar drawings of the same graph *equivalent* when the sequence of the edges in clockwise order around each node is the same in both drawings. The equivalence classes of planar drawings are called *combinatorial embeddings*. A combinatorial embedding also defines the set of cycles in the graph that bound faces in a planar drawing.

[⋆] Partially supported by DFG-Grant Mu 1129/3-1, Forschungsschwerpunkt "Effiziente Algorithmen für diskrete Probleme und ihre Anwendungen"

[⋆⋆] Supported by the Graduiertenkolleg "Effizienz und Komplexität von Algorithmen und Rechenanlagen"

The complexity of embedding planar graphs has been studied by various authors in the literature [5, 4, 6]. E.g., Bienstock and Monma have given polynomial time algorithms for computing an embedding of a planar graph that minimizes various distance functions to the outer face [5]. Moreover, they have shown that computing an embedding that minimizes the diameter of the dual graph is NP-hard.

In this paper we deal with the following optimization problem concerned with embeddings: Given a planar graph and a cost function on the cycles of the graph. Find an embedding Π such that the sum of the cost of the cycles that appear as face cycles in Π is minimized. When choosing the cost 1 for all cycles of length greater or equal to five and 0 for all other cycles, the problem is NP-hard [12].

Our motivation to study this optimization problem and in particular its integer linear programming formulation arises in graph drawing. Most algorithms for drawing planar graphs need not only the graph as input but also a combinatorial embedding. The aesthetic properties of the drawing often changes dramatically when a different embedding is chosen.

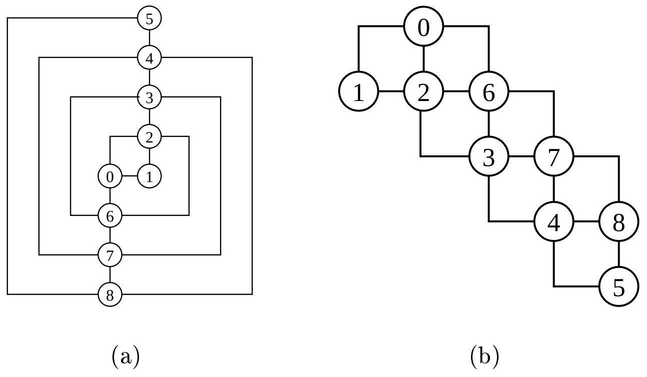


Fig. 1. The impact of the chosen planar embedding on the drawing

Figure 1 shows two different drawings of the same graph that were generated using the bend minimization algorithm by Tamassia [11]. The algorithm used different combinatorial embeddings as input. Drawing 1(a) has 13 bends while drawing 1(b) has only 7 bends. It makes sense to look for the embedding that will produce the best drawing. Our original motivation has been the following.

In graph drawing it is often desirable to optimize some cost function over all possible embeddings in a planar graph. In general these optimization problems are NP-hard [9]. For example: The number of bends in an orthogonal planar drawing highly depends on the chosen planar embedding. In the planarization method, the number of crossings highly depends on the chosen embedding when the deleted edges are reinserted into a planar drawing of the rest-graph. Both problems can be formulated as flow problems in the geometric dual graph. A flow between vertices in the geometric dual graph corresponds to a flow between adjacent face cycles in the primal graph. Once we have characterized the set of all feasible embeddings (via an integer linear formulation on the variables associated with each cycle), we can use this in an ILP-formulation for the corresponding flow problem. Here, the variables consist of 'flow variables' and 'embedding variables'.

This paper introduces an integer linear program whose set of feasible solutions corresponds to the set of all possible combinatorial embeddings of a given biconnected planar graph. One way of constructing such an integer linear program is by using the fact that every combinatorial embedding corresponds to a 2-fold complete set of cycles (see MacLane [10]). The variables in such a program are all simple cycles in the graph; the constraints guarantee that the chosen subset of all simple cycles is complete and that no edge of the graph appears in more than two simple cycles of the subset.

We have chosen another way of formulating the problem. The advantage of our formulation is that we only introduce variables for those simple cycles that form the boundary of a face in at least one combinatorial embedding of the graph, thus reducing the number of variables tremendously. Furthermore, the constraints are derived using the structure of the graph. We achieve this by constructing the program recursively using a data structure called SPQR-tree suggested by Di Battista and Tamassia ([2]) for the on-line maintenance of triconnected components. The static variant of this problem was studied in [?]. SPQR-trees can be used to code and enumerate all possible combinatorial embeddings of a biconnected planar graph. Furthermore we introduce a new splitting operation which enables us to construct the linear description recursively.

