Bend Minimization in Orthogonal Drawings Using Integer Programming

Petra Mutzel and René Weiskircher

Vienna University of Technology
Favoritenstraße 9-11 E186, A-1040 Vienna, Austria
{mutzel|weiskircher}@ads.tuwien.ac.at

Abstract. We consider the problem of minimizing the number of bends in an orthogonal planar 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 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 computational experiments on a benchmark set containing hard problem instances that was already used for testing the performance of a previously published branch & bound algorithm for solving the same problem. Our new algorithm is about twice as fast as the branch & bound approach for the graphs of the benchmark set.

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*.

A well known approach for drawing general graphs is the topology-shape-metrics method. 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 seconds 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 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 [?] and minimizing certain distance measures on the faces of a graph with respect to the outer face [?,?]. 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 [?].

Bertolazzi et al. [?] 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 [?,?] 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 optimize the number of bends in an orthogonal drawing over the set of all embeddings of a planar graph. Solving this program using a commercial solver (CPLEX) is significantly faster for large and difficult graphs than the branch & bound approach of Bertolazzi et al . as our computational results show.

Section 2 introduces SPQR-trees and 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 3. 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 4, 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 5, 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 a set of hard benchmark graphs are given in Section 6. We compare the algorithm with a well known heuristic and with the branch & bound algorithm of Bertolazzi et al. The conclusion (Section 7) summarizes the main results and contains possible starting points for future work.

2 The ILP-Formulation Describing the Set of All Embeddings

The integer linear program (ILP) suggested in [?] 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. A graph is biconnected, if the number of its connected components can not be increased by deleting a vertex.

SPQR-trees have been defined by Di Battista and Tamassia [?]. 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 (Q-nodes, S-nodes, R-nodes and P-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.

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 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 that they can be constructed in linear time [?,?]. As described in [?], SPQR-trees can be used to represent the set of all combinatorial embeddings of a biconnected planar graph. 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 variables of the ILP correspond to directed cycles of the graph. Our recursive construction of the ILP guarantees that we only compute variables for cycles that form the boundary of a face in at least one embedding of the graph. So we generate the minimum set of variables needed to describe all embeddings. While the number of directed cycles in a graph grows exponentially with the size of the graph, our computational experiments in Section 6 show that the number of variables in our ILP grows only linearly.

We construct the program by splitting the SPQR-tree into smaller SPQR-trees, recursively 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. These graphs have a very simple structure and ILPs that describe their combinatorial embeddings are easy to construct. One type of constraints, similar to the subtour elimination constraints used in ILPs for the asymmetric travelings salesman problem (ATSP), are not explicitly added to the ILP because the number of these constraints is exponential. Instead we separate them in the optimization procedure using the same methods used for solving ATSP-problems with integer programming. We construct the ILPs of more complex graphs by merging the ILPs of the graphs

generated by the splitting procedure and adding additional glue constraints. Using structural induction, we can show that the resulting ILP is correct and that the variables correspond exactly to the set of cycles that are face cycles in at least one embedding of the graph.

3 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 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 [**?**]. 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 mentioned in [**?**]. According to this convention, the vertices are drawn as boxes of the same size and the edges are positioned on a finer grid then the vertices. Because of this modification, more than one edge can be incident to each of the four sides of a vertex (see Fig. 1 for an example).

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 [**?**]. The network 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 *c-nodes*).

Let f be the bijection that maps the vertices of G to the v-nodes of N and the face cycles of the planar embedding to the c-nodes. Then there is an arc between the v-node v_1 and the c-node v_2 if the vertex $f^{-1}(v_1)$ is on the cycle $f^{-1}(v_2)$. This arc is directed towards v_2 if the degree of $f^{-1}(v_1)$ is at most four and towards v_1 otherwise. There is an arc from the c-node v_3 to the c-node v_4 if the two cycles $f^{-1}(v_3)$ and $f^{-1}(v_4)$ share an edge.

