A New Approximation Algorithm for Bend Minimization in the Kandinsky Model*

Wilhelm Barth[1], Petra Mutzel[2], and Canan Yıldız[1]

[1] Institute of Computer Graphics and Algorithms, Vienna University of Technology
Favoritenstraße 9-11, 1040 Wien, Austria
{barth,yildizca}@ads.tuwien.ac.at
http://www.ads.tuwien.ac.at/
[2] Department of Computer Science, University of Dortmund
Otto-Hahn-Str. 14, D-44227 Dortmund, Germany
petra.mutzel@cs.uni-dortmund.de
http://ls11-www.cs.uni-dortmund.de/

Abstract. The Kandinsky model has been introduced by Fößmeier and Kaufmann in order to deal with planar orthogonal drawings of planar graphs with maximal vertex degree higher than four [7]. No polynomial-time algorithm is known for computing a (region preserving) bend minimal Kandinsky drawing. In this paper we suggest a new 2-approximation algorithm for this problem. Our extensive computational experiments [13] show that the quality of the computed solutions is better than those of its predecessors [6]. E.g., for all instances in the Rome graph benchmark library [4] it computed the optimal solution, and for randomly generated triangulated graphs with up to 800 vertices, the absolute error was less than 2 on average.

1 Introduction

Given a planar graph $G = (V, E, F)$ with a fixed embedding F, we consider the problem of finding a region-preserving planar orthogonal drawing with the minimum number of bends. For graphs with maximal degree 4 this problem can be solved in polynomial time [12] if no parallel edge segments leaving a vertex at the same side are allowed. The idea is to set up a network in which each flow corresponds to an orthogonal drawing and vice versa. A bend-minimal orthogonal drawing can thus be obtained from a flow of minimum cost in this network [12], which can be solved in polynomial time [2,10].

Several extensions have been suggested in order to deal with graphs of higher vertex degree. The Kandinsky model has been suggested by Fössmeier and Kaufmann [7]. In this model the vertices are represented by squares of equal size placed on a coarse vertex grid on the plane. The edges consist of continuous sequences of horizontal and vertical line segments routed on a finer edge grid. Thus more than one edge can leave a vertex from the same side, forming a

* Full paper, submitted to GD 2006.

M. Kaufmann and D. Wagner (Eds.): GD 2006, LNCS 4372, pp. 343–354, 2007.

so called 0°-*angle*. Moreover, faces are not allowed to be represented by empty regions (empty faces)[1]. This model has the great advantage that vertices may have prescribed sizes and do not grow arbitrarily as it is the case, e.g., in the GIOTTO model [3], or with respect to the number of edges leaving on one side as in [9].

So far no polynomial time algorithm for computing a bend-minimal drawing in the Kandinsky model, which we refer to as the KMCF-problem, is known[2]. Fössmeier and Kaufmann have suggested the *Kandinsky network*, which can be seen as an extension of the Tamassia network. Unfortunately, a minimum-cost flow in the Kandinsky network must satisfy additional constraints in order to correspond to a bend-minimal Kandinsky drawing of the underlying graph. While there does not exist a polynomial time algorithm to find such a flow, the network can be used for setting up an integer linear program (ILP) for the KMCF-problem.

Bertolazzi et al. [5] have suggested a restriction of the Kandinsky model, the *simple-podevsnef* model, which can be solved in polynomial time. For the KMCF-problem, Eiglsperger [6] has presented a polynomial time algorithm that guarantees to compute a solution with at most twice as many bends as the optimal solution. The algorithm is based on the idea to first compute a minimum-cost flow in the Kandinsky network ignoring the additional constraints, and then to 'repair' the obtained infeasible solution.