Our computational results on two benchmark sets of graphs have been quite surprising. We expected that the size of the linear system will grow exponentially with the size of the graph. Surprisingly, we could only observe a linear growth. However, the time for generating the system grows sub-exponentially; but for practical instances it is still reasonable. For a graph with 500 vertices and 10^{19} different combinatorial embeddings the construction of the ILP took about 10 minutes. Very surprising was the fact that the solution of the generated ILPs took only up to 2 seconds using CPLEX.

Section 2 gives a brief description of the data structure SPQR-tree. In Section 3 we describe the recursive construction of the linear constraint system using a new splitting operation. Our computational results are described in Section 4.

2 SPQR-Trees

In this section, we give a brief description of the SPQR-tree data structure for biconnected planar graphs. A connected graph is biconnected, if it has no cut vertex. A *cut vertex* of a graph $G = (V, E)$ 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 connected graph without a separation pair is called *triconnected.*

SPQR-trees have been suggested by Di Battista and Tamassia ([2]). They represent a decomposition of a planar biconnected graph according to its split pairs. A *split pair* is a pair of nodes in the graph that is either connected by an edge or has the property that its removal increases the number of connected components. The *split components* of a split pair p are the maximal subgraphs of the original graph, for which p is not a split pair. When a split pair p is connected by an edge, one of the split components consists just of this edge together with the incident nodes while the other one is the original graph without the edge.

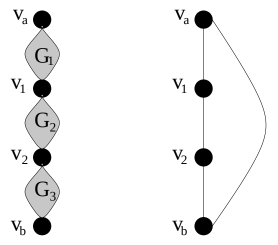
The construction of the SPQR-tree works recursively. At every node v of the tree, we split the graph into smaller edge-disjoint subgraphs. We add an edge to each of them to make sure that they are biconnected and continue by computing their SPQR-tree and making the resulting trees the subtrees of the node used for the splitting. Every node of the SPQR-tree has two associated graphs:

– The *skeleton* of the node defined by a split pair p is a simplified version of the whole graph where the split-components of p are replaced by single edges.
– The *pertinent graph* of a node v is the subgraph of the original graph that is represented by the subtree rooted at v.

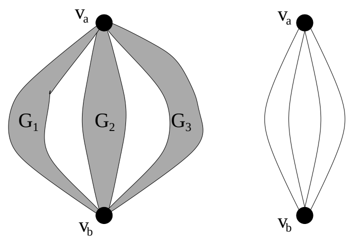
The two nodes of the split pair p that define a node v are called the *poles* of v. For the recursive decomposition, a new edge between the poles is added to the pertinent graph of a node which results in a biconnected graph that may have multiple edges. The SPQR-tree has four different types of nodes that are defined by the structure and number of the split components of its poles v_a and v_b:

1. **Q-node:** The pertinent graph of the node is just the single edge $e = \{v_a, v_n\}$. The skeleton consists of the two poles that are connected by two edges. One of the edges represents the edge e and the other one the rest of the graph.
2. **S-node:** The pertinent graph of the node has at least one *cut vertex* (a node whose removal increases the number of connected components). When we have the cut vertices v_1, v_2 to v_k, they then split the pertinent graph into the components G_1, G_2 to G_{k+1}. In the skeleton of the node, G_1 to G_{k+1} are replaced by single edges and the edge between the poles is added. The decomposition continues with the subgraphs G_i, where the poles are v_i and v_{i+1}. Figure 2(a) shows the pertinent graph of an S-node together with the skeleton.
3. **P-node:** v_a and v_b in the pertinent graph have more than one split-components G_1 to G_k. In the skeleton, each G_i is replaced by a single edge and the edge between the poles is added. The decomposition continues with the subgraphs G_i, where the poles are again v_a and v_b. Figure 2(b) shows the pertinent graph of a P-node with 3 split components and its skeleton.
4. **R-node:** None of the other cases is applicable, so the pertinent graph is biconnected. The poles v_a and v_b are not a split pair of the pertinent graph.

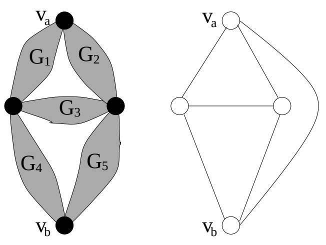
In this case, the decomposition depends on the *maximal split pairs* of the pertinent graph with respect to the pair $\{v_a, v_b\}$. A split pair $\{v_1, v_2\}$ is maximal with respect to $\{v_a, v_b\}$, if for every other split pair $\{v_1', v_2'\}$, there is a split component that includes the nodes v_1, v_2, v_a and v_b. For each maximal split pair p with respect to $\{v_a, v_b\}$, we define a subgraph G_p of the original graph as the union of all the split-components of p that do not include v_a and v_b. In the skeleton, each subgraph G_p is replaced by a single edge and the edge between the poles is added. The decomposition proceeds with the subgraphs defined by the maximal split pairs (see Fig. 2(c)).



