TECHNISCHE UNIVERSITÄT WIEN
Institut für Computergraphik und Algorithmen

Bend Minimization in Planar Orthogonal Drawings Using Integer Programming

Petra Mutzel and René Weiskircher

Forschungsbericht / Technical Report

TR–186–1–04–02

20. August 2004

Favoritenstraße 9-11 / E186, A-1040 Wien, Austria
Tel. +43 (1) 58801-18601, Fax +43 (1) 58801-18699
www.cg.tuwien.ac.at

Bend Minimization in Planar Orthogonal Drawings Using Integer Programming

Petra Mutzel* René Weiskircher†

Abstract

We consider the problem of minimizing the number of bends in a planar orthogonal graph drawing. While the problem can be solved via network flow for a given planar embedding of a graph G, it is NP-hard if we consider the set of all planar embeddings of G. Our approach for biconnected graphs combines an integer linear programming (ILP) formulation for the set of all embeddings of a planar graph with the network flow formulation for fixed embeddings. We report on extensive computational experiments with two benchmark sets containing a total of more than 12,000 graphs where we compared the performance of our ILP-based algorithm with a heuristic and a previously published branch & bound algorithm for solving the same problem. Our new algorithm is significantly faster than the previously published approach for the larger graphs of the benchmark graphs derived from industrial applications and almost twice as fast for the benchmark graphs from the artificially generated set of hard instances of the problem.

1 Introduction

Drawing graphs is important in many scientific and economic areas. Applications include the drawing of UML diagrams in software engineering and business process modeling as well as in the visualization of databases. A popular way of drawing graphs is representing the vertices as boxes and the edges as sequences of horizontal and vertical line segments connecting the boxes. This drawing style is called *orthogonal* drawing. A point where two segments of an edge meet is called a *bend*.

Fig. 1 shows an orthogonal drawing that represents ownership relations between several Spanish companies [HJ00]. The vertices represent the companies and there is an edge from one company to another if the first company owns a percentage of the second company.

A well known approach for drawing general graphs is the topology-shape-metrics method (see for example [DBETT99]). In the first step, the topology of the drawing is computed. The objective in this phase is to minimize the number of edge crossings. In the second step, the shape of the drawing is calculated. In the case of orthogonal drawings, the angles and the bends of the edges are computed. The objective is to minimize the number of bends for the given topology. Finally, the metrics of the drawing is computed while trying to achieve short edge lengths and small area for the given shape. In this paper, we focus on the bend minimization step (the second step). Given a planar graph, the task is to compute an orthogonal representation with the minimum number of bends.

The infinite set of different planar drawings of a graph can be partitioned into a finite set of equivalence classes called *embeddings* of a graph. An embedding defines

*Vienna University of Technology, mutzel@ads.tuwien.ac.at
†Vienna University of Technology, weiskircher@ads.tuwien.ac.at

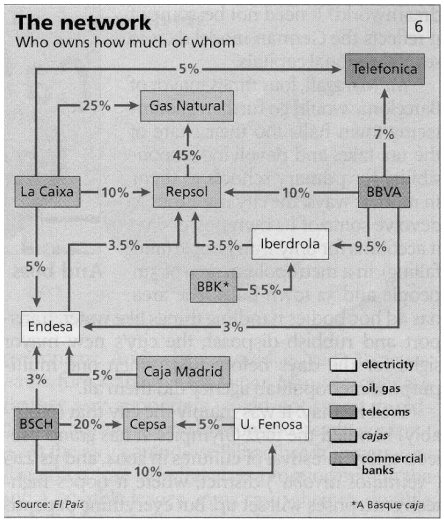


Figure 1: An orthogonal graph drawing representing ownership relations between Spanish companies

the topology of a planar drawing without assigning lengths or shapes to the edges or fixing the shapes and positions of vertices.

A *combinatorial embedding* fixes the sequence of incident edges around each vertex in clockwise order. This also fixes the list of *faces* of a drawing. The faces are the connected regions of the plane defined by a planar drawing. A *planar embedding* additionally defines the outer (unbounded) face of a planar drawing. *Orthogonal representations* are equivalence classes of orthogonal drawings, that fix the planar embedding and the bends and angles in an orthogonal drawing.

There are some results in the literature on the topic of optimizing certain functions over the set of all embeddings of a graph. Bienstock and Monma have studied the complexity of covering vertices by faces [BM88] and minimizing certain distance measures on the faces of a graph with respect to the outer face [BM89, BM90]. Tamassia presented the first algorithm for minimizing the number of bends in a planar orthogonal drawing for the case where the embedding is fixed [Tam87]. Garg and Tamassia have shown that optimizing the number of bends in an orthogonal drawing over the set of all embeddings of a planar graph is NP-hard [GT01]. In [DBLV98], Di Battista, Liotta and Vargiu show that the problem is polynomially solvable for series-parallel graphs and 3-planar graphs.

If we consider only orthogonal drawings where the bends of the edges are placed on the same grid as the vertices and all vertices occupy only one grid point, we can only represent graphs with a maximum degree of four (so called *four-graphs*). In [DL98], Didimo and Liotta present an algorithm that produces planar orthogonal drawings of four-graphs with the minimum number of bends. The running time is exponential only in the number of vertices with degree four.

Bertolazzi et al [BDBD00] have devised a branch & bound algorithm for solving the bend minimization problem over the set of all embeddings of a planar graph using SPQR-trees. In this paper, we attack the same problem using integer linear programming. To do this, we combine our integer linear program describing the set of all combinatorial embeddings of a planar biconnected graph [MW99, MW00] with a linear program that describes the set of all orthogonal representations of a planar graph with a fixed embedding. The result is a mixed integer linear program

that represents the set of all orthogonal representations for a planar biconnected graph over the set of all embeddings.

We use this new mixed integer linear program to minimize the number of bends in an orthogonal drawing over the set of all embeddings of a planar graph. Like the approach in [BDBD00], our new method can only guarantee optimality for biconnected graphs, because we also use the SPQR-tree, which is only defined for graphs that have this property. Our algorithm first computes the mixed integer linear program and then uses a commercial solver (CPLEX) to find an optimal solution. This solution is then transformed into an orthogonal representation of the graph.

We tested our approach on two different benchmark sets of graphs. The first consists of graphs derived from industrial applications and the second one of graphs computed by a graph generator. The latter set was also used in [BDBD00] to measure the performance of the branch & bound algorithm. Our new approach is faster for the large graphs in the first benchmark set than the branch & bound approach of Bertolazzi et al and about twice as fast on the graphs in the seconds benchmark set, as our computational results show.

In Section 2, we give a short overview of SPQR-trees. We present the four different types of nodes in the tree and the properties of the tree that are important in our approach. Section 3 summarizes the recursive construction of the integer linear program that describes the combinatorial embeddings of a graph.

The linear program describing the orthogonal representations of a graph for a fixed embedding is the topic of Section 4. This is basically the formulation as a linear program of a minimum cost flow problem in a special network constructed from the graph and the embedding. In Section 5, we present the new mixed integer linear program that is the result of merging the integer linear program describing the embeddings of a graph with the linear program that describes the orthogonal representations for a graph where the embedding is fixed.

The topic of Section 6 is the algorithm that we use to compute an orthogonal representation of a graph with the minimum number of bends over the set of all embeddings. The computational results we obtained by applying the algorithm to two sets containing a total of more than 12,000 benchmark graphs are given in Section 7. We compare the algorithm with a well known heuristic and with the branch & bound algorithm of Bertolazzi et al. The conclusion (Section 8) summarizes the main results and contains possible starting points for future work.