In this paper we present the *Cyclic Shift Algorithm* (CS), an alternative 2-approximation algorithm for the KMCF-problem. The idea of our new algorithm is to first solve the LP-relaxation of the corresponding integer linear program (i.e., to ignore the integer constraints), and then to 'repair' the obtained infeasible solution with augmenting cycles. We also consider a variation of the CS algorithm, the *Successive Cyclic Shift Algorithm* (CSS), which performs the cycle augmentations in successive iterations. Extensive computational experiments (see also [13]) have shown that the quality of the computed solutions is better than those of the approximation algorithm in [6]. Moreover, our CS algorithm finds optimal solutions for all instances of the Rome graph benchmark library.

The remainder of this work is organized as follows: In section 2 we briefly refer to the Kandinsky network, which is slightly modified compared to the network described in [7]. In section 3 we describe both variants of the new Cyclic Shift Algorithm (CS and CSS). In section 4 we show that our algorithms guarantee to find a 2-approximate solution to the KMCF-problem. Finally, in section 5 we present experimental results with our new algorithms, two versions of the algorithm suggested in [6], and an exact ILP approach. The tests were performed on the Rome graph benchmark library and on a large set of randomly generated graphs (belonging to different graph classes).

[1] In [7] the Kandinsky model is referred to as *podevsnef*, which originates from "Planar Orthogonal Drawings with Equal Vertex Size and Non-Empty Faces".

[2] The approach suggested in [7] turned out to be not correct, see also Eiglsperger [6].

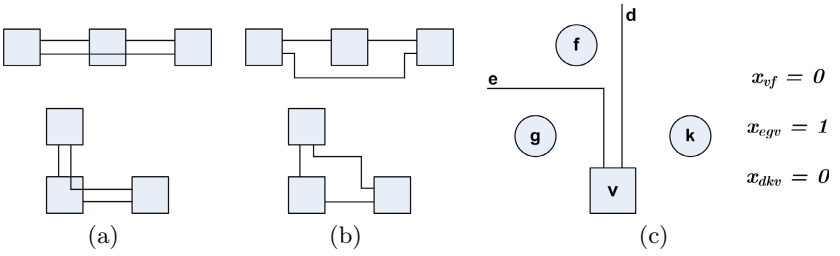

Fig. 1. (a) Invalid and (b) valid drawings (c) vertex-bend forced by a 0°-angle

2 The Kandinsky Model

Allowing 0°-angles at the vertices leads to several complications which have to be taken into account: Firstly an edge segment may traverse a vertex, secondly an empty face may be generated (Fig. 1a). For preventing this, a bend for each 0°-angle will be forced; such a bend is called *vertex-bend* (see Fig. 1(c)). All other (ordinary) bends are called *edge-bends*. It can be shown [7] that one distinct vertex-bend for each 0°-angle suffices to prevent invalid constructions shown in Fig. 1(a) and to enforce valid Kandinsky drawings (see Fig. 1(b)).

2.1 Variables and Constraints for Orthogonal Drawing

We introduce variables $x_{vf} \in \{0, 1, 2, 3, 4\}$ associated with the angle at vertex v between two adjacent edges enclosing the face f. One variable unit corresponds to a 90° angle. The sum of all angles at v has to be 360°, i.e.

$$\sum_{f \in F(v)} x_{vf} = 4 \quad \forall v \in V \tag{1}$$

where $F(v)$ denotes all faces adjacent to v.

The variables x_{ef} and x_{eg} provide the number of edge-bends on the edge $e \in E$ dividing the two adjacent faces f and g, where x_{ef} represents the convex edge-bends in face f (which at the same time are concave edge-bends in face g) and x_{eg} the convex edge-bends in face g.

Consider a vertex-bend on an edge e incident to vertex v and separating the faces f and g. With the 0/1-variable x_{egv} we associate a vertex-bend forced by a 0°-angle in face f and thus forming a convex bend in g (Fig. 1c). As stated before we have to assure that one of the two edges d and e forming a 0°-angle has a vertex-bend, i.e.

$$x_{vf} + x_{egv} + x_{dkv} \geq 1 \quad \forall v \in V, f \in F(v) \ . \tag{2}$$

Obviously, an edge e is not allowed to have a vertex-bend from the left and the right side in opposite directions. This is forced by the constraint:

$$x_{efv} + x_{egv} \leq 1 \quad \forall v \in V, e \in E(v) \tag{3}$$

where $E(v)$ denotes the set of edges incident with v.