(a) S-node



(b) P-node



(c) R-node

Fig. 2. Pertinent graphs and skeletons of the different node types of an SPQR-tree

The SPQR-tree of a biconnected planar graph G where one edge is marked (the so-called *reference edge*) is constructed in the following way:

1. Remove the *reference edge* and consider the end-nodes of it as the poles of the remaining graph G'. Depending on the structure of G' and the number of split components of the poles, choose the type of the new node v (S, P, R or Q).
2. Compute the subgraphs G_1 to G_k as defined above for the different cases and add an edge between the poles of each of the subgraphs.

3. Compute the SPQR-trees T_1 to T_k for the subgraphs where the added edge is the reference edge and make the root of these trees the sons of v.

When we have completed this recursive construction, we create a new Q-node representing the reference edge of G and make it the root of the whole SPQR-tree by making the old root a son of the Q-node. This construction implies that all leaves of the tree are Q-nodes and all inner nodes are S-, P-, or R-nodes. Figure 3 shows a biconnected planar graph and its SPQR-tree where the edge $\{1, 2\}$ was chosen as the reference edge.

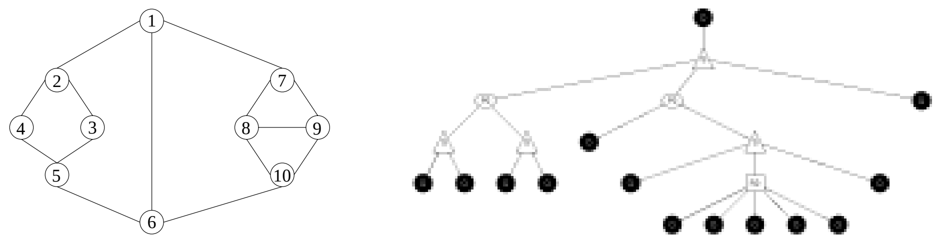


Fig. 3. A biconnected planar graph and its SPQR-tree

When we see the SPQR-tree as an unrooted tree, we get the same tree no matter what edge of the graph was marked as the reference edge. The skeletons of the nodes are also independent of the choice of the reference edge. Thus, we can define a unique SPQR-tree for each biconnected planar graph. Another important property of these trees is that their size (including the skeletons) is linear in the size of the original graph and they can be constructed in linear time ([2]).

As described in [2], SPQR-trees can be used to represent all combinatorial embeddings of a biconnected planar graph. This is done by choosing 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. In most publications, combinatorial embeddings are defined in such a way, that only one combinatorial embedding for a triconnected planar graph exists (note that a combinatorial embedding does not fix the outer face of a drawing which realizes the embedding). Our definition distinguishes between two combinatorial embeddings which are mirror-images of each other (the order of the edges around each node in clockwise order is reversed in the second embedding). When the skeleton of a P-node has k edges, there are $(k-1)!$ different embeddings of its skeleton.

Every combinatorial embedding of the original graph defines a unique combinatorial 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. The reason for this fact is that each skeleton is a simplified version of the original graph where the split components of some split pair are replaced by single edges. Thus, if the SPQR-tree of G has r R-nodes and the P-nodes P_1 to P_k where the skeleton of P_i has L_i edges, than the number of combinatorial embeddings of G is exactly

$$2^r \sum_{i=1}^{k} (L_i - 1)! \quad .$$

Because the embeddings of the R- and P-nodes determine the embedding of the graph, we call these nodes the *decision nodes* of the SPQR-tree. In [3], the fact that SPQR-trees can be used to enumerate all combinatorial embeddings of a biconnected planar graph was used to devise a branch-and-bound algorithm for finding a planar embedding and an outer face for a graph such that the drawing computed by Tamassia's algorithm has the minimum number of bends among all possible orthogonal drawings of the graph.

3 Recursive Construction of the Integer Linear Program

3.1 The Variables of the Integer Linear Program

The skeletons of P-nodes are *multi-graphs*, so they have multiple edges between the same pair of nodes. Because we want to talk about directed cycles, we can be much more precise when we are dealing with *bidirected graphs*. A directed graph is called *bidirected* if there exists a bijective function $r : E \to E$ such that for every edge $e = (v, w)$ with $r(e) = e^R$ we have $e^R = (w, v)$ and $r(e^R) = e$ We can turn an undirected graph into a bidirected graph by replacing each undirected edge by two directed edges that go in opposite directions. The undirected graph G that can be transformed in this way to get the bidirected graph G' is called the *underlying graph* of G'.