2 SPQR-Trees

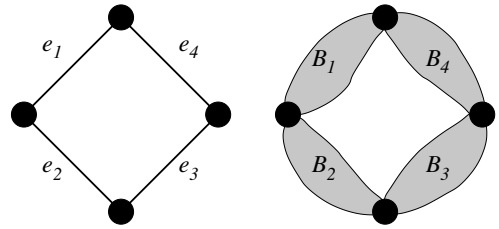
In this section, we give a brief overview of the SPQR-tree data structure for biconnected graphs. A graph is biconnected, if it is connected and can not be disconnected by deleting a vertex. SPQR-trees have been developed by Di Battista and Tamassia [DBT96]. They represent a decomposition of a biconnected graph into its triconnected components. A connected graph is triconnected, if there is no pair of vertices in the graph whose removal splits the graph into two or more components.

An SPQR-tree has four types of nodes and with each node we associate a biconnected graph which is called the *skeleton* of that node. This graph can be seen as a simplified version of the original graph and its vertices are vertices of the original graph. The edges in a skeleton represent subgraphs of the original graph. Thus, each node of the SPQR-tree defines a decomposition of the graph. The node types and their skeletons are as follows:

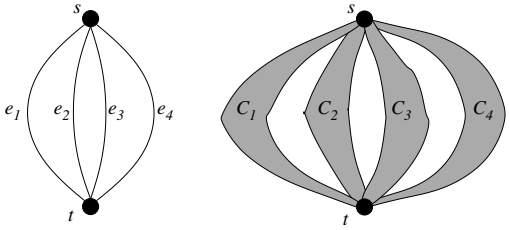
1. ***Q*-node:** The skeleton consists of two vertices connected by two edges. One of the edges represents an edge of the original graph and the other one the rest of the graph.

2. ***S*-node:** The skeleton is a simple cycle with at least three vertices.
3. ***P*-node:** The skeleton consists of two vertices connected by at least three edges.
4. ***R*-node:** The skeleton is a triconnected graph with at least four vertices.

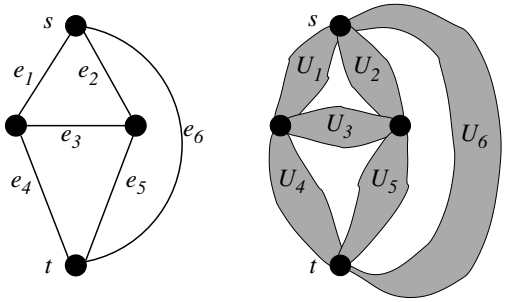
All leaves of the SPQR-tree are Q-nodes and all inner nodes S-, P or R-nodes. When we see the SPQR-tree as an unrooted tree, then it is unique for every biconnected graph. Another important property of these trees is that their size (including the skeletons) is linear in the size of the original graph and that they can be constructed in linear time [HT73, GM01]. See Figure 2 for examples of the skeletons of inner nodes of an SPQR-tree and the decomposition of the graph they define.



(a) S-node



(b) P-node



(c) R-node

Figure 2: The skeletons of the inner nodes of an SPQR-tree together with the decomposition of the graph they define

As described in [DBT96], SPQR-trees can be used to represent the set of all combinatorial embeddings of a biconnected planar graph. Every combinatorial em-

bedding 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 skeletons of S- and Q-nodes are simple cycles, so they have only one embedding. But the skeletons of the R- and P-nodes have at least two different combinatorial embeddings. This is the reason why they determine the embedding of the graph and we call these nodes the *decision nodes* of the SPQR-tree.

3 The ILP-Formulation Describing the Set of All Embeddings

The integer linear program (ILP) suggested in [MW99] describing the set of all combinatorial embeddings of a planar graph is constructed recursively using the SPQR-tree data structure. Because SPQR-trees are only defined for biconnected graphs, the same is true for the ILP. We construct the program recursively by splitting the SPQR-tree into smaller SPQR-trees, constructing ILPs for these smaller trees, and then merging them into an ILP for the original graph. The basis of the recursive construction are SPQR-trees that have only one inner node (S-, P- or R-node).

3.1 The ILP for Graphs Where the SPQR-tree Has Only One Inner Node

If the SPQR-tree for a graph G has exactly one inner node μ, then G is isomorphic to the skeleton S_μ of μ. It follows that the graph is either a simple cycle (μ is an S-node), a triconnected graph (μ is an R-node) or consists of two vertices connected by at least three edges (μ is a P-node).

In all three cases, the set of combinatorial embeddings is quite easy to describe. If the graph is a cycle, it has only one combinatorial embedding. If it is a triconnected graph, it has two embeddings that are mirror images of each other. If it consists of two vertices and at least three edges, the embeddings are determined by the different circular permutations of the edges incident to the two vertices.

The variables of the ILP correspond to the set of directed cycles of the graph, that are face cycles in at least one embedding. A directed cycle is a face cycle in an embedding, if the area of the plane on the right side of the cycle is empty in every planar drawing that realizes the embedding. First we describe the three possible cases for graphs whose SPQR-tree has only one inner node together with the ILPs that describe their embeddings. We distinguish the three cases by the type of the only inner node v of the SPQR-tree.

If v is an S-node, then G is a simple cycle. Therefore, G contains exactly two directed cycles and both are face cycles in the only combinatorial embedding of G. It follows that the corresponding ILP has two variables that are both equal to one in the only solution.

If v an R-node, the graph G is triconnected. A triconnected graph has two combinatorial embeddings that are mirror images of each other. This means that for any directed cycle c that is a face cycle in the first embedding, the directed cycle $\bar{c}$ passing the same edges as c in the opposite direction is a face cycle of the other embedding.

Let l be the number of faces in an embedding of G (note that $l = m - n + 2$ if m is the number of edges in G and n the number of vertices since G is planar). Then there are $2l$ directed cycles in G that are face cycles in an embedding. If we write down the two embedding of G as two vectors $\vec{s_1}, \vec{s_2} \in \{0,1\}^{2l}$, then each of

the vectors contains l ones. Any entry with value one in $\vec{s_1}$ has value zero in $\vec{s_2}$ and vice versa. Therefore it is straight forward to describe the embeddings of G as an ILP.

If v is a P-node, the graph G consists of two vertices connected by k edges with $k \geq 3$. Since combinatorial embeddings can be defined by the circular sequence of the edges around each vertex, G has $(k-1)!$ different embeddings. Each pair of edges in G corresponds to two directed cycles and all the cycles are face cycles in at least one of the embeddings. Therefore, there are $k^2 - k$ variables in the ILP.

We realized that in this case, the set of embeddings of G can be interpreted as the set of Hamiltonian tours in a complete directed graph H [MW00]. The graph H has one vertex for every edge of G and one edge for each directed cycle. The edge in H that corresponds to the directed cycle c in G connects the vertices that correspond to the two edges that form the cycle c.

Figure 3 shows this correspondence. The numbers of the edges of the graph on the left correspond to the numbers of the vertices in the complete directed graph H on the right. Edge (v_i, v_j) in H corresponds to the cycle in G that passes edge e_i from v_a to v_b and edge e_j from v_b to v_a. The cycles that are face cycles in the embedding of G on the left correspond to the thick edges in the graph H on the right. Thus, the embedding of G corresponds to the tour in H consisting of the thick edges. Note that the sequence of the edges in clockwise order around vertex v_a corresponds to the sequence of the vertices defined by the tour.