To make sure that each face has the correct shape of a rectilinear polygon, it has to be guaranteed that the difference between the number of convex and concave angles is equal to 4 within each inner face and equal to -4 within the outer face. Here angles at vertices (*vertex-angles*) and angles formed by bends (*bend-angles*) must be included in the calculation. This leads to constraints

$$\sum_{e=(v,w)\in E(f)} (x_{ef} - x_{eg} + x_{efv} - x_{egv} + x_{efw} - x_{egw}) - \sum_{v\in V(f)} (x_{vf} - 2) = \mp 4 \tag{4}$$

for all faces $f \in F$, where g is the face on the other side of the edge (v, w) and $E(f)$ (resp. $V(f)$) denotes the set of edges (resp. vertices) on the boundary of f. It can be shown that each valid set of variables satisfying (1)-(4) defines a valid Kandinsky shape of the underlying graph [7].

2.2 The Corresponding Kandinsky Network

With each graph $G = (V, E, F)$ we associate a network $\mathcal{N} = (N, A)$ with additional constraints, such that the cost of a flow x in $\mathcal{N}$ is equal to the number of bends in the corresponding Kandinsky shape of G. The network is a directed graph with the **node set** $N = N_V \cup N_F \cup N_H$, where

- N_V: contains a node for each vertex $v \in V$,
- N_F: contains a node for each face $f \in F$,
- N_H: contains a wreath of artificial nodes around each vertex $v \in V$,

and the **edge set** $A = A_{VH} \cup A_{HF} \cup A_{FH} \cup A_{FF}$, where an edge in one of these four subsets has the following capacity constraints on the flow and cost:

- A_{VH}: $[0:4]$ 0 represents a vertex-angle,
- A_{HF}: $[1:4]$ 0 lower bound forces a vertex-bend,
- A_{FH}: $[0:1]$ 1 represents a vertex-bend,
- A_{FF}: $[0:\infty]$ 1 represents an edge-bend.

The flow variables in the network correspond to the variables introduced in section 2.1. Before we provide the details for the flow variables in the Kandinsky network, we consider the classical Tamassia network. There, the nodes consist of $N_V \cup N_F$ only, and the edges of $A_{VF} \cup A_{FF}$, where each edge in A_{VF} represents a vertex-angle and has lower bound 1. Fig. 2(a) shows a part of the Tamassia network, which corresponds to the neighbourhood of a vertex with three adjacent edges (dotted lines are edges of the corresponding graph and not of the network). The variables in the figure correspond to flows on the network-edges. The flow has to satisfy the constraints (1) and (4) of section 2.1, hence each vertex-node is a source with a supply of 4 and each face-node a target with a demand of $4 - 2|f|$ (resp. $4 + 2|f|$ if f is the outer face). In [12] it is shown that each minimum-cost flow in this network corresponds to an orthogonal drawing with the minimum number of bends and vice versa.