A *directed cycle* in the bidirected graph $G = (V, E)$ is a sequence of edges of the following form: $c = ((v_1, v_2), (v_2, v_3), \ldots, (v_k, v_1)) = (e_1, e_2, \ldots, e_k)$ with the properties that every node of the graph is contained in at most two edges of c and if $k = 2$, then $e_1 \neq e_2$ holds. We say a planar drawing of a bidirected graph is the drawing of the underlying graph, so the embeddings of a bidirected graph are identical with the embeddings of the underlying graph.

A *face cycle* in a combinatorial embedding of a bidirected planar graph is a directed cycle of the graph, such that in any planar drawing that realizes the embedding, the left side of the cycle is empty. Note that the number of face cycles of a planar biconnected graph is $m - n + 2$ where m is the number of edges in the graph and n the number of nodes.

Now we are ready to construct an integer linear program (ILP) in which the feasible solutions correspond to the combinatorial embeddings of a biconnected planar bidirected graph. The variables of the program are binary and they correspond to directed cycles in the graph. As objective function, we can choose any linear function on the directed cycles of the graph. With every cycle c we associate a binary variable x_c. In a feasible solution of the integer linear program, a variable x_c has value 1 if the associated cycle is a face cycle in the represented embedding and 0 otherwise. To keep the number of variables as small as possible, we only introduce variables for those cycles that are a face cycle in at least one combinatorial embedding of the graph.

3.2 Splitting an SPQR-Tree

We use a recursive approach to construct the variables and constraints of the ILP. Therefore, we need an operation that constructs a number of smaller problems out of our original problem such that we can use the variables and constraints computed for the smaller problems to compute the ILP for the original problem. This is done by splitting the SPQR-tree at some decision-node v. Let e be an edge incident to v whose other endpoint is not a Q-node. Deleting e splits the tree into two trees T_1 and T_2. We add a new edge with a Q-node attached to both trees to replace the deleted edge and thus ensure that T_1 and T_2 become complete SPQR-trees again. The edges corresponding to the new Q-nodes are called *split edges*. For incident edges of v, whose other endpoint is a Q-node, the splitting is not necessary. Doing this for each edge incident to v results in $d+1$ smaller SPQR-trees, called the *split-trees* of v, where d is the number of inner nodes adjacent to v . This splitting process is shown in Fig. 4. Since the new trees are SPQR-trees, they represent planar biconnected graphs which are called the *split graphs* of v. We will show how to compute the ILP for the original graph using the ILPs computed for the split graphs.

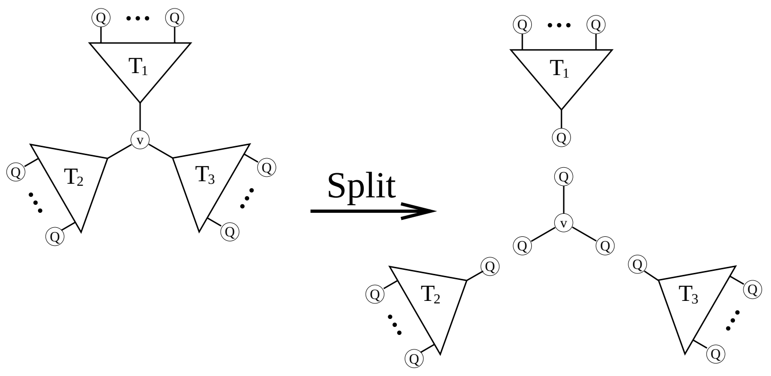


Fig. 4. Splitting an SPQR-tree at an inner node

As we have seen, the number and type of decision-nodes in the SPQR-tree of a graph determines the number of combinatorial embeddings. The subproblems we generated by splitting the tree either have only one decision-node or at least one fewer than the original problem.

3.3 The Integer Linear Program for SPQR-Trees with One Inner Node

We observe that a graph whose SPQR-tree has only one inner node is isomorphic to the skeleton of this inner node. So the split-tree of v which includes v, called the *center split-tree* of v, represents a graph which is isomorphic to the whole graph.

