Optimizing Over All Combinatorial Embeddings of a Planar Graph

Petra Mutzel René Weiskircher

MPI–I–98–1-029 December 1998

Author's Address

Max-Planck-Institut für Informatik
Im Stadtwald
66123 Saarbrücken
`http://www.mpi-sb.mpg.de/~mutzel` and `http://www.mpi-sb.mpg.de/~weiski`

Acknowledgements

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.

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 the data structure of 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 600 seconds 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.

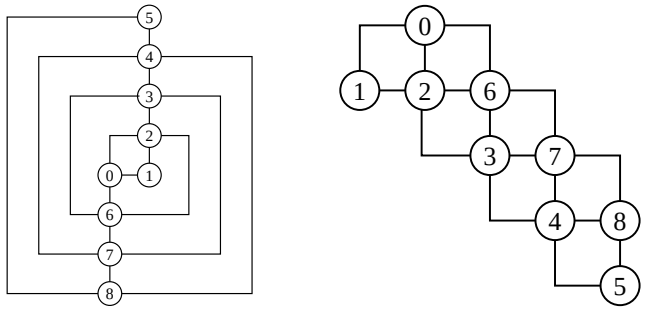
The complexity of embedding planar graphs has been studied by various authors in the literature [5, 4, 6]. E.g., Bienstock and Monma have given polynomial time algorithms for computing an embedding of a planar graph that minimizes various distance functions to the outer face [5]. Moreover, they have 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 [13].

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.

Figure 1 shows two different drawings of the same graph that were generated using the bend minimization algorithm by Tamassia [12]. 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. We conjecture that there are statistical dependencies between the length of the face cycles in the embedding and nice drawings. If we are able to solve the stated optimization problem efficiently for practical instances, we will be able to do extensive experiments in order to get insights into this. But 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 [10]. 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 prob-



(a) (b)

Figure 1: The impact of the chosen planar embedding on the drawing

lems 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 circuits (see MacLane [11]). 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]). 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 5.

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 adjacent 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)).

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.

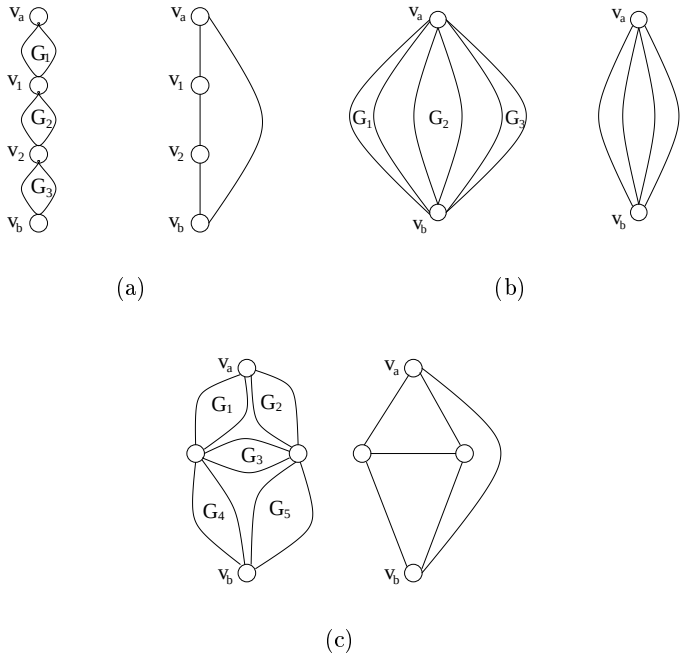


Figure 2: Pertinent graphs and skeletons of the different node types of an SPQR-tree

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.

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

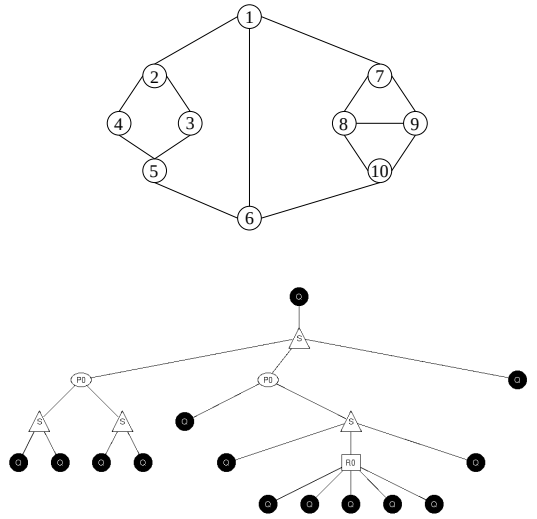


Figure 3: A biconnected planar graph and its SPQR-tree

Q-nodes are simple circles, 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. 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 drawing). 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 Intuition

The SPQR-tree represents the decomposition of a biconnected planar graph with respect to its triconnected components. All embeddings of the graph can be enumerated by enumerating all possible embeddings of these components in respect to the rest of the graph. Our approach uses the fact that each skeleton of a node in the SPQR-tree represents a simplified version of the original graph. By computing the integer linear programs (ILP) for these simple graphs and using a lifting procedure, we can compute an ILP for the original graph.