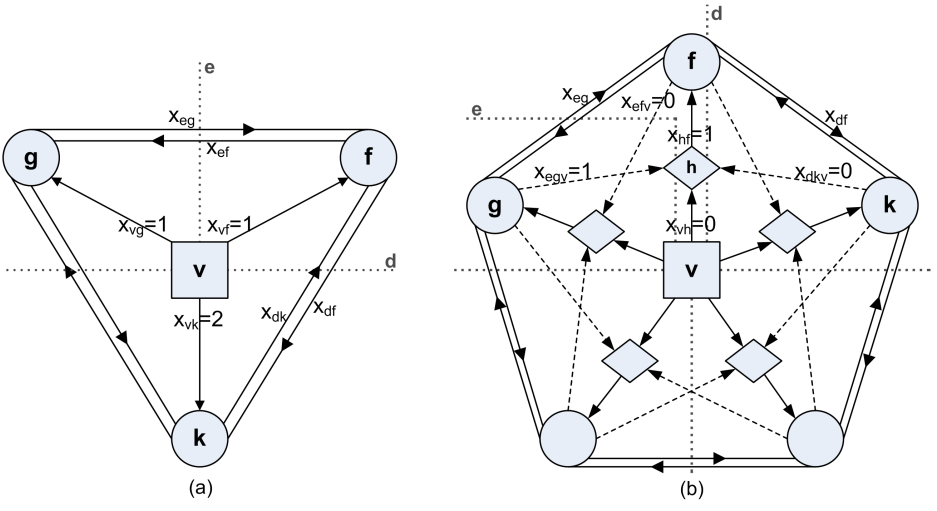

Fig. 2. Part of the Tamassia network (a) and the Kandinsky network (b)

In the Kandinsky model 0°-angles (i.e., 0-flow on vf-edges) are allowed and we have to guarantee that there is a distinct vertex-bend for each one of them. In the network this can be achieved by introducing an artificial structure as shown in Fig. 2(b). Here each vf-edge in the Tamassia network is split into two parts by introducing an artificial node h, where the new vh-edge represents the vertex-angle (formerly represented by the vf-edge). Note that Equation (1) changes in this case to $\sum_{h \in H(v)} x_{vh} = 4$, where $H(v)$ is the wreath of artificial nodes around v. A similar change takes place in Equation (4).

Furthermore, new fh-edges are introduced, representing the vertex-bends. The new hf-edge has a lower bound of 1. So in case of a 0°-angle ($x_{vh} = 0$) one of the two fh-edges ending in node h has to carry a flow of 1 unit in order to satisfy the capacity constraint on the hf-edge. In this way a wreath of artificial nodes and fh-edges is formed around each vertex v such that inequality (2) is always satisfied by a valid flow.

We call a pair of fh-edges, each one corresponding to a vertex-bend on the same edge but in opposite directions (pair of intersecting dashed lines in Fig. 2b), **bundles**. At most one of the two edges of a bundle is allowed to carry positive flow. This cannot be achieved by means of network flow techniques. Therefore the flow has to satisfy the additional Constraint (3).

It can be shown that each valid minimum-cost flow satisfying (1), (3), (4) and the capacity constraints in the Kandinsky network corresponds to a bend minimal Kandinsky drawing of the underlying graph G [7,6,13].

2.3 The Integer Linear Program

The following ILP is equivalent to the KMCF-problem.

$$\min z = \sum_{e=(v,w)\in E} (x_{ef} + x_{eg} + x_{efv} + x_{egv} + x_{efw} + x_{egw}) \quad \text{(a)}$$

subject to

$$\sum_{h\in H(v)} x_{vh} = 4 \qquad \forall v \in V, h \in H(v) \quad \text{(b)}$$

$$\sum_{e=(v,w)\in E(f)} (x_{ef} - x_{eg} + x_{efv} - x_{egv} + x_{efw} - x_{egw}) - \sum_{v\in V(f)} (x_{vh} - 2) = \mp 4 \qquad \forall f \in F \quad \text{(c)}$$

$$x_{efv} + x_{egv} \leq 1 \qquad \forall e = (u, v) \in E \quad \text{(d)}$$

$$0 \leq x_{vh} \leq 4,\ 1 \leq x_{hf} \leq 4 \qquad \forall v \in V h \in H(v)$$
$$0 \leq x_{ef} \leq \infty,\ 0 \leq x_{efv} \leq 1 \qquad \forall e \in E, f \in F(e), v \in V(e) \quad \text{(e)}$$

All variables integer (f)

Fig. 3. The integer linear program (ILP) of the KMCF-problem

3 The Cyclic Shift (CS) Algorithm

The ILP-formulations of the KMCF-problem and the classical minimum-cost flow (MFC) problem differ only by the *bundle capacity constraints* (Fig. 3.(d)). In section 3.1 we first describe the basic variant of our Cyclic Shift Algorithm (CS), and section 3.2 then describes the *successive* variant (CSS).