The ILPs for SPQR-trees with only one inner node are defined as follows:

– S-node: When the only inner node of the SPQR-tree is an S-node, the whole graph is a simple cycle. Thus it has two directed cycles and both are face-cycles in the only combinatorial embedding of the graph. So the ILP consists of two variables, both of which must be equal to one.
– R-node: In this case, the whole graph is triconnected. According to our definition of planar embedding, every triconnected graph has exactly two embeddings, which are mirror-images of each other. When the graph has m edges and n nodes, we have $k = 2(m - n + 2)$ variables and two feasible solutions. The constraints are given by the convex hull of the points in k-dimensional space, that correspond to the two solutions.
– P-node: The whole graph consists only of two nodes connected by k edges with $k \geq 3$. Every directed cycle in the graph is a face cycle in at least one embedding of the graph, so the number of variables is equal to the number of directed cycles in the graph. The number of cycles is $l = 2\binom{k}{2}$ because we always get an undirected cycle by pairing two edges and, since we are talking about directed cycles, we get twice the number of pairs of edges. As already mentioned, the number of embeddings is $(k-1)!$. The constraints are given as the convex hull of the points in l-dimensional space that represent these embeddings.

3.4 Construction of the ILP for SPQR-Trees with More than One Inner Node

We define, how to construct the ILP of an SPQR-tree T from the ILPs of the split-trees of a decision node v of T. Let G be the graph that corresponds to T and $T_1, \ldots, T_k$ the split-trees of v representing the graphs G_1 to G_k. We assume that T_1 is the center split-tree of v. Now we consider the directed cycles of G. We can distinguish two types:

1. *Local cycles* are cycles of G that also appear in one of the graphs $G_1, \ldots, G_k$.
2. *Global cycles* of G are not contained in any of the G_i.

Every split-tree of v except the center split-tree is a subgraph of the original graph G with one additional edge (the split edge corresponding to the added Q-node). The graph that corresponds to the center split-tree may have more than one split edge. Note that the number of split edges in this graph is not necessarily equal to the degree of v, because v may have been connected to Q-nodes in the original tree. For every split edge e, we define a subgraph *expand*(e) of the original graph G, which is represented by e. The two nodes connected by a split edge always form a split pair p of G. When e belongs to the graph G_i represented by the split-tree T_i, then *expand*(e) is the union of all the split components of G that share only the nodes of p and no edge with G_i.

For every directed cycle c in a graph G_i represented by a split-tree, we define the set $R(C)$ of *represented cycles* in the original graph . A cycle c' of G is in $R(c)$, when it can be constructed from c by replacing every split edge $e = (v, w)$ in c by a simple path in *expand*(e) from v to w.

The variables of the ILPs of the split-trees that represent local cycles will also be variables of the ILP of the original graph G. But we will also have variables that correspond to global cycles of G. A global cycle c in G will get a variable in the ILP, when the following conditions are met:

1. There is a variable x_{c_1} in the ILP of T_1 with $c \in R(c_1)$.
2. For every split-tree T_i with $2 \leq i \leq k$ where c has at least one edge in G_i, there is a variable x_{c_i} in the ILP of T_i such that $c \in R(c_i)$.

So far we have defined all the variables for the integer linear program of G. The set C of all constraints of the ILP of T is given by

$$C = C_l \cup C_c \cup C_G \quad .$$

First we define the set C_l which is the set of *lifted constraints* of T. Each of the graphs $T_1, \ldots, T_k$ is a simplified versions of the original graph G. They can be generated from G by replacing some split components of one or more split pairs by single edges. When we have a constraint that is valid for a split graph, a weaker version of this constraint is still valid for the original graph. The process of generating these new constraints is called *lifting* because we introduce new variables that cause the constraint to describe a higher dimensional half space or hyper plane. Let

$$\sum_{j=1}^{l} a_j x_{c_j} \doteq R$$

be a constraint in a split-tree, where $\doteq\, \in \{\leq, \geq, =\}$ and let X be the set of all variables of T. Then the lifted constraint for the tree T is the following:

$$\sum_{j=1}^{l} a_j \sum_{c:\; c \in R(c_j) \cap X} x_c \doteq R$$

We define C_l as the set of lifted constraints of all the split-trees. The number of constraints in C_l is the sum of all constraints in all split-trees.

The set C_c is the set of *choice constraints*. For a cycle c in G_i, which includes a split edge, we have $|R(c)| > 1$. All the cycles in $R(c)$ share either at least one directed edge or they pass a split graph of the split node in the same direction. Therefore, only one of the cycles in $R(c)$ can be a face cycle in any combinatorial embedding of G (proof omitted). For each variable x_c in a split tree with $|R(c)| > 1$ we have therefore one constraint that has the following form:

$$\sum_{c' \in R(c) \wedge x_{c'} \in X} x_{c'} \leq 1$$

The set C_G consists of only one constraint, called the *center graph constraint*. Let F be the number of face cycles in a combinatorial embedding of G_1, C_G the set of all global cycles c in G and C_L the set of all local cycles c in G_1 then this constraint is:

$$\sum_{c \,\in\, (C_g \cup C_l) \cap C} x_c = F$$

This constraint is valid, because we can produce every drawing D of G by replacing all split edges in a drawing D_1 of G_1 with the drawings of subgraphs of G. For each face cycle in D_1, there will be a face cycle in D, that is either identical to the one in D_1 (if it was a local cycle) or is a global cycle. This defines the ILP for any biconnected planar graph.