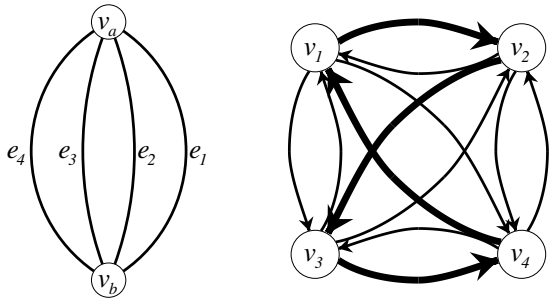


Figure 3: A graph where the only inner node in its SPQR-tree is a P-node and the corresponding graph H

Because of the correspondence between the Hamiltonian tours in H and the embeddings of G, we can use the ILP-formulation for the asymmetric traveling salesman problem (ATSP) to describe the set of embeddings of G. We use the formulation of [CDT95], which consists of a linear number of *degree constraints* and an exponential number of *subtour elimination constraints.* The degree constraints say that each edge in G is contained in exactly two face cycles of each embedding, once for each direction of the edge. The subtour elimination constraints say that for each proper subset E' of the edges of G, there must be at least one face cycle that contains an edge of E' and an edge not contained in E'.

Because the number of subtour elimination constraints grows exponential with the number of edges in G (there is one constraint for each proper subset of the edges of G), we define the ILP for a graph whose SPQR-tree has a P-node as the only inner node just as the set of degree constraints. To cope with the subtour elimination constraints, we use the same approach that is used for the ATSP-problem: We separate the subtour elimination constraints during the optimization process.

So we first compute a solution vector $\vec{s}$ for the problem without the subtour elimination constraints and then check if $\vec{s}$ violates any of the subtour elimination

constraints. If this is not the case, we have found a valid solution. Otherwise, we add the violated subtour elimination constraints to the set of constraints and re-optimize. To check if there is a violated subtour elimination constraint, we can find a minimum cut in the graph H where the weight of each edge is defined by the corresponding component of the vector $\vec{s}$.

If we define the value of the cut as the sum of the weights of all the edges that leave the cut, then we have found a violated subtour elimination constraint if the cut has weight smaller than one. Otherwise, we know that $\vec{s}$ satisfies all subtour elimination constraints and degree constraints. Every integer vector that satisfies the degree constraints and the subtour elimination constraints represents a combinatorial embedding of G.

3.2 The ILP for Graphs whose SPQR-tree Has More than One Inner Node

If the SPQR-tree of G has more than one inner node, we split the tree at an arbitrary decision node (R- or P-node). This is done as shown in Fig. 4. In this figure, the split node is v. We split all the edges that connect v to the rest of the tree by inserting two new Q-nodes per edge. The reason why we have to insert these nodes is that in every SPQR-tree, the leaves of the tree must be Q-nodes. The result of the splitting operation is a set of SPQR-trees called *split trees*. The graph represented by a split tree is called *split graph*. Each split graph can be obtained from G by replacing subgraphs connected to the rest of G only via a pair of vertices by an edge. These new edges, that we call *virtual edges*, correspond to the Q-nodes we add in the splitting process. Figure 5 shows an example for a graph and its three split graphs. The grey edges are the virtual edges of the split graphs.

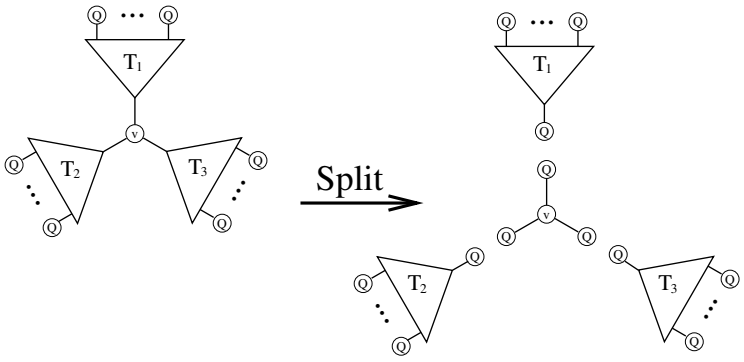


Figure 4: Splitting an SPQR-tree at an inner node

One of the resulting split trees contains v as the only inner node and so the ILP for the corresponding split graph has already been defined in the last section. We call this split graph the *center split graph*. In Figure 5, the center split graph is G_0. All other split trees contain at least one decision node less than the original tree, because they do not contain v. We continue the recursive splitting process until we have only SPQR-trees with just one inner node. This is always possible because any SPQR-tree with more than one inner node contains at least one decision node (two S-nodes can never be adjacent).

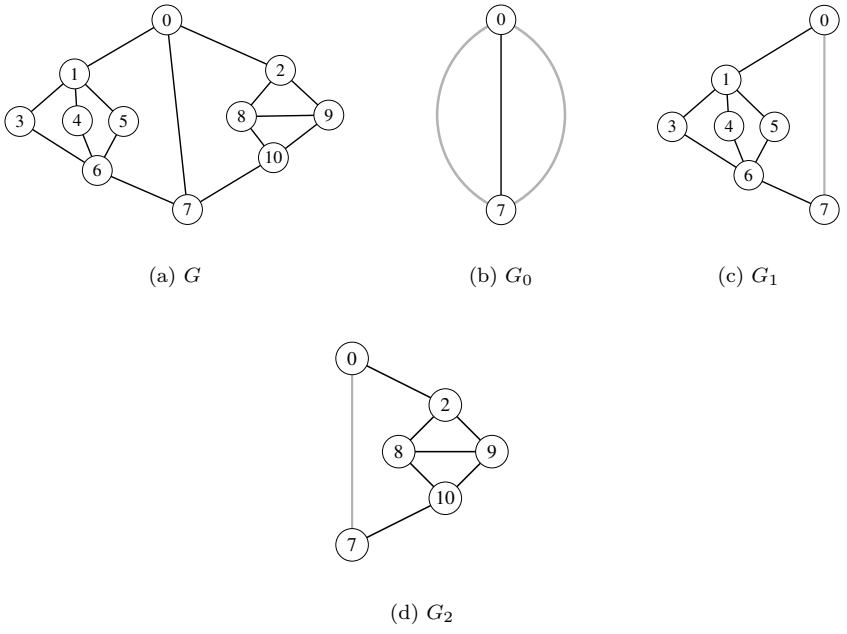
After recursively computing the ILPs for the split graphs, we merge them into an ILP for the original graph G. To achieve this, we need to lift the constraints computed for the split graphs, since G contains more potential face cycles than the split graphs. Therefore, the ILP describing its embeddings has more variables and

thus a higher dimension. The crucial fact in this step is to find for each cycle c in a split graph the set of cycles of G that are represented by c.

Let $G_0, \ldots, G_k$ be the split graphs of G and G_0 the center split graph. Let C_i for $0 \leq i \leq k$ be the set of cycles in G_i that are represented by a variable in the corresponding ILP and $C = \bigcup_{i=0}^{k} C_i$. Then we can split C into two sets: The set C_L of cycles that are also contained in G (because they do not include a virtual edge) and the set C_M of cycles that are not cycles in G (because they contain a virtual edge).