3.1 The Basic Cyclic Shift (CS) Algorithm

CS1: We drop the integer constraint and solve the resulting LP-relaxation. In general the optimum value z_{CS1} of this relaxed problem is smaller than the optimum z_{opt} of the ILP and its solution may contain non-integral values. In particular there may exist bundles, both of whose edges have fractional flows. We call such bundles *critical*, whereas bundles with at least one edge having a zero flow value are called *non-critical*. If the solution is integral, the optimal solution of the ILP is found ($z_{CS1} = z_{opt}$) and the algorithm terminates.
CS2: All critical bundles in the solution obtained in step CS1 will be transformed into non-critical ones by augmenting the flow along *correcting cycles*. Note that the constraints (b),(c) and (d) are not violated by these augmentations and additional costs arise only while traversing edges in A_{FH} and A_{FF}. We use three different types of cycle-corrections. For details we refer to [13]. In the following we denote with $x(e)$ the flow in an edge e of the Kandinsky network.

Type 1 (Fig. 4): If the left hand neighbour of a critical bundle $A = (a^L, a^R)$ is a non-critical one $B = (b^L, b^R)$ with $x(b^L) = 0$, then we augment the flow along the correcting cycle $(-a^L, fk, a^R, -vh, vh')$ by $x(a^L)$, so that A becomes non-critical. Note that the capacities of fk and a^R will not be exceeded by this transformation, since fk has infinite capacity and we have $x(a^L) + x(a^R) \leq 1$. Furthermore, the lower bound 1 of edge hf ensures that the flow over vh before

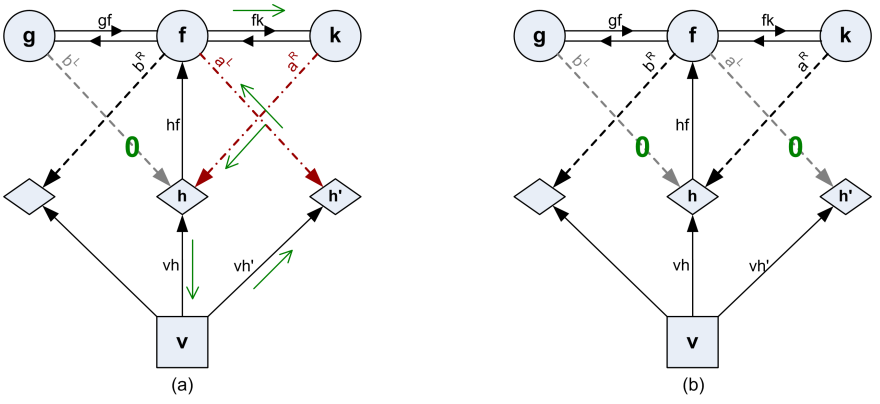

Fig. 4. Cycle-correction of Type 1

our transformation is at least $x(a^L)$, since we have $x(vh) \geq 1 - x(a^R) \geq x(a^L)$. The additional cost of this correction is $x(a^L)$.

Now we can apply the same cycle-correction to the right-hand neighbour of bundle A in case that it is critical, too. We continue with these procedure until we reach a non-critical bundle. Thus we transform a chain I of critical bundles between two non-critical ones, into non-critical bundles. The total additional cost is $\sum_{A \in I} x(a^L)$.

We can also perform these transformations in the same way but in the opposite direction beginning with the rightmost bundle of the chain, if its right-hand neighbour is a bundle B with $x(b^R) = 0$. The total cost would be $\sum_{A \in I} x(a^R)$.