3.5 Correctness of the ILP

Theorem 1. *Every feasible solution of the generated ILP corresponds to a combinatorial embedding of the given biconnected planar graph G and vice versa: every combinatorial embedding of G corresponds to a feasible solution for the generated ILP.*

Because the proof of the theorem is quite complex and the space is limited, we can only give a sketch of the proof. The proof is split into three lemmas.

Lemma 1. *Let G be a biconnected planar graph and let T be its SPQR-Tree. Let μ be a decision node in T with degree d, $T_1, \ldots, T_{d'}$ with $d' \leq d$ be the split trees of μ (T_1 is the center split tree) and $G_1, \ldots, G_{d'}$ the associated split graphs. Every combinatorial embedding Γ of G defines a unique embedding for each G_i. On the other side, if we fix a combinatorial embedding Γ_i for each G_i, we have defined a unique embedding for G.*

proof: (Sketch) To show the first part of the lemma, we start with a drawing Z of G that realizes embedding Γ. When G'_i is the graph G_i without its split edge, we get a drawing Z_1 of G_1 by replacing in Z the drawings of the G'_i with $2 \leq i \leq d'$ with drawings of single edges that are drawn inside the area of the plane formerly occupied by the drawing of G'_i. We can show that each drawing of G_1 that we construct in this way realizes the same embedding Γ_1. We construct a planar drawing of each G_i with $2 \leq i \leq d'$ by deleting all nodes and edges from Z that are not contained in G_i and drawing the split edge into the area of the plane that was formerly occupied by the drawing of a path in G between the poles of G_i not inside G_i. Again we can show that all drawings produced in this way realize the same embedding Γ_i.

To show the second part of the lemma, we start with special planar drawings Z_i of the G_i that realize the embeddings Γ_i. We assume that Z_1 is a straight line drawing (such a drawing always exists [8]) and that each Z_i with $2 \leq i \leq d'$ is drawn inside an ellipse with the split edge on the outer face and the poles drawn as the vertices on the major axis of the ellipse. Then we can construct a drawing of G by replacing the drawings of the straight edges in Z_1 by the drawings Z_i of the G_i from which the split edges have been deleted. We can show that every drawing Z we construct in this way realizes the same embedding Γ of G.

■

To proof the main theorem, we first have to define the *incidence vector* of a combinatorial embedding. Let C be the set of all directed cycles in the graph that are face cycles in at least one combinatorial embedding of the graph. Then the incidence vector of an embedding Γ is given as a vector in $\{0,1\}^{|C|}$ where the components representing the face cycles in Γ have value one and all other components have value zero.

Lemma 2. *Let $\Gamma = \{c_1, c_2, \ldots, c_k\}$ be a combinatorial embedding of the biconnected planar graph G. Then the incidence vector χ^Γ satisfies all constraints of the ILP we defined.*

proof: (Sketch) We proof the lemma using induction over the number n of decision nodes in the SPQR-Tree T of G. The value $\chi(c)$ is the value of the component in χ associated with the cycle c. We don't consider the case $n = 0$, because G is a simple cycle in this case and has only one combinatorial embedding.

1. $n = 1$:
   No splitting of the SPQR-tree is necessary, the ILP is defined directly by T. The variables are defined as the set of all directed cycles that are face cycles in at least one combinatorial embedding of G. Since the constraints of the ILP are defined as the convex hull of all incidence vectors of combinatorial embeddings of G, χ^Γ satisfies all constraints of the ILP.
2. $n > 1$:
   From the previous lemma we know that Γ uniquely defines embeddings Γ_i with incidence vectors χ_i for the split graphs G_i. We will use the induction basis to show that χ^Γ satisfies all lifted constraints. We know that the choice constraints are satisfied by χ^Γ because in any embedding there can be only on cycle passing a certain split pair in the same direction. When lifting constraints, we replace certain variables by the sum of new variables and the choice constraints guarantee that this sum is either 0 or 1. Using this fact and the construction of the χ_i from χ^Γ, we can show that the sums of the values of the new variables are always equal to the value of the old variable. Therefore, all lifted constraints are satisfied.
   To see that the center graph constraint is satisfied, we observe that any embedding of the skeleton of the split node has F faces. We can construct any embedding of G from an embedding Γ_1 of this skeleton by replacing

edges by subgraphs. The faces in Γ that are global cycles are represented by faces in Γ_1 and the faces that are local cycles in G are also faces in Γ_1. Therefore the center graph constraint is also satisfied.