The cycles in C_L are also represented by variables in the ILP for G. We use the cycles in C_M to construct cycles in G. Every cycle c in $C_i \cap C_M$ with $1 \leq i \leq k$ contains exactly one virtual edge. If we remove this edge, we get a path p in G_i connecting the two vertices of G_i that it shares with G_0. If we take a cycle of $C_0 \cap C_M$ and replace each virtual edge by a path p in the corresponding split graph obtained from a cycle in C_M, we get a cycle in G that is a face cycle in at least one embedding of the graph (this can be shown by structural induction).

Consider for example Figure 5. We construct cycles in G by taking a cycle in the center split graph G_0 and replacing its virtual edges by paths in the corresponding split graphs. We choose the directed cycle c_1 that passes the left virtual edge of G_0 from vertex 0 to vertex 7 and the non-virtual edge from 7 to 0. The set L_{c_1} consists of the three cycle $(7, 0, 1, 3, 6)$, $(7, 0, 1, 4, 6)$ and $(7, 0, 1, 5, 6)$ (given by the sequence of vertices on the cycle). We combine c_1 with the cycle c_2 in G_1 given by the sequence $(0, 1, 3, 6, 7)$ of vertices. The set L_{c_2} consists of the cycles $(0, 1, 3, 6, 7)$, $(0, 1, 3, 6, 7, 10, 8, 2)$ and $(0, 1, 3, 6, 7, 10, 9, 2)$. Note that there are no cycles in L_{c_2} using edge $(8, 9)$ because such a cycle can never be a face cycle. The combination of the cycles c_1 and c_2 results in cycle $(0, 1, 3, 6, 7)$ in G.



(a) G (b) G_0 (c) G_1

(d) G_2

Figure 5: A graph G and its splitgraphs G_0, G_1 and G_2

The variables in the ILP for G correspond to the cycles in C_L and to the cycles in G that we can construct by combining cycles in C_M. In general, a cycle in C_M is used to construct several cycles in G. We store for each cycle c in C_M the set L_c

of cycles in G constructed by combining c with other cycles. For each cycle in C_L, we define L_c as the cycle itself. These sets are used for lifting the constraints of the ILPs of the split graphs.

Let c be a cycle in a split graph that is represented by variable x_c in the corresponding ILP. The set L_c defines a set L_{x_c} of variables in the ILP for the original graph. We use these sets L_{x_c} to lift the constraints of the ILPs of the split graphs. We simply replace each variable x_c in each constraint computed for a split graph by the sum of the variables in L_{x_c}.

Remember that we have not explicitly computed the subtour elimination constraints for graphs whose only inner node in the SPQR-tree is a P-node. Instead, there is a complete bidirected graph H for each such graph where we can find violated subtour elimination constraints by computing a minimum cut. When we compute the ILP for G from the ILPs of the split graphs, we have to updated the set of cycles that are represented by each edge of H.

If the only inner node of an SPQR-tree is a P-node, every edge e in H represents exactly one directed cycle c of the graph. The weight of e is the value of the corresponding component in the current solution vector corresponding to c. When we compute the ILP of the original graph from the ILPs of the split graphs, we assume that every edge in H represents a set of cycles. If the only inner node of the split graph is a P-node, each of these sets has only one element. Let H be a complete bidirected graph computed for the separation of subtour elimination constraints in a split graph and e an edge in H representing the set C_e of cycles. When we have computed the composed cycles of the original graph, we replace each cycle c in C_e by the set L_c of composed cycles generated using c. Thus, the new set of cycles represented by e consists of the set of all the cycles in G constructed from cycles in C_e.

After we have computed the new variables, lifted the constraints of the split graphs and updated the complete bidirected graphs computed for each P-node skeleton, we add some additional constraints. The first type of constraints says that out of all cycles that are represented by the same cycle in a split graph, at most one can be a face cycle in any embedding of the original graph. This is true because there is a split component of a split pair of the graph that all these cycles pass in the same direction.

To get an intuition why two such cycles can never be face cycles in the same embedding consider Figure 6. We assume that v_1 and v_2 are a split pair of the graph with the split components G_1 and G_2. We also assume that the two directed cycles c_1 and c_2 use the same path p in G_1 but different paths (p_1 and p_2) in G_2. If we assume that both c_1 and c_2 are face cycles in the same embedding Π of G, then we have a contradiction: Since c_1 is a face cycle, the area right of c_1 must be empty in Π and so the edges of p_2 must be left of p_1. But since c_2 is also a face cycle, the area to the right of c_2 must be empty and the path p_1 must be left of p_2.

The second type of constraints fixes the number of cycles in each embedding that contains edges from more than one split graph. This number is equal to the number of face cycles in an embedding of the center split graph that contain virtual edges. Using structural induction, we can show that the resulting ILP is correct and that its variables correspond to the set of cycles that are face cycles in at least one embedding of the graph. The last fact is important, because it is the reason why the number of variables is still manageable for practical problem instances (see Section 7).

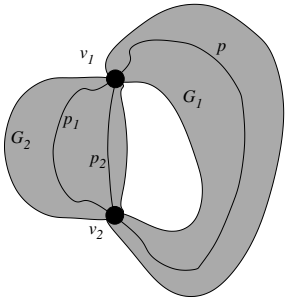


Figure 6: Two cycles that pass a split component in the same direction can not both be face cycles in the same embedding

4 The Linear Program Describing Orthogonal Representations for a Fixed Embedding

Orthogonal representations not only fix the embedding of a graph but also the number, type and sequence of the bends on each edge and the angles between edges incident to the same vertex in an orthogonal drawing. They do not fix the lengths of the edge segments in the drawing. The first efficient algorithm for computing an orthogonal representation of a graph with the minimum number of bends for a fixed planar embedding was presented by Tamassia [Tam87]. This algorithm constructs a flow network using the planar embedding and then computes a minimum cost flow in this network. This flow can be translated into an orthogonal representation of the graph with the minimum number of bends for the fixed embedding.

The drawback of the original method of Tamassia is that it can not deal with vertices of degree greater than four. Some modifications of the algorithm have been published that get over this constraint. The approach that we use implements the *podevsnef* drawing convention (planar orthogonal drawings with equal vertex size and non-empty faces) first defined in [FK96]. According to this convention, the vertices are drawn as boxes of equal size and the edges are positioned on a finer grid than the vertices. Because of this modification, more than one edge can be incident to each of the four sides of a vertex (see Fig. 7 for an example of a podevsnef drawing).

Bertolazzi et al describe a minimum cost flow network N that can be used to compute an orthogonal representation in a simplified podevsnef model with the minimum number of bends for a fixed embedding [BDBD00]. The model is simplified, because whenever edges run parallel, the first bend of the rightmost edge in the bundle is a bend to the right. Another property of the simplified model is that a vertex with degree at least four has always at least one edge attached to each side. This is the case in Fig. 7. The network N for G contains one node for every vertex of G (called *v-nodes*) and one vertex for every face cycle of the given embedding (called *f-nodes*).

The basic idea of the network is that the flow on its arcs corresponds to bends on edges and to the angles between neighboring edges incident to the same vertex. One unit of flow corresponds to an angle of 90 degrees. The supply of flow assigned to each node and the capacity of the edges guarantee that a feasible flow corresponds to an orthogonal representation of the graph. The costs are assigned to the arcs such that the cost of a flow is equal to the number of bends in the corresponding orthogonal representation.