When we take a closer look at the skeleton of a node in the SPQR-tree, we observe that it can be constructed from the original graph by replacing one or several subgraphs by single edges, which we will later call *split edges*. We can think of such an edge as representing the set of all the simple paths in the original graph, that connect the two nodes of the split edge. So every circle in a skeleton that includes a split edge represents a set of circles in the original graph that we get by replacing the split edge with the paths represented by it.

The variables of our program correspond to directed circles in the graph that are face cycles in at least one planar embedding. We can guarantee this, because our recursive construction computes the set of variables for the original problem using the sets of variables from subproblems for which the ILP has already been computed. So we construct circles in the original graph from circles in the subproblems by replacing split edges with paths in the graph.

3.2 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 circles, 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 circle* 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 circle of the graph, such that in any planar drawing that realizes the embedding, the left side of the circle 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 circles in the graph. As objective function, we can choose any linear function on the directed circles of the graph. With every circle 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 circle 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 circles that are a face cycle in at least one combinatorial embedding of the graph.

3.3 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 adjacent edge of 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 adjacent edges of v, whose other endpoint is a Q-node, the splitting is not necessary. Doing this for each edge adjacent 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.

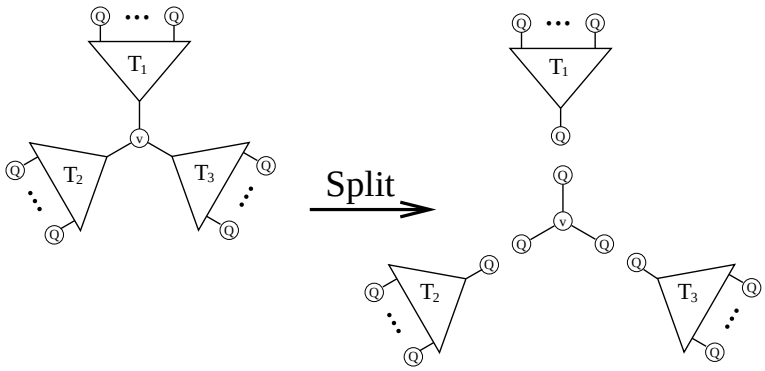


Figure 4: Splitting an SPQR-tree at an inner node

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.

3.4 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 circle. Thus it has two directed circles 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 circle 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 circles in the graph. The number of circles is

$$l = 2\binom{k}{2}$$

because we always get an undirected circle by pairing two edges and, since we are talking about directed circles, 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.

3.5 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 circles of G. We can distinguish two types:

1. *Local circles* are circles of G that also appear in one of the graphs $G_1, \ldots, G_k$.

2. *Global circles* 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 circle c in a graph G_i represented by a split-tree, we define the set $R(C)$ of *represented circles* in the original graph . A circle 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 circles will also be variables of the ILP of the original graph G. But we will also have variables that correspond to global circles of G. A global circle 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 circle c in G_i, which includes a split edge, we have $|R(c)| > 1$. All the circles 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 circles 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 circles c in G and C_L the set of all local circles 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 circle) or is a global circle. This defines the ILP for any biconnected planar graph.

4 The main theorem and its proof

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 main theorem is quite complex, we have split it 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 E of G defines a unique embedding for each G_i. On the other side, if we fix a combinatorial embedding E_i for each G_i, we have defined a unique embedding for G.*

Proof: First we will show how E defines a unique combinatorial embedding for G_i with $2 \leq i \leq d'$. Let e be the split edge in G_i and G_i' be the graph we get by deleting e from G_i and let Z be a drawing that realizes E. Then we can construct a planar drawing Z_i' of G_i' by deleting all the drawings of nodes and edges from Z that are not contained in G_i'. To get a drawing Z_i of G_i, we have to add a drawing of e to Z_i'. We have to prove, that every planar drawing we can construct in this way realizes the same embedding E_i.

First we show, that it is possible to add a drawing of e without losing planarity. Let u and v be the two nodes connected by e. Then $\{u, v\}$ is a split pair, and removing these nodes from G splits the graph in at least 2 components. One of these components is G_i'. Since all split components are connected, there must be a path connecting u and v in each of them which proves that there is a path p in G connecting u and v that does not use any edge in G_i'. Since Z is a planar drawing, we can get a planar drawing of G_i by drawing e as the same curve that represented p in Z.

To prove that every drawing of G_i that is constructed in this way realizes the same embedding E_i, we first observe that the embedding E_i' of G_i' realized by Z_i' is unique. What is left to show is that there are not two different face cycles c_1 and c_2 in E_i' that we can split by inserting the drawing of e. So we have to show that there is only one face in Z_i' where we can insert e. If

there where two such face cycles c_1 and c_2, then both must go through u and v. But then $\{u, v\}$ is a split pair of G'_i which is not possible because of the way SPQR-trees are constructed. This proves that every embedding E of G defines a unique embedding E_i for each G_i with $2 \leq i \leq d'$.