■

Lemma 3. *Let G be a biconnected planar graph and $\chi \in \{0,1\}^{|C|}$ a vector satisfying all constraints of the ILP. Then χ is the incidence vector of a combinatorial embedding Γ of G.*

proof: Again, we use induction on the number n of decision nodes in the SPQR-tree T of G and we disregard the case $n = 0$.

1. $n = 1$:
   Like in the previous lemma, our claim holds by definition of the ILP.
2. $n > 1$:
   The proof works in two stages: First we construct vectors χ_i for each split graph from χ and prove that these vectors satisfy the ILPs for the G_i, and are therefore incidence vectors of embeddings Γ_i of the G_i by induction basis. In the second stage, we use the Γ_i to construct an embedding Γ for G and show that χ is the incidence vector of Γ.
   The construction of the χ_i works as follows: When x is a variable in the ILP of G_i and the corresponding cycle is contained in G, then x gets the value of the corresponding variable in χ. Otherwise, we define the value of x as the sum of the values of all variables in χ whose cycles are represented by the cycle of x. This value is either 0 or 1 because χ satisfies the choice constraints.
   Because χ satisfies the lifted constraints, the χ_i must satisfy the original constraints and by induction basis we know that each χ_i represents an embedding Γ_i of G_i. Using these embeddings for the split graphs, we can construct an embedding Γ for G like in lemma 1.
   To show that χ is the incidence vector of Γ, we define χ^Γ as the incidence vector of Γ and show that χ and χ^Γ are identical. By construction of Γ and χ^Γ, the components in χ^Γ and χ corresponding to local cycles must be equal. The number of global cycles whose variable in χ has value 1 must be equal to the number of faces in Γ consisting of global cycles. This is guaranteed by the center graph constraint. Using the fact that for all face cycle in Γ_1 there must be a represented cycle in G whose component in χ and in χ^Γ is 1 we can show that both vectors agree on the values of the variables of the global cycles, and thus must be identical.

■

4 Computational Results

In our computational experiments, we tried to get statistical data about the size of the integer linear program and the times needed to compute it. Our implementation works for biconnected planar graphs with maximal degree four, since we

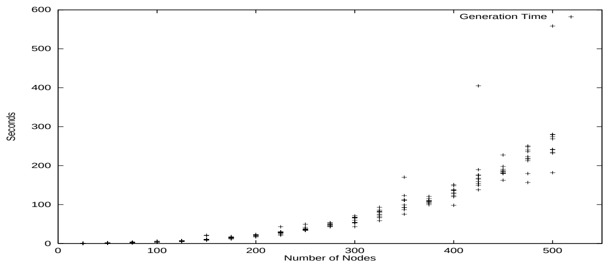


Fig. 5. Generation time for the ILP

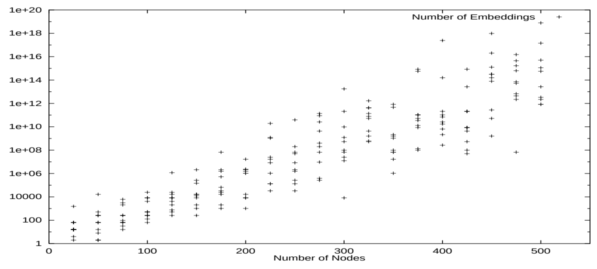


Fig. 6. Number of embeddings

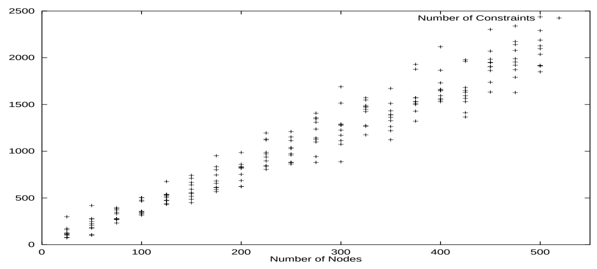


Fig. 7. Number of constraints

are interested in improving orthogonal planar drawings. First we used a benchmark set of 11491 practical graphs collected by the group around G. Di Battista in Rome ([1]). We have transformed these graphs into biconnected planar graphs with maximal degree four using planarization, planar augmentation, and the ring approach described in [7]. This is a commonly used approach getting orthogonal drawings with a small number of bends [7]. The obtained graphs have up to 160 vertices; only some of them had more than 100 different combinatorial embeddings. The maximum number of embeddings for any of the graphs was 5000.