Let b be the bijection that maps the vertices of G to the v-nodes of N and the

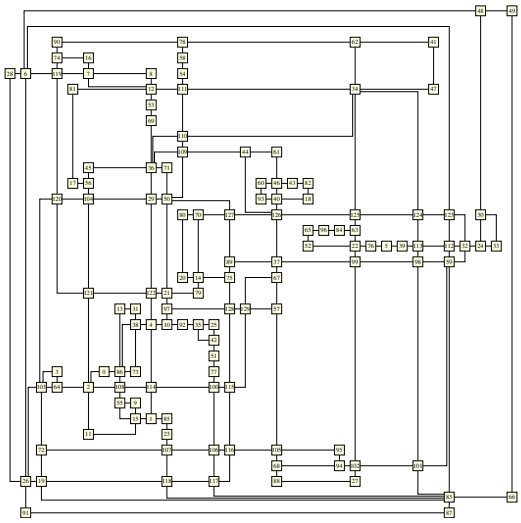


Figure 7: A podevsnef drawing of a graph.

face cycles of the planar embedding to the f-nodes. Then there is an arc between the v-node v_1 and the f-node v_2 if the vertex $b^{-1}(v_1)$ is on the boundary of the face $b^{-1}(v_2)$. The direction of this arc depends on the degree d of $b^{-1}(v_1)$. If d is at most four, the arc is directed towards v_2 and otherwise towards v_1. The set E_{vf} is the set of all arcs that start in a v-node and end in an f-node, while the set E_{cf} is the set of arcs that start in an f-node and end in a v-node of N.

There is an arc from the f-node v_3 to the f-node v_4 if the two face cycles $b^{-1}(v_3)$ and $b^{-1}(v_4)$ share an edge. So we have a pair of arcs running in opposite directions for each pair of f-nodes corresponding to face cycles that share an edge of G. The set of these arcs is called E_{ff}.

The supply of a v-node v is $4 - deg(b^{-1}(v))$. The reason is that the sum of the angles between neighboring edges incident to the vertex $b^{-1}(v)$ must be 360 degrees (remember that each unit of flow corresponds to 90 degrees). If we define the degree of a face cycle as the number of its edges, then the supply of a node w in N corresponding to an inner face cycle $b^{-1}(w)$ is also $4-deg(b^{-1}(w))$ and if $b^{-1}(w)$ is the outer face cycle, the supply is $-4 - deg(b^{-1}(w))$. This guarantees that the flow in N describes shapes for the face cycles that correspond to simple polygons of vertical and horizontal line segments.

Each unit of flow on the edges in E_{ff} has cost one because it corresponds to a bend in the orthogonal representation. The same holds for edges in E_{fv} because flow on these edges corresponds to a zero degree angle that causes a bend on one of the two edges that run parallel. The correspondence between the flow in N and an orthogonal representation with the number of bends equal to the cost of the flow is described in more detail in [BDBD00].

We used this network and transformed it into a linear program. There is one variable f_e for each arc e in the network that represents the amount of flow routed via e. One constraint for each vertex in the network makes sure that the outgoing amount of flow minus the incoming amount is equal to the supply of the node. Some nodes in the network have negative supply, and thus consume flow. We have one constraint for each arc that says that the flow on the arc must be non-negative and also upper bounds on the flow on arcs in E_{vf} and E_{fv}. The objective function minimizes the sum of the amount of flow over each arc multiplied by the cost of

the arc. An optimal solution represents a minimum cost flow in N and thus an orthogonal representation with the minimum number of bends.

LP 1 shows the corresponding linear program. The set E_N is the set of arcs in the network, V the set of vertices in G and F the set of faces in the embedding. The face f_o is the outer face of the embedding. The degree of a face is defined as the number of edges that form the boundary of the face. The variable f_e denotes the flow on arc e. The constraints guarantee that the solution corresponds to an orthogonal representation of G.

LP 1

$$\min \quad \sum_{e \in E_N} cost(e) \cdot f_e$$

subject to

$$\begin{aligned}
\sum_{e=(v,w)\in E_N} f_e - \sum_{e=(w,v)\in E_N} f_e &= 4 - deg(b^{-1}(v)) && \forall \text{ nodes } v \text{ with } b^{-1}(v) \neq f_o \\
\sum_{e=(b(f_o),w)\in E_N} f_e - \sum_{e=(w,b(f_o))\in E_N} f_e &= -4 - deg(f_o) && \\
f_e &\leq 4 - deg(b^{-1}(v)) && \forall\, e = (v,w) \in E_{vf} \\
f_e &\leq 1 && \forall\, e \in E_{fv} \\
f_e &\geq 0 && \forall\, e \in E_N
\end{aligned}$$

5 The Mixed Integer Linear Program Describing the Set of All Orthogonal Representations of a Graph

The flow network N of the last section describing the set of orthogonal representations of a graph for a fixed embedding contains one f-node for every face of the embedding. When we want to optimize over the set of all embeddings of a graph, we do not know at the beginning which cycles will be face cycles in an optimal solution. Therefore, we construct a new network N', where we have one c-node for every cycle in the graph, that is a face cycle in at least one embedding. These nodes play a similar role to the f-nodes in the LP for a fixed embedding in the previous section. The set of cycles that are face cycles in at least one embedding of the graph corresponds to the set of variables in our ILP from Section 3 that describes the set of all embeddings of a graph.

In a solution of the embedding ILP, the variable corresponding to a cycle has value one if this cycle is a face cycle in the embedding represented by the solution and zero otherwise. The capacities of the arcs in the network N' depend on the values of the cycle variables. If the cycle-variables represent an embedding Π of the graph, then the set of feasible flows in N' corresponds to the set of feasible flows in the network N constructed for embedding Π. Let A be the set of arcs incident to the c-node for cycle c in N' and the variable for c in the embedding ILP be zero. Then all arcs in A must have capacity zero. This has the same effect on the flow as removing the c-node from the network.

We first compute the capacities of the arcs and the demand of each c-node analogously to the corresponding values for the f-nodes in the network N. Then we multiply the amount of flow produced or consumed by a c-node with the value

of the corresponding variable in the ILP. This ensures that vertices in N' that correspond to cycles in G that are not face cycles do not produce or consume flow.

Any arc that starts or ends at a c-node has capacity zero if the c-node corresponds to a cycle whose ILP-value is zero. If the capacity of the arc is limited even if the corresponding cycle is a face cycle, we can just multiply this limit with the ILP-value of the cycle to achieve this behavior. The arcs in the network N that connect two f-nodes have unlimited capacity. But we can easily compute an upper bound f_{max} for the flow produced in the whole network N, for example by computing the sum of all positive supplies in the network. The resulting value can be used as the upper bound on the flow on any arc. For each arc a in N' connecting two c-nodes v_1 and v_2, we set the capacity to the minimum of $f_{max}x_i$ where x_i is the binary variable in the embedding ILP for the cycle corresponding to node v_i. This guarantees that the flow on a is zero if at least one of the cycles represented by the nodes v_i is not a face cycle in the represented embedding.

In this way, the capacities of the arcs and the amount of flow produced and consumed by the vertices in N' depend on the values of the cycle variables in the ILP. We transform N' into a linear program and merge it with the ILP that represents the embeddings of the graph. Since we need to know the cycle chosen as the outer face cycle, we introduce an outer face variable for each cycle and add constraints that guarantee that exactly one of the cycles chosen as face cycles is chosen as the outer face cycle. The result is a mixed integer linear program (MILP), where an optimal solution corresponds to an orthogonal representation with the minimum number of bends over the set of *all* embeddings of the input graph.