Type 2 (Fig. 5): If $x(b^R) = 0$, a similar left to right correction can be applied. We first augment the flow along the cycle $(-a^R, kf, -hf)$ (Fig. 5a) by $\min(x(a^R), x(hf) - 1)$. Note that this can be done with 0 cost. After this either a^R has zero flow (then A is already non-critical) or hf has flow 1. In the latter case b^L and a^R have a total flow of no more than 1 and we can augment the flow along the correcting cycle $(-a^R, kf, fg, b^L)$ (Fig. 5b) to obtain 0 flow on a^R without violating the capacity constraint of b^L.

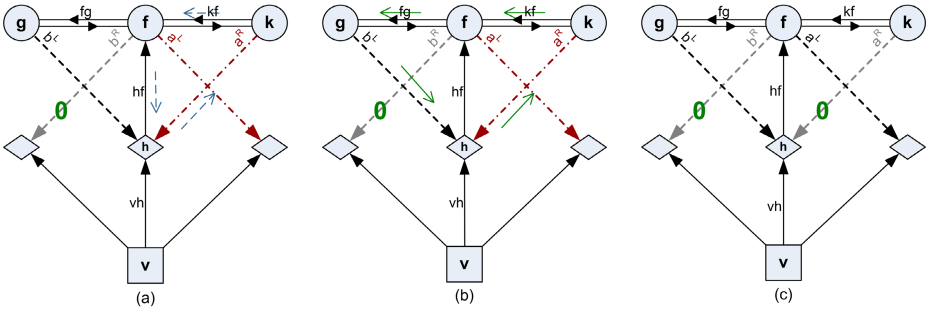

Fig. 5. Cycle-correction of Type 2

The bundle A becomes non-critical and we can continue applying the same cycle-corrections on the remaining bundles of the chain I. The additional cost is bounded by $2x(a^R)$ for a single bundle and thus by $\sum_{A\in I} 2x(a^R)$ for the entire chain. If $x(b^L) = 0$ holds for the right neighbour of the chain the correction may be performed beginning with the rightmost bundle. In this case, the additional cost is bounded by $\sum_{A\in I} 2x(a^L)$.

It follows that all chains of contiguous critical bundles enclosed by two non-critical bundles can be transformed into non-critical ones by applying cycle-corrections either from right to left or from left to right. In the CS algorithm we always choose the direction with cheaper cost.

Type 3: If all bundles in the wreath of bundles around a vertex v are critical, we begin with two adjacent bundles A and B (Fig. 6a) and apply the same cycle-corrections as described in Type 2, so that A becomes non-critical. B not only remains critical, but may even exceed its bundle capacity (Fig. 3.(d)). But the edge capacities are not violated. If we continue applying counter clockwise the same cycle-correction on B and C, B becomes non-critical, C eventually exceeds its bundle capacity and so forth. After the cycle-correction is applied on the last critical bundle and A (Fig. 6b), all bundles have become non-critical and A does not exceed its capacity. The total additional cost is bounded by $\sum_{A\in I} 2x(a^R)$. If we process the bundles in the opposite (clockwise) direction the cost is bounded by $\sum_{A\in I} 2x(a^L)$. We choose the cheaper direction.

We will denote the set of all critical bundles of the network by I_c and the cost for correcting all of them by $\text{cost}(I_c)$. Then we have $z_{CS2} := z_{CS1} + \text{cost}(I_c)$.

CS3: From each bundle we choose an edge with a zero flow value and set its upper bound to zero, i.e. we *lock* the bundle. After that at most one edge of each bundle can carry flow, thus the bundle capacity constraint is redundant and the KMCF-problem is reduced to a MCF problem.