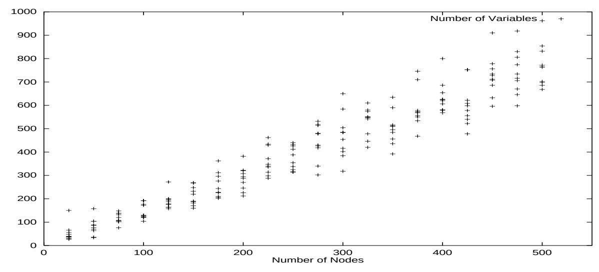


Fig. 8. Number of variables

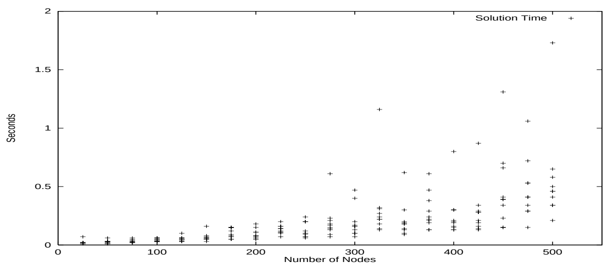


Fig. 9. Solution time

The times for generating the ILPs have been below one minute; the ILPs were quite small. CPLEX has been able to solve all of them very quickly.

In order to study the limits of our method, we started test runs on extremely difficult graphs. We used the random graph generator developed by the group around G. Di Battista in Rome that creates biconnected planar graphs with maximal degree four with an extremely high number of embeddings (see [3] for detailed information). We generated graphs with the number of nodes ranging from 25 to 500, proceeding in steps of 25 nodes and generating 10 random graphs for each number of nodes. For each of the 200 graphs, we generated the ILP and measured the time needed to do this. The times are shown in Fig. 5. They grow sub-exponentially and the maximum time needed was 10 minutes on a Sun Enterprise 10000.

The number of embeddings of each graph is shown in Fig. 6. They grow exponentially with the number of nodes, so we used a logarithmic scale for the y-axis. There was one graph with more than 10^{19} combinatorial embeddings. These numbers were computed by counting the number of R- and P-nodes in the SPQR-tree of each graph. Each R-node doubles the number of combinatorial embeddings while each P-node multiplies it by 2 or 6 depending on the number of edges in its skeleton. Figures 7 and 8 show the number of constraints and variables in each ILP. Surprisingly, both of them grow linearly with the number of nodes. The largest ILP has about 2500 constraints and 1000 variables.

To test how difficult it is to optimize over the ILPs, we have chosen 10 random objective functions for each ILP with integer coefficients between 0 and 100 and computed a maximal integer solution using the mixed integer solver of CPLEX. Figure 9 shows the maximum time needed for any of the 10 objective functions. The computation time always stayed below 2 seconds.

Our future goal will be to extend our formulation such that each solution will not only represent a combinatorial embedding but an orthogonal drawing of the graph. This will give us a chance to find drawings with the minimum number of bends or drawings with fewer crossings. Of course, this will make the solution of the ILP much more difficult.

Acknowledgments We thank the group of G. Di Battista in Rome for giving us the opportunity to use their implementation of SPQR-trees in *GDToolkit*, a software library that is part of the ESPRIT ALCOM-IT project (work package 1.2), and to use their graph generator.

References

[1] G. Di Battista, A. Garg, G. Liotta, R. Tamassia, E. Tassinari, and F. Vargiu. An experimental comparison of four graph drawing algorithms. *Comput. Geom. Theory Appl.*, 7:303–326, 1997.

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

[3] P. Bertolazzi, G. Di Battista, and W. Didimo. Computing orthogonal drawings with the minimum number of bends. *Lecture Notes in Computer Science*, 1272:331–344, 1998.

[4] D. Bienstock and C. L. Monma. Optimal enclosing regions in planar graphs. *Networks*, 19(1):79–94, 1989.

[5] D. Bienstock and C. L. Monma. On the complexity of embedding planar graphs to minimize certain distance measures. *Algorithmica*, 5(1):93–109, 1990.

[6] J. Cai. Counting embeddings of planar graphs using DFS trees. *SIAM Journal on Discrete Mathematics*, 6(3):335–352, 1993.

[7] P. Eades and P. Mutzel. *Algorithms and theory of computation handbook*, chapter 9 Graph drawing algorithms. CRC Press, 1999.

[8] I. Fary. On straight line representing of planar graphs. *Acta. Sci. Math.(Szeged)*, 11:229–233, 1948.

[9] A. Garg and R. Tamassia. On the computational complexity of upward and rectilinear planarity testing. *Lecture Notes in Computer Science*, 894:286–297, 1995.

[10] S. MacLane. A combinatorial condition for planar graphs. *Fundamenta Mathematicae*, 28:22–32, 1937.

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

[12] G. J. Woeginger. personal communications, 1998.