To prove that E defines a unique embedding E_1 for G_1, we take any drawing Z of G that realizes E and replace the drawing of every subgraph G'_i of G that is represented by a split edge $\{u, v\}$ in G_1 by a single edge that is drawn as the same curve as some path connecting u and v in G'_i. Thus we get a planar drawing Z_1 and since the sequence of the edges around each node is fixed using this construction, the embedding E_1 realized by Z_1 is unique. Figure 5 shows how to construct drawings for the split graphs from a drawing of G.

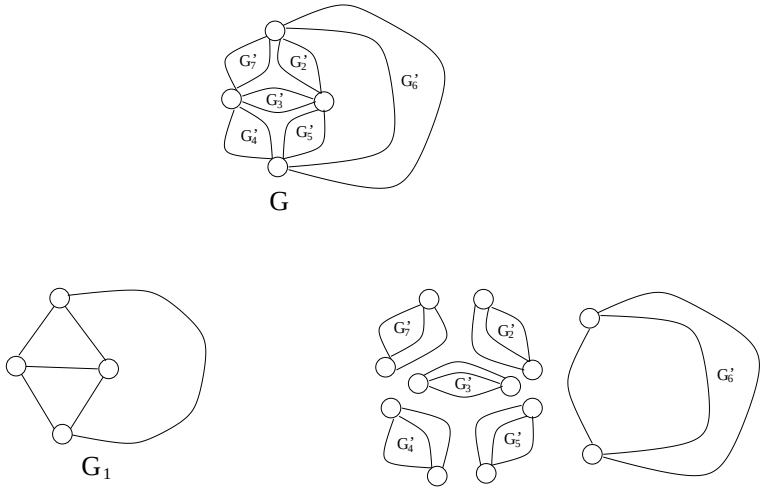


Figure 5: Deriving drawings for the split graphs from a drawing of G

To prove the opposite direction, we have to show how combinatorial embeddings for each G_i define a unique combinatorial embedding for G. To do this, we show how to construct a drawing of G from drawings of the G_i that realize the embeddings E_i and that this drawing always realizes the same embedding E. For each G_i with $2 \leq i \leq d$, we generate a drawing that realizes E_i and has the following properties:

1. The split edge of G_i is on the outer face.
2. Let the two nodes u_i and v_i be connected by the split edge of G_i. Then there is an ellipse L_i with the property that the drawings of u_i and v_i form the vertices on the major axis of L_i and all other edges and nodes of the drawing are inside L_i.

It is always possible to generate a drawing with these properties that realizes any combinatorial embedding. Let $Z_2, \ldots, Z_d$ be drawings of the G_i

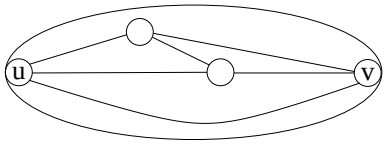


Figure 6: Drawing a split graph into an ellipse

that have the two properties stated above and Z_i' be the drawing we get by deleting the split edge from Z_i. Then each Z_i' is a drawing of a subgraph of G that is represented by a single edge (a split edge) in G_1. Consider now any drawing Z_1 of G_1 that realizes E_1 and where every edge is drawn as a straight line. We know that such a drawing exists ([9]). We get a planar drawing of G by replacing each drawing of a split edge in Z_1 by the drawing Z_i' of the associated split graph where we have removed the drawing of the split edge. Each drawing Z we generate in this way using the embeddings $E_1, \ldots, E_d$ realizes the same embedding E because the sequence of the edges around each node in clockwise order is fixed by the construction method. ■

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 circles in the graph that are a face cycle in at least one combinatorial embedding of the graph. Then the incidence vector of an embedding E is given as a vector in $\{0,1\}^{|C|}$ where the components representing the face cycles in C have value one and all other components have value zero.

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

Proof: 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 circle c. We don't consider the case $n = 0$, because G is a simple circle 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 circles 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_e satisfies all constraints of the ILP.

2. $n > 1$:
   Let μ be the node in T we used in the construction of the ILP to split T into the split trees $T_1, \ldots, T_d$. We split E into two subsets E_G and E_L such that E_G holds all global circles of E and E_L the local circles of E in respect to μ. Then every circle c in E_L is contained in some split graph G_i with $1 \leq i \leq d$. As we have seen in the previous lemma, the combinatorial embedding E defines uniquely a combinatorial embedding E_i for each G_i. Every local circle in E, that is contained in G_i is also contained in E_i. Apart from local circles that are contained in E, every E_i with $2 \leq i \leq d$ contains two more circles, that are not contained in E. This is true because every G_i has exactly one edge that is not contained in G, the split edge of G_i. In every embedding, each edge is contained in exactly two circles, so there are two circles in E_i that are not circles in E.

   Consider now the choice constraints of the ILP for all circles c in a split graph of μ that include at least one split edge.