The flow on arcs connecting v-nodes with c-nodes determines the angles between edges incident to the same vertex while the flow on arcs connecting two c-nodes determines the bends. Flow on an arc from a c-node to a v-node implies a zero-degree angle at the corresponding vertex between two incident edges and causes a bend on one of the edges. The amount of flow that each vertex in N produces or consumes together with the capacities for the edges guarantee that every feasible flow corresponds to an orthogonal representation. The cost per unit of flow on the arcs of the network are defined in such a way, that the cost of

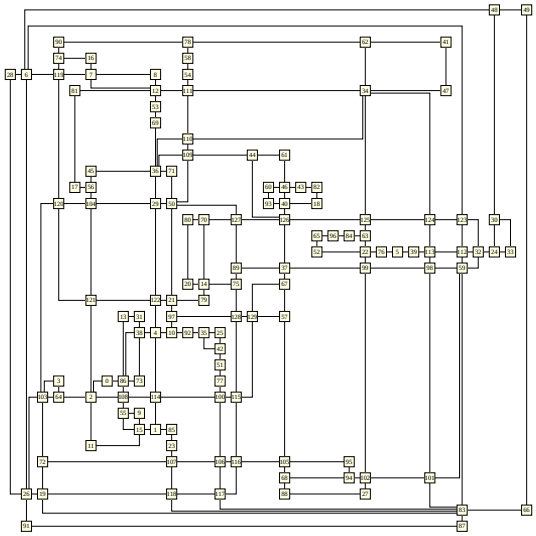

Fig. 1. A podevsnef drawing of a graph.

each feasible flow is equal to the number of bends in the represented orthogonal representation.

We used this network and transformed it into a linear program. There is one variable for each arc in the network that represents the amount of flow routed via this arc. One constraint for each vertex in the network makes sure that the number of incoming amount of flow minus the number of outgoing amount is equal to the demand of the node (some nodes have negative demand). We have one constraint for each arc that sets upper and lower bounds for the flow on the arc. 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. Because of space constraints, we do not present the LP here, but the constraints are all contained in the mixed integer linear program of Section 4, that is used to compute an orthogonal representation with the minimum number of bends over all embeddings.

4 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 with a fixed embedding contains one c-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 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. The set of these cycles corresponds to the set of variables in our ILP from Section 2 that describes the set of all embeddings of a graph.

In a solution of the embedding ILP, the variables of the cycles that are face cycles in the represented embedding have value one while all other variables have value zero. Let A be the set of edges 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 flow zero. Therefore, the flow on the arcs of the network N' incident to c-nodes corresponding to cycles in G whose variable in the ILP is zero must also be zero.

To achieve this, we take the variables of the ILP into account when we compute the capacities of the edges and the amount of flow that each c-node consumes or produces. We first compute the capacities of the arcs and the demand of each c-node analogously to the corresponding values 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, because the corresponding variable in the ILP is zero.

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 edge is limited even if the corresponding cycle is a face cycle, we can just multiply this limit with the ILP-value of the cycle. The arcs in the network N that connect two c-nodes have unlimited capacity. But we can easily compute an upper bound f_{max} for the flow produced in N' (we get a trivial upper bound by adding the supply of all nodes). This value can be used as the upper bound for 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 the two products $f_{max}x_i$ where x_i is the 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 chosen embedding.

The result is the network N', where the capacities of the edges and the amount of flow produced and consumed by the vertices depend on the values of the cycle variables in the ILP. We transform this network into a linear program and merge it with the ILP that represents the embeddings of the graph. 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. The set C is the set of cycles in G that are face cycles in at least one embedding. The variable x_c is one if cycle c is a face cycle and variable o_c is one if it 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} have the opposite direction. The expression $len(c)$ denotes the number of edges in cycle c.

MILP 1

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

subject to

$$\begin{aligned}
\sum_{c \in C} o_c &= 1 \\
x_c - o_c &\geq 0 && \forall c \in C \\
\sum_{e=(v,w) \in E_N} f_e - \sum_{e=(w,v) \in E_N} f_e &= 4 - deg(v) && \forall v \in V \\
\sum_{e=(c,w) \in E_N} f_e - \sum_{e=(w,c) \in E_N} f_e &= x_c(4 - len(c)) - 8o_c && \forall c \in C \\
f_e &\leq x_c(4 - deg(v)) && \forall e = (v,c) \in E_{vc} \\
f_e &\leq x_c && \forall e = (c,v) \in E_{cv} \\
f_e &\leq x_{c_1} f_{max} && \forall e = (c_1,c_2) \in E_{cc} \\
f_e &\leq x_{c_2} f_{max} && \forall e = (c_1,c_2) \in E_{cc} \\
f_e &\geq 0 && \forall e \in E_N \\
x_c, o_c &\in \{0,1\} && \forall c \in C
\end{aligned}$$

5 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 separate subtour elimination constraints from the embedding ILP and re-optimize if necessary. 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 orthogonal representations we exclude in this way are mirror images of other orthogonal representations that can still be represented. 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 constraints.