MILP 1 is the resulting mixed integer linear program. We omitted the constraints that define the embedding because they are defined recursively and are not the main topic of this paper. Again, variable f_e denotes the flow on arc e. The set C is the set of cycles in G that are face cycles in at least one embedding. For each of these cycles c, the variable x_c is one if c is a face cycle and variable o_c is one if c is the outer face cycle. The set E_{cc} is the set of arcs that connect two c-nodes. Arcs in E_{vc} start in a v-node and end in a c-node while the arcs in E_{cv} start in a c-node and end in a v-node. The expression $len(c)$ denotes the number of edges in cycle c. The function b is the bijection from the vertices of G to the v-nodes in N' and from the directed cycles in C to the c-nodes of the network.

MILP 1

$$\min \quad \sum_{e \in E_N} cost(e) \cdot f_e$$

subject to

$$\sum_{c \in C} o_c = 1 \tag{1}$$

$$x_c - o_c \geq 0 \qquad \forall\, c \in C \tag{2}$$

$$\sum_{e=(v,w) \in E_N} f_e - \sum_{e=(w,v) \in E_N} f_e = 4 - deg(v) \qquad \forall\, v \in V \tag{3}$$

$$\sum_{e=(c,w) \in E_N} f_e - \sum_{e=(w,c) \in E_N} f_e = x_c(4 - len(c)) - 8o_c \qquad \forall\, c \in C \tag{4}$$

$$f_e \leq x_c(4 - deg(v)) \qquad \forall\, e = (v,c) \in E_{vc} \tag{5}$$

$$f_e \leq x_c \qquad \forall\, e = (c,v) \in E_{cv} \tag{6}$$

$$f_e \leq x_{c_1} f_{max} \qquad \forall\, e = (c_1,c_2) \in E_{cc} \tag{7}$$

$$f_e \leq x_{c_2} f_{max} \qquad \forall\, e = (c_1,c_2) \in E_{cc} \tag{8}$$

$$f_e \geq 0 \qquad \forall\, e \in E_N \tag{9}$$
$$x_c, o_c \in \{0,1\} \qquad \forall\, c \in C \tag{10}$$

Constraint 1 says that there is exactly one cycle chosen as the outer face cycle and the constraints of type 2 guarantee that this cycle is chosen among the face cycles. Constraint 3 is the same as in the LP of the previous section because the supply of the nodes representing vertices of G is independent of the chosen embedding. The node in N' corresponding to a cycle that is not chosen as a face cycle does not consume or supply any flow since its cycle variable and its outer face variable are both zero. This is guaranteed by Constraint 4. If the outer face variable of a cycle is one, the constraint sets its supply to $-4 - len(c)$ and if the cycle variable is one but the outer face variable is zero to $4 - len(c)$.

The constraints of type 5, 6, 7 and 8 make sure that the arcs incident to c-vertices where the corresponding cycle is not a face cycle in the chosen embedding have capacity zero. If a cycle is chosen as face cycle, the arcs incident to the corresponding c-node in N' have either the same capacities as in the network N of the previous section or if the capacity in N is unbounded (for the arcs in E_{cc}), the capacity is now set to an upper bound on the flow in the network.

6 The Algorithm for Minimizing the Number of Bends

The algorithm first computes the recursive ILP describing the set of all combinatorial embeddings of the graph. This also gives us the set of cycles of the graph that are face cycles in at least one embedding. This information is then used for computing the network N' and the corresponding MILP. We use CPLEX (version 6.5) to compute a solution and then check if there are any violated subtour elimination constraints by computing a minimum cut in the ATSP-graph computed for each P-node skeleton. If we find a violated constraint, we add it to the MILP and re-optimize. When we have found a feasible solution, we transform it into an orthogonal representation of the graph.

To improve the performance of the algorithm, we modified the MILP slightly. For example, we only need outer face variables for half of the cycles. The reason is that for every cycle c represented by a variable in the embedding ILP, the cycle $\bar{c}$ passing the same edges in the opposite direction is also represented by a variable. The orthogonal representations we exclude by introducing outer face variables only for one direction of each undirected cycle are mirror images of other orthogonal representations that can still be represented. Of course, every orthogonal representation has the same number of bends as its mirror image. We also hard-coded a complete description of the set of embeddings for P-node skeletons with less than five vertices into our program to reduce the need for separating subtour elimination constraints.

7 Computational Results

To test our approach, we used two sets of benchmark graphs. The first was introduced in [DBGL+97] and consists of 11,529 graphs that either come from industrial applications or were derived from such graphs by introducing small changes. We call this set the *real world set*. The second set consists of randomly generated graphs that were used in [BDBD00] to test the performance of the branch & bound approach for minimizing the number of bends. We call this set the *artificial set*.

The graphs in the artificial set are already biconnected and planar, so we can directly apply our new algorithm for minimizing the number of bends as well as the branch & bound algorithm. The majority of the graphs in the real world set are not planar and biconnected. Therefore, we used a standard approach to transform them into planar biconnected graphs whose drawing can be easily transformed into a drawing of the original graph.

This is done as follows: We first determined the topology of the graphs using the standard planarization method implemented in the AGD-library [AGD99]. This method computes an embedding of a planar subgraph of the original graph and then inserts the missing edges one by one into the current embedding. The crossings produced by inserting an edge are replaced by artificial vertices with degree four. If the resulting graph was not biconnected, we introduced new edges while maintaining planarity using the heuristic presented in [FM98] that is also implemented in the AGD-library. Note that these operations can increase the number of vertices and edges of a graph considerably.

First, we compared the results produced by our algorithm for minimizing the number of bends over all embeddings to the results computed by a popular heuristic. This heuristic chooses an arbitrary embedding for the graph and then computes a minimum cost flow in the network given in section 4. The flow defines an orthogonal representation of the graph with the minimum number of bends for the chosen embedding.

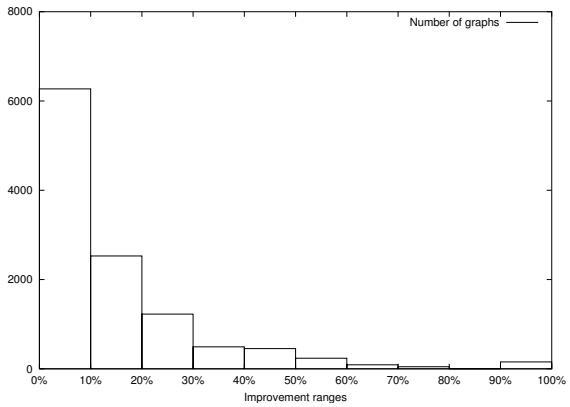
Let h be the number of bends in the orthogonal representation computed by the heuristic and o the number of bends in an orthogonal representation with the minimum number of bends over all embeddings. For each graph in the benchmark sets, we computed the following value: $\frac{h-o}{h}100\%$. This is the percentage of the improvement we get using an optimal algorithm.

We broke the set of all graphs of each benchmark set into 10 subsets. The first subset contained the graphs where the improvement was smaller than 10 percent, the second subset the graphs where the improvement was at least 10 and smaller than 20 percent and so on. The last subset contained the graphs where the improvement was at least 90 percent. Note that the improvement is 100 percent if the heuristic solutions contains bends while the optimum solution contains no bends.