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

   As we have already pointed out in the definition of the choice constraints, any two circles in R_c pass at least one edge of G in the same direction or pass at least one split graph of μ in the same direction. So they can't be face cycles in the same combinatorial embedding. Since E is a combinatorial embedding, χ_E must satisfy all choice constraints of the ILP.

   We use this fact and the induction basis to prove that χ_E satisfies all lifted constraints. Because our claim holds for $n-1$, we know that the incidence vector χ_{E_i} satisfies all constraints of the ILP for G_i with $1 \leq i \leq d$. The lifted constraints of the ILP are constructed out of the constraints for the G_i by replacing each variable x_c in the constraint by the sum of the variables for G, that are associated with circles in $R(c)$. The right side of the constraint and the relation sign remain unchanged. So we have to show, that the left side of the constraint has the same value when we apply it to χ_{E_i} as when we lift it and apply it to χ_E. Because the choice constraints hold, this is equivalent with the claim

$$\chi_i(c') = \sum_{c \in C \cap R(c')} \chi(c) \quad \forall \text{ variables } x_{c'} \text{ of the ILP of } G_i.$$

   If c is a local circle, this is obvious, because we have $R(c') = c'$ and c' is a face cycle in E if and only if it is a face cycle in E_i. Now we assume that c is a global circle. Every split edge in G_i represents a

subgraph of G, and each face cycle of E_i was generated from a face cycle of E by collapsing some path into a single edge. So when x_c is zero, there can be no face cycle in E that is in the set $R(c)$ and when x_c is one, there must be exactly one circle in $R(c)$ that is a face cycle of E. So the left side of the constraint has the same value when we lift it and apply it to χ_E and the constraint is satisfied.

When C_g is the set of global circles in G, C_l the set of local circles in G_1 and F the number of face cycles in any combinatorial embedding of G_1, then the center graph constraint has the following form:

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

The graph G_1 is the skeleton of the split node. It is constructed by replacing all the split graphs in G by single edges. So if c is a face cycle of E and belongs to C_l, then it is also a face cycle of E_1. And if c is a global face cycle of E, the circle c' in G_1 that was generated by replacing some paths of c by single edge is a face cycle in E_1. On the other side, every face cycle of E_1 is also a face cycle of E, if it contains no split edge and for every face cycle c of E_1 that contains at least one split edge, there is a global face cycle in E that can be constructed from c by replacing the split edges with paths from the corresponding split graphs. And since $|E_1| = F$, the equality is satisfied by χ_E.

■

Lemma 3 *If G is a biconnected planar graph and $\chi \in \{0,1\}^{|C|}$ a vector that satisfies all constraints of the ILP, than χ is the incidence vector of a combinatorial embedding E of G.*

Proof: Again, we use induction over the number n of decision nodes in the SPQR-tree T of G and we disregard the case $n = 0$ because G is only a simple circle in this case.

1. $n = 1$:
   T has only one decision node, and the skeleton of this node is isomorphic with G. The ILP is the convex hull of all incidence vectors of combinatorial embeddings and thus every $\{0,1\}$ vector that satisfies all constraints is the incidence vector of a combinatorial embedding.

2. $n > 1$:
   The proof works in two stages: First we construct vectors χ_i for each split graph out of χ and prove that these vectors satisfy the ILPs of the G_i, and are therefore incidence vectors of embeddings E_i of the G_i by induction basis. In the second stage, we use the E_i to construct an embedding E for G and show that χ is the incidence vector of E.

So first we construct the vectors χ_i. Let c be a circle in some G_i that is a face cycle in at least one embedding of G_i. If C is the set of circles in G that are face cycles in at least one embedding, we set:

$$\chi_i(c) = \sum_{c' \in C \cap R(c)} \chi(c')$$

Because the choice constraints are satisfied, the value of this sum is always zero or one and so all components in χ_i will be either zero or one. To show that χ_i is the incidence vector of a combinatorial embedding E_i, it is sufficient to prove that χ_i satisfies all constraints of the ILP of G_i, because the induction basis guarantees that χ_i is an incidence vector of an embedding when all constraints are satisfied.

We know that χ satisfies all constraints that are lifted from constraints in the ILP of G_i. Now the right side of the constraints does not change during lifting. Consider the left side of an arbitrary constraint lifted from the constraints of G_i, when C_i is the set of circles in G_i that are associated with variables in the ILP of G_i:

$$\sum_{c \in C_i} (\lambda_c \cdot \sum_{c' \in R(c) \cap C} x_{c'}) \quad (\lambda_c \in \mathbb{R})$$

The left side of the original constraint in the ILP of G_i is:

$$\sum_{c \in C_i} (\lambda_c \cdot x_c)$$

We constructed χ_i such that $\chi_i(c) = \sum_{c' \in C \cap R(c)} \chi(c')$ and thus we know that all constraints of G_i are satisfied by χ_i and that it is the incidence vector of a combinatorial embedding E_i of G_i.