6 Computational Results

Since we wanted to compare the performance of our approach with the branch & bound method for bend minimization by Bertolazzi [**?**], we used the same set of graphs

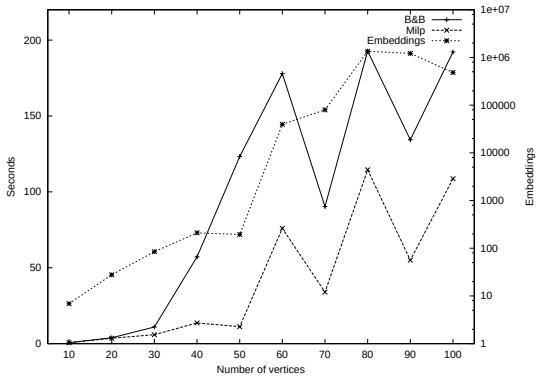

Fig. 2. Run time comparison with the branch & bound algorithm (linear scale) together with the average number of embeddings (logscale)

that they used for testing the performance of their algorithm. This set consists of 500 randomly generated graphs, 50 different graphs for each number of vertices from 10 to 100 in steps of 10.

Our algorithm and the branch & bound algorithm have the same limitations: They can only be applied to planar biconnected graphs, because they both use SPQR-trees. All the graphs in the benchmark set have these properties.

First, we compared the optimal results produced by our algorithm with 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 of section 3.

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. For each graph in the benchmark set, we computed the following value: $\frac{h-o}{h}100\%$. This is the percentage of the improvement we get using an optimal algorithm. Almost half of all graphs (246 out of 500) show a significant improvement (greater than 10%). The greatest absolute difference in the number of bends that we observed from the heuristic solution to the optimal solution was 12 bends. The average number of saved bends per graph using the optimal algorithm was 2.26. The average improvement over all graphs was 17.43%.

We compared the average running time for graphs with the same number of vertices of our new algorithm (MILP) and the branch & bound algorithm (B&B) from [**?**]. Both algorithms were tested on a Sun Enterprise 450 Model 4400 with 4GB main memory. The running time of MILP includes the time needed for the recursive construction of the ILP that describes the embeddings of a graph.

Figure 2 shows the average running time of both algorithms. The x-axis shows the number of vertices in the graphs and the y-axis on the left the average running time in seconds for all graphs in the benchmark set with that number

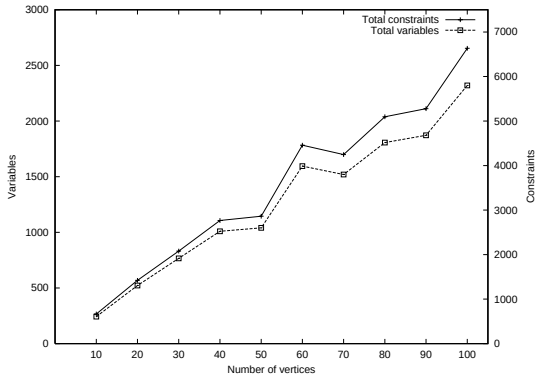

Fig. 3. The average number of constraints and variables grows only linearly with the size of the graphs

of vertices. The plot shows that our new algorithm needs on average only half the time needed by the branch & bound algorithm to compute the drawing with the minimum number of bends. The same plot contains the curve showing the average number of embeddings for graphs with the same number of vertices. As expected, the average number of embeddings grows exponentially with the size of the graphs (note that the y-axis on the right is logarithmic). However, the average number of constraints and variables in our mixed integer linear program grows only linearly with the size of the graphs (see Figure 3).

We also applied the branch & bound algorithm and our new algorithm to a set of 11529 graphs derived from graphs used in industrial applications. We created these graphs by planarizing the graphs in the benchmark set used in [**?**] and then adding edges to make them planar and biconnected. Because of space considerations, we can only mention a few statistics. The branch & bound algorithm failed to provide an optimal solution in one hour of computation time for 197 of the graphs, while our new algorithm exceeded this time limit for only 25 graphs. While the branch & bound algorithm is slightly faster on average for the graphs in the set with less than about 120 vertices, our algorithm has a significant speed advantage for the graphs with more than 150 vertices.

7 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. A possible approach to get rid of this limitation is to work with the block 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 benchmark graphs.