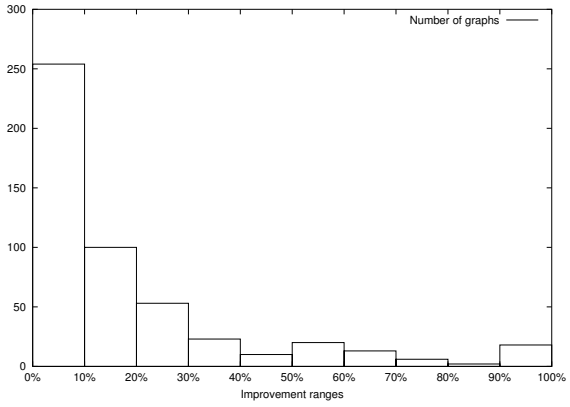
The x-axes in Figure 8 show the improvement ranges while the height of the boxes corresponds to the number of graphs that fall into that range. The diagram on the left shows the data for the real world graphs while the diagram on the right shows the data for the artificial graphs. Both diagrams are remarkably similar. For about half of all graphs, optimizing over all embeddings results in a significant reduction of the number of crossings (at least 10 percent).

We compared the average running times for graphs with the same number of vertices of our new algorithm (MIX) and the branch & bound algorithm (B&B) from [BDBD00]. Both algorithms are written in C++ and were compiled with the flag `-O` using gcc version 2.95.1. The algorithms ran on a Sun Enterprise 450 Model 4400 with four Sun UltraSPARC-II 400 MHz CPUs and 4 GB of memory. Figure 9 shows the corresponding diagrams for the real world graphs (Figure 9(a)) and for the artificial graphs (Figure 9(b)). The x-axes show the number of vertices while the y-axes give the average time in seconds needed for a problem instance. Note that the times given for MIX include the time needed for constructing the recursive ILP, the flow network and the complete MILP.

Figure 9(a) shows that B&B is faster for graphs with up to 120 vertices, but for larger graphs, our new algorithm is faster. E.g. for one graph with 130 vertices and 205 edges, B&B needed 5244 seconds while MIX found an optimal solution in only 108 seconds. The corresponding drawing is shown in Figure 7 on page 11. Our integer linear program for this graph has 10,059 rows and 3473 columns. The number of binary variables is 333. In the preprocessing step, CPLEX eliminated



(a)



(b)

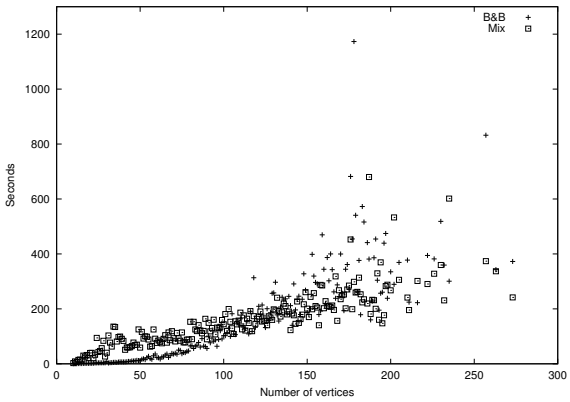
Figure 8: Improvement ranges of our algorithm compared to the heuristic for the real world benchmark set (a) and the artificial benchmark set (b).

4920 rows and 544 columns. The resulting matrix has 15,974 non-zeroes. The branch & bound tree computed by CPLEX has 282 nodes, the number of simplex iterations is 32,512. The solver generated 20 flow cover cuts in the root node. These are valid for all nodes in the tree. Flow cover cuts can be used whenever the capacity of an arc in a flow network depends on a binary variable [PRW85, GNS99].

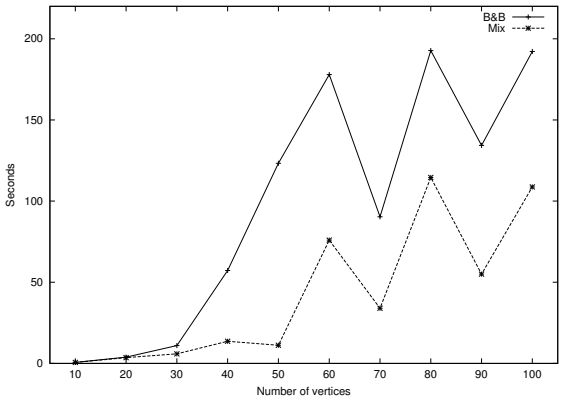
There were 197 graphs in the real world benchmark set, that B&B could not solve in one hour computation time. Our algorithm could not solve 25 graphs within the same time limit. Another interesting fact is that we only had to add subtour elimination constraints and re-optimize for six graphs out of 11,529. We never had to add more than one subtour elimination constraint.

As Figure 9(b) shows, the speed advantage of our new algorithm is very pronounced for the artificial graphs. The branch & bound algorithm needs on average almost twice as long to compute an optimal solution compared to our new method.

Figure 10 shows the average number of embeddings for graphs with the same number of vertices for the real world benchmark set and the artificial benchmark



(a)



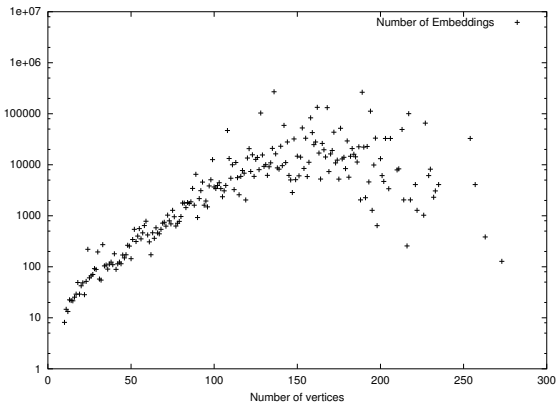
(b)

Figure 9: Runtime comparison of our new algorithm with the branch & bound algorithm for the real world benchmark set (a) and the artificial benchmark set (b)

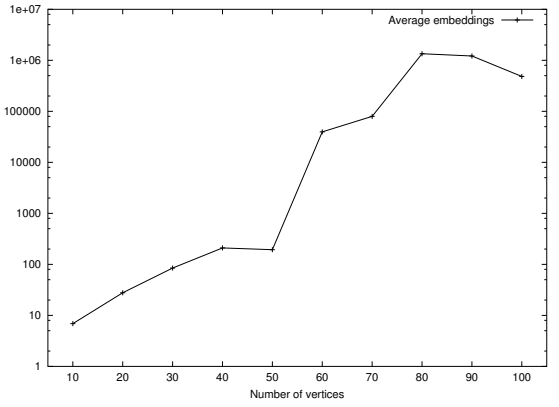
set. Again, the x-axes show the number of vertices. The y-axes have a logarithmic scale and show the average number of embeddings for the graphs.

Figure 10(a) shows the number of embeddings of the real world graphs. Until about 150 vertices, the average number of embeddings grows exponentially with the size of the graphs (remember that the y-axis is logarithmic). The reason for the drop in the number of embeddings for larger graphs is the planarization method. Graphs where many edges have to be deleted to obtain a planar graph tend to have large three-connected components after the planarization method is applied because this method replaces crossings with vertices of degree four.