We have completed stage 1 of the proof and proceed to stage 2, so we want to construct an embedding E for G from the E_i and show that χ is the incidence vector of E. We construct E as described in lemma 1. So every circle c in G_i for $1 \leq i \leq d$ which does not include a split edge is a face cycle in E if and only if it is a face cycle in E_i. Consider the way we have defined each χ_i:

$$\chi_i(c) = \sum_{c' \in C \cap R(c)} \chi(c')$$

Since c does not include a split edge, we have $R(c) = \{c\}$. And so we have that $\chi(c) = 1$ if and only if $c \in E$. We have now proven that χ and E agree about the local circles. It remains to show that they agree about the global circles.

First we show that the number of components with value 1 in χ that are associated with global circles is equal to the number of global circles in the set E. This is guaranteed by the center graph constraint:

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

Here C_g is the set of all global circles and C_l the set of all local circles in G_1. Since χ and E agree on all local circles, the number of global circles whose associated component in χ is 1 is $F - |C_l \cap E|$ which is equal to the number of global circles in E. We will now show that for all $c_g \in C_g \cap E$ we have $\chi(c_g) = 1$ and thus prove the lemma.

Let c_g be any circle in $C_g \cap E$. Then there is exactly one circle c_{1_g} in the center split graph G_1 with $c_g \in R(c_{1_g})$. We have $c_{1_g} \in E_1$ which follows from lemma 1. From that we can conclude that there is a circle c'_g in G with $c'_g \in R(c_{1_g})$ such that $\chi(c'_g) = 1$ (from the construction of χ_1 and E_1). So we have to show that $c'_g = c_g$. Since $\{c'_g, c_g\} \subseteq R(c_{1_g})$, both circles are represented by the same circle c_{1_g} in G_1. Consider now this circle c_{1_g}. It contains split edges and might also contain edges from G. When A_1 is the set of edges in E_{1_g} that are also edges in G and A_2 the set of split edges in c_{1_g}, then the edges in A_1 must be present in c_g and c'_g. So both circles can only differ in the paths they take through the split graphs having the split edges from A_2. Let e be an edge in A_2 and G_e be the split graph associated with this split edge. When p is the path of c_g that runs through G_e, then there is a circle c_l in E_e that consists of p and the split edge of G_e. This follows from the construction of E. If c'_g took a path p' different from p through G_e, we would have another circle in E_e passing through the split edge in the same direction as c_l. But in any embedding, there is only one circle that passes an edge in a certain direction. Therefor, p and p' must be the same.

Since we can apply this argumentation to all split edges in the circle c_{1_g}, we can show that c_g and c'_g are in fact the same circle. Thus we have $\chi(c_g) = 1$ and it follows that χ is indeed the incidence vector of the combinatorial embedding E.

■

5 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 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 ([7]). We have transformed these graphs into biconnected planar graphs with maximal degree four using planarization, planar augmentation, and the ring approach described in [8]. This is a commonly used approach for getting orthogonal drawings with a small number of bends [1].

For each of the resulting graphs, we constructed the integer linear program. As objective function, we minimized the number of face cycles with more than five edges. This problem is NP-complete ([13]). Figure 7 shows that the time for generating the ILP grows sub-exponentially with the number of nodes of the graph. We were able to generate every ILP in less than 40 seconds. Figure 8 shows the number of combinatorial embeddings of each of the graphs. We noticed that only very few graphs of the test-suit had a large number of embeddings. The maximum number of embeddings for any graph was 4608.

The Figs. 9 and 10 show that the number of constraints and variables in the ILP grow linearly with the size of the graphs. The maximum number of constraints was 428, while the maximum number of variables was 166. Figure 11 shows the time in seconds needed by the mipopt-solver of CPLEX to solve the ILPs to optimality. There was no ILP in our test-suit that took more than 0.06 seconds to solve.

In order to study the limits of our method, we started a similar test-run 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 again generated the ILP and measured the time needed to do this. The times are shown in Fig. 12. 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. 13. 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 14 and 15 show the number of constraints and variables in each ILP. 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 16 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.

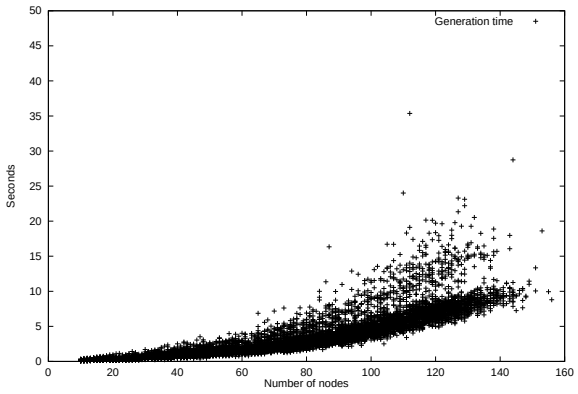


Figure 7: Generation time in the benchmark test-run

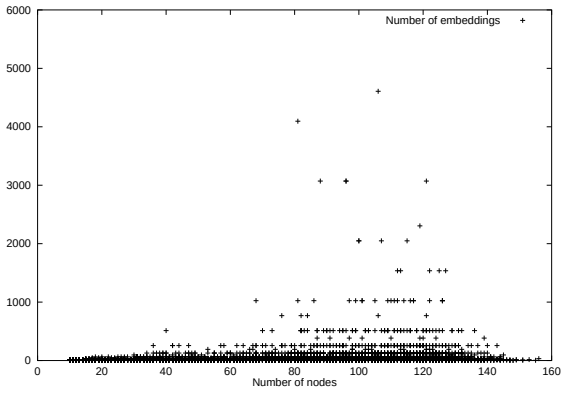


Figure 8: Number of embeddings in the benchmark test-run

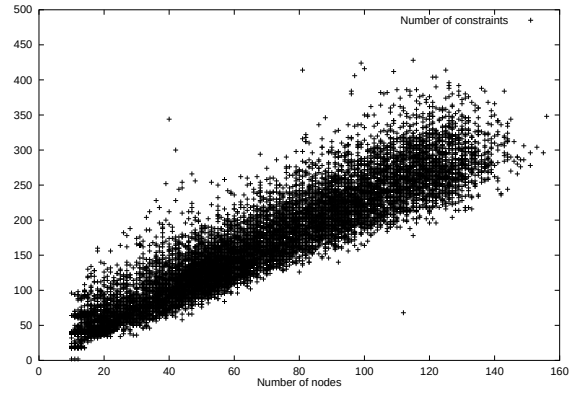


Figure 9: Number of constraints in the benchmark test-run

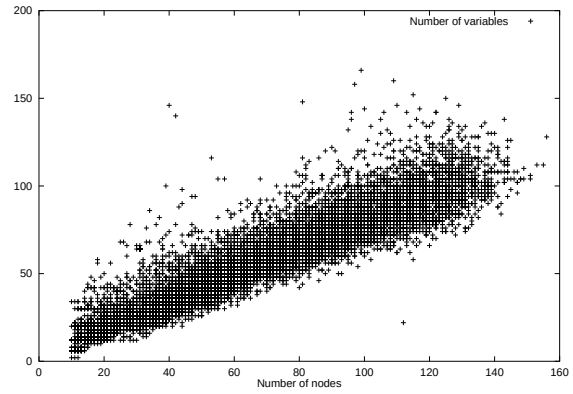


Figure 10: Number of variables in the benchmark-test-run

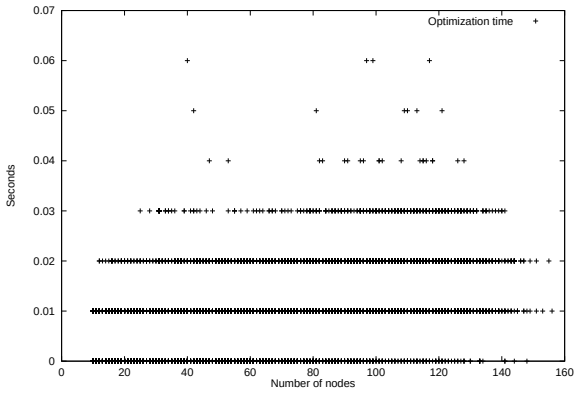


Figure 11: Optimization time in the benchmark test-run

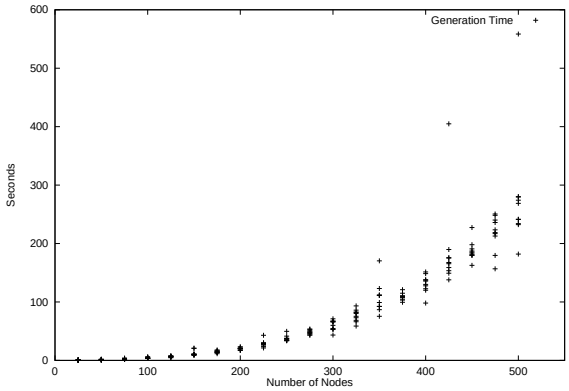


Figure 12: Generation time in the artificial test-run

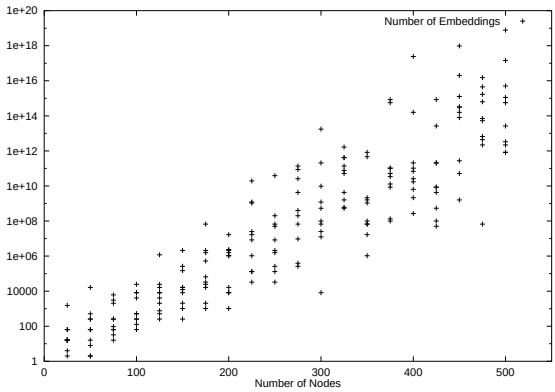


Figure 13: Number of embeddings in the artificial test-run

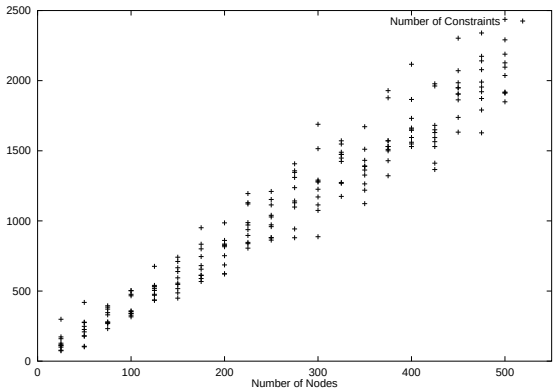


Figure 14: Number of constraints in the artificial test-run

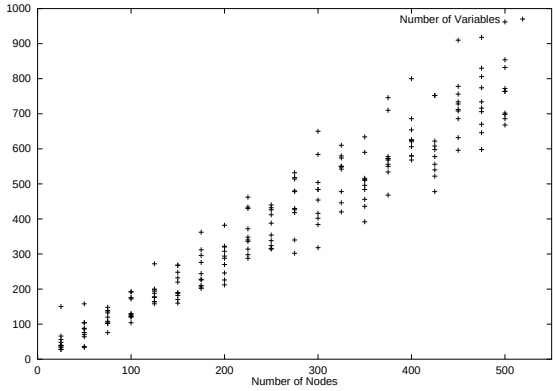


Figure 15: Number of variables in the artificial test-run

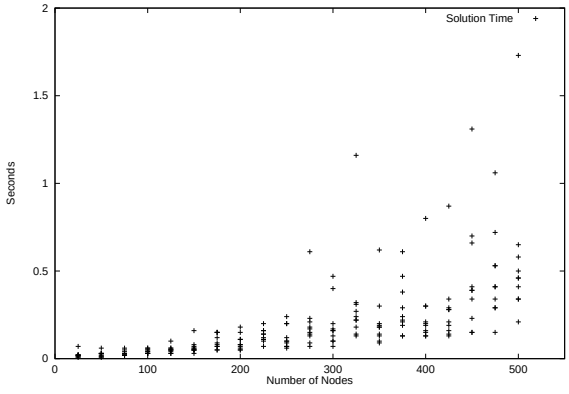


Figure 16: Optimization time in the artificial test-run

References

[1] G. Di Battista, P. Eades, R. Tamassia, and I.G. Tollis. *Graph Drawing.* Prentice Hall, 1999.

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

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

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

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

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

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

[13] G. J. Woeginger. personal communications, July 1998.

Below you find a list of the most recent technical reports of the Max-Planck-Institut für Informatik. They are available by anonymous ftp from `ftp.mpi-sb.mpg.de` under the directory `pub/papers/reports`. Most of the reports are also accessible via WWW using the URL `http://www.mpi-sb.mpg.de`. If you have any questions concerning ftp or WWW access, please contact `reports@mpi-sb.mpg.de`. Paper copies (which are not necessarily free of charge) can be ordered either by regular mail or by e-mail at the address below.

Max-Planck-Institut für Informatik
Library
attn. Birgit Hofmann
Im Stadtwald
D-66123 Saarbrücken
GERMANY