The growth of the average number of embeddings for the artificial graphs is also roughly exponential until about 80 vertices, as Figure 10(b) shows. The number of embeddings varies widely for graphs with the same number of vertices because the set was generated using five different settings for the parameters of the generation algorithm that influence the number of embeddings. The details of the generation algorithm and the parameter settings can be found in [BDBD00].



(a)



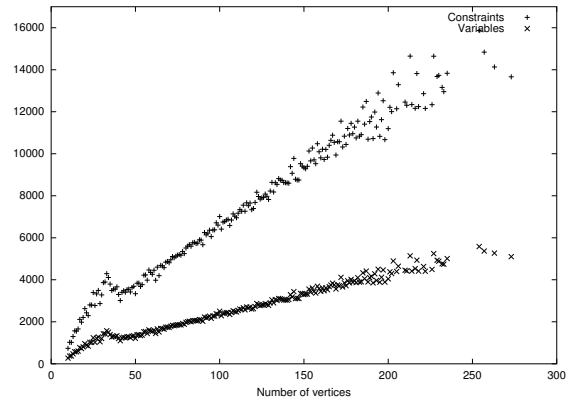
(b)

Figure 10: The average number of embeddings for graphs in the real world benchmarks set (Figure 10(a)) and the artificial benchmark set (Figure 10(b))

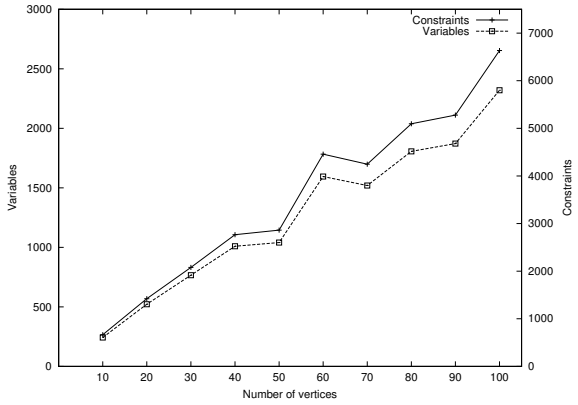
Figure 11 shows the average number of constraints and embeddings in the mixed integer linear programs computed by our algorithm for the real world graphs (Figure 11(a)) and for the artificial graphs (Figure 11(b)). For both benchmarks sets, the number of variables and constraints grows roughly linear with the size of the graphs. This benign growth behavior is in sharp contrast to the exponential growth of the number of embeddings. The spike at 35 vertices in Figure 11(a) mirrors the smaller spike in the number of embeddings for the real world graphs.

8 Conclusion

Using methods of integer linear programming to minimize the number of bends in an orthogonal drawing seems to be a promising approach. The main drawback is that at the moment, the algorithm only works for biconnected graphs. The reason is that SPQR-trees are only defined for biconnected graphs. One possible approach



(a)



(b)

Figure 11: The average number of constraints and variables in the mixed integer linear programs for the real world benchmark set (a) and the artificial benchmark set (b)

to get overcome this limitation is to work with the block-cut-tree of biconnected components of the graph. If it can be used to describe the set of all embeddings of a connected graph as an ILP, our approach can be easily extended to deal with any planar graph.

Acknowledgment

We thank Walter Didimo for providing the code of the branch & bound algorithm and the real world benchmark set.

References

[AGD99] *AGD User Manual (Version 1.1)*, 1999. Universität Wien, Max-

Planck-Institut Saarbrücken, Universität Trier, Universität zu Köln. See also `http://www.mpi-sb.mpg.de/AGD/`.

[BDBD00] P. Bertolazzi, G. Di Battista, and W. Didimo. Computing orthogonal drawings with the minimum number of bends. *IEEE Transactions on Computers*, 49(8):826–840, 2000.

[BM88] D. Bienstock and C. L. Monma. On the complexity of covering vertices by faces in a planar graph. *SIAM Journal on Computing*, 17(1):53–76, 1988.

[BM89] D. Bienstock and C. L. Monma. Optimal enclosing regions in planar graphs. *Networks*, 19(1):79–94, 1989.

[BM90] D. Bienstock and C. L. Monma. On the complexity of embedding planar graphs to minimize certain distance measures. *Algorithmica*, 5(1):93–109, 1990.

[CDT95] G. Carpaneto, M. Dell'Amico, and P. Toth. Exact solution of large scale asymmetric travelling salesman problems. *ACM Transactions on Mathematical Software*, 21(4):394–409, 1995.

[DBETT99] G. Di Battista, P. Eades, R. Tamassia, and I.G. Tollis. *Graph Drawing.* Prentice Hall, 1999.

[DBGL+97] 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.

[DBLV98] G. Di Battista, G. Liotta, and F. Vargiu. Spirality and optimal orthogonal drawings. *SIAM Journal on Computing*, 27(6):1764–1811, December 1998.

[DBT96] G. Di Battista and R. Tamassia. On-line planarity testing. *SIAM Journal on Computing*, 25(5):956–997, 1996.

[DL98] W. Didimo and G. Liotta. Computing orthogonal drawings in a variable embedding setting. *LNCS*, 1533:79–88, 1998.

[FK96] U. Fößmeier and M. Kaufmann. Drawing high degree graphs with low bend numbers. In F. J. Brandenburg, editor, *Graph Drawing (Proc. GD '95)*, volume 1027 of *LNCS*, pages 254–266. Springer-Verlag, 1996.

[FM98] S. Fialko and P. Mutzel. A new approximation algorithm for the planar augmentation problem. In *Proceedings of the Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 260–269, San Francisco, California, 1998.

[GM01] C. Gutwenger and P. Mutzel. A linear time implementation of SPQR-trees. In J. Marks, editor, *Graph Drawing (Proc. 2000)*, volume 1984 of *LNCS*, pages 77–90. Springer-Verlag, 2001.

[GNS99] Z. Gu, G. L. Nemhauser, and M. P. Savelsbergh. Lifted flow cover inequalities for mixed 0-1 integer programs. *Math. Program.*, 85A(3):439–467, 1999.

[GT01] A. Garg and R. Tamassia. On the computational complexity of upward and rectilinear planarity testing. *SIAM Journal on Computing*, 31(2):601–625, 2001.

[HJ00] S. Hugh-Jones. Survey: Spain. *The Economist*, 2000.

[HT73] J. E. Hopcroft and R. E. Tarjan. Dividing a graph into triconnected components. *SIAM Journal on Computing*, 2(3):135–158, 1973.

[MW99] P. Mutzel and R. Weiskircher. Optimizing over all combinatorial embeddings of a planar graph. In G. Cornuéjols, R. Burkard, and G. Wöginger, editors, *Proceedings IPCO '99*, volume 1610 of *LNCS*, pages 361–376. Springer Verlag, 1999.

[MW00] P. Mutzel and R. Weiskircher. Computing optimal embeddings for planar graphs. In D. Z. Du, P. Eades, V. Estivill-Castro, X. Lin, and A. Sharma, editors, *Proceedings COCOON '00*, volume 1858 of *LNCS*, pages 95–104. Springer Verlag, 2000.

[PRW85] M. W. Padberg, T. .J. Van Roy, and L. A. Wolsey. Valid linear inequalities for fixed charge problems. *Operations Research*, (33):842–861, 1985.

[Tam87] R. Tamassia. On embedding a graph in the grid with the minimum number of bends. *SIAM Journal on Computing*, 16(3):421–444, 1987.