e-mail: `library@mpi-sb.mpg.de`

MPI-I-98-2-018	F. Eisenbrand	A Note on the Membership Problem for the First Elementary Closure of a Polyhedron
MPI-I-98-2-017	M. Tzakova, P. Blackburn	Hybridizing Concept Languages
MPI-I-98-2-014	Y. Gurevich, M. Veanes	Partisan Corroboration, and Shifted Pairing
MPI-I-98-2-013	H. Ganzinger, F. Jacquemard, M. Veanes	Rigid Reachability
MPI-I-98-2-012	G. Delzanno, A. Podelski	Model Checking Infinite-state Systems in CLP
MPI-I-98-2-011	A. Degtyarev, A. Voronkov	Equality Reasoning in Sequent-Based Calculi
MPI-I-98-2-010	S. Ramangalahy	Strategies for Conformance Testing
MPI-I-98-2-009	S. Vorobyov	The Undecidability of the First-Order Theories of One Step Rewriting in Linear Canonical Systems
MPI-I-98-2-008	S. Vorobyov	AE-Equational theory of context unification is Co-RE-Hard
MPI-I-98-2-007	S. Vorobyov	The Most Nonelementary Theory (A Direct Lower Bound Proof)
MPI-I-98-2-006	P. Blackburn, M. Tzakova	Hybrid Languages and Temporal Logic
MPI-I-98-2-005	M. Veanes	The Relation Between Second-Order Unification and Simultaneous Rigid E-Unification
MPI-I-98-2-004	S. Vorobyov	Satisfiability of Functional+Record Subtype Constraints is NP-Hard
MPI-I-98-2-003	R.A. Schmidt	E-Unification for Subsystems of S4
MPI-I-98-1-028	A. Crauser, K. Mehlhorn, E. Althaus, K. Brengel, T. Buchheit, J. Keller, H. Krone, O. Lambert, R. Schulte, S. Thiel, M. Westphal, R. Wirth	On the performance of LEDA-SM
MPI-I-98-1-027	C. Burnikel	Delaunay Graphs by Divide and Conquer
MPI-I-98-1-026	K. Jansen, L. Porkolab	Improved Approximation Schemes for Scheduling Unrelated Parallel Machines
MPI-I-98-1-025	K. Jansen, L. Porkolab	Linear-time Approximation Schemes for Scheduling Malleable Parallel Tasks
MPI-I-98-1-024	S. Burkhardt, A. Crauser, P. Ferragina, H. Lenhof, E. Rivals, M. Vingron	q-gram Based Database Searching Using a Suffix Array (QUASAR)
MPI-I-98-1-023	C. Burnikel	Rational Points on Circles
MPI-I-98-1-022	C. Burnikel, J. Ziegler	Fast Recursive Division
MPI-I-98-1-021	S. Albers, G. Schmidt	Scheduling with Unexpected Machine Breakdowns
MPI-I-98-1-020	C. Rüb	On Wallace's Method for the Generation of Normal Variates

MPI-I-98-1-019		2nd Workshop on Algorithm Engineering WAE '98 - Proceedings
MPI-I-98-1-018	D. Dubhashi, D. Ranjan	On Positive Influence and Negative Dependence
MPI-I-98-1-017	A. Crauser, P. Ferragina, K. Mehlhorn, U. Meyer, E. Ramos	Randomized External-Memory Algorithms for Some Geometric Problems
MPI-I-98-1-016	P. Krysta, K. Loryś	New Approximation Algorithms for the Achromatic Number
MPI-I-98-1-015	M.R. Henzinger, S. Leonardi	Scheduling Multicasts on Unit-Capacity Trees and Meshes
MPI-I-98-1-014	U. Meyer, J.F. Sibeyn	Time-Independent Gossiping on Full-Port Tori
MPI-I-98-1-013	G.W. Klau, P. Mutzel	Quasi-Orthogonal Drawing of Planar Graphs
MPI-I-98-1-012	S. Mahajan, E.A. Ramos, K.V. Subrahmanyam	Solving some discrepancy problems in NC*
MPI-I-98-1-011	G.N. Frederickson, R. Solis-Oba	Robustness analysis in combinatorial optimization
MPI-I-98-1-010	R. Solis-Oba	2-Approximation algorithm for finding a spanning tree with maximum number of leaves
MPI-I-98-1-009	D. Frigioni, A. Marchetti-Spaccamela, U. Nanni	Fully dynamic shortest paths and negative cycle detection on diagraphs with Arbitrary Arc Weights
MPI-I-98-1-008	M. Jünger, S. Leipert, P. Mutzel	A Note on Computing a Maximal Planar Subgraph using PQ-Trees
MPI-I-98-1-007	A. Fabri, G. Giezeman, L. Kettner, S. Schirra, S. Schönherr	On the Design of CGAL, the Computational Geometry Algorithms Library
MPI-I-98-1-006	K. Jansen	A new characterization for parity graphs and a coloring problem with costs
MPI-I-98-1-005	K. Jansen	The mutual exclusion scheduling problem for permutation and comparability graphs
MPI-I-98-1-004	S. Schirra	Robustness and Precision Issues in Geometric Computation
MPI-I-98-1-003	S. Schirra	Parameterized Implementations of Classical Planar Convex Hull Algorithms and Extreme Point Compuations
MPI-I-98-1-002	G.S. Brodal, M.C. Pinotti	Comparator Networks for Binary Heap Construction
MPI-I-98-1-001	T. Hagerup	Simpler and Faster Static AC^0 Dictionaries
MPI-I-97-2-012	L. Bachmair, H. Ganzinger, A. Voronkov	Elimination of Equality via Transformation with Ordering Constraints
MPI-I-97-2-011	L. Bachmair, H. Ganzinger	Strict Basic Superposition and Chaining
MPI-I-97-2-010	S. Vorobyov, A. Voronkov	Complexity of Nonrecursive Logic Programs with Complex Values
MPI-I-97-2-009	A. Bockmayr, F. Eisenbrand	On the Chvátal Rank of Polytopes in the 0/1 Cube
MPI-I-97-2-008	A. Bockmayr, T. Kasper	A Unifying Framework for Integer and Finite Domain Constraint Programming
MPI-I-97-2-007	P. Blackburn, M. Tzakova	Two Hybrid Logics
MPI-I-97-2-006	S. Vorobyov	Third-order matching in $\lambda \rightarrow$-Curry is undecidable
MPI-I-97-2-005	L. Bachmair, H. Ganzinger	A Theory of Resolution
MPI-I-97-2-004	W. Charatonik, A. Podelski	Solving set constraints for greatest models
MPI-I-97-2-003	U. Hustadt, R.A. Schmidt	On evaluating decision procedures for modal logic
MPI-I-97-2-002	R.A. Schmidt	Resolution is a decision procedure for many propositional modal logics
MPI-I-97-2-001	D.A. Basin, S. Matthews, L. Viganò	Labelled modal logics: quantifiers
MPI-I-97-1-028	M. Lermen, K. Reinert	The Practical Use of the $\mathcal{A}^*$ Algorithm for Exact Multiple Sequence Alignment