CS4: We solve the MCF problem obtained in step CS3. Since capacities and cost are integer, this problem has always an integer optimal solution that can

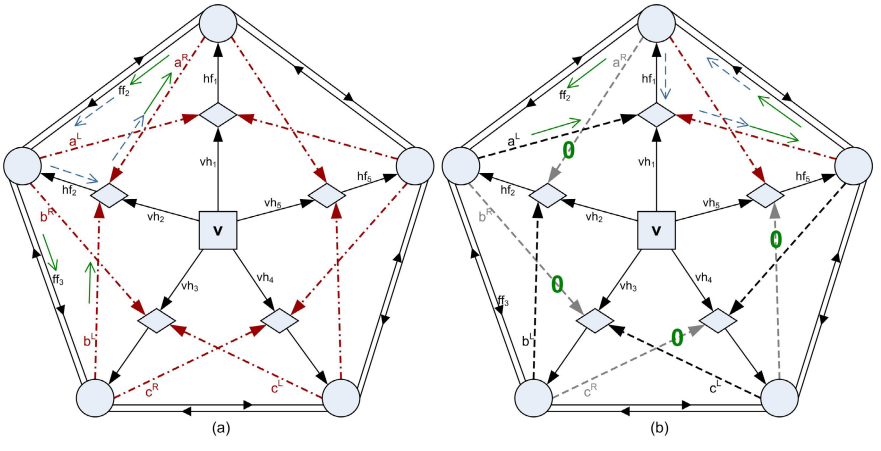

Fig. 6. Cycle-correction of Type 3

be found in polynomial time with a standard MCF algorithm. For the obtained objective value z_{CS} the following holds: $z_{opt} \leq z_{CS} \leq z_{CS2}$. If $z_{CS} < z_{CS1} + 1$, the solution found by CS is optimal.

3.2 The Successive Cyclic Shift (CSS) Algorithm

Altough the basic variant yields very good practical results, we also experimented with the following version. Here, we lock only the bundles around one certain vertex v in each iteration instead of locking all bundles at once, and solve the LP again. Thus the remaining bundles can adapt their values to this restriction. Bundles once locked, remain locked in following iterations. Each iteration of this algorithm consists of the following steps:

CSS1: We solve the LP-relaxation as in CS1. Note that all bundles locked in previous iterations still stay locked.
CSS2: Same as CS2. Additionally, we determine for each vertex $v \in V$ the cost $c(v)$ caused by correcting the critical bundles around this vertex.
CSS3: We compute a feasible solution by solving the MCF-problem obtained by locking all bundles. If this solution z_{CSS} is better than the ones of the previous iterations, we take it. If $z_{CSS} < z_{CSS_{(1)}} + 1$, where $z_{CSS_{(1)}}$ is the solution of the LP-relaxation in the first iteration, the optimal solution is found and CSS terminates.
CSS4: We only lock the bundles around the vertex v with the highest cost $c(v)$ and go to CSS1.

We can go on with the iterations until the solution in step CSS1 is integer. This will happen at the latest when all bundles are locked. But consider that in the third step of each iteration a feasible solution is computed, so we can stop iterating at some point and take the best solution found so far as the final one. In our experiments, we restricted the number of iterations to 5.

4 Worst Case Analysis for Quality and Runtime

The objective value of the LP-relaxation in step CS1 is in general smaller than the objective value z_{opt} of the ILP: $z_{CS1} \leq z_{opt}$.

In step CS2 the additional costs $\text{cost}(I)$ which arise during the correction of a chain I of critical bundles is, if the correction is performed from left to right

$$\sum_{A \in I} x(a^L) \text{ in case of Type 1 and } \sum_{A \in I} 2x(a^R) \text{ in case of Type 2;}$$

if performed from right to left

$$\sum_{A \in I} x(a^R) \text{ in case of Type 1 and } \sum_{A \in I} 2x(a^L) \text{ in case of Type 2.}$$

Since we choose the cheaper direction, provably the following holds:

$$\text{cost}(I) \leq \sum_{A \in I} (x(a^L) + x(a^R)) \tag{5}$$

Thus for the cost caused by the correction of all the chains we obtain:

$$z_{CS2} \leq z_{CS1} + \sum_{I \in I_C} \sum_{A \in I} (x(a^L) + x(a^R)) \leq 2z_{CS1} \leq 2z_{opt} \tag{6}$$

In the following two steps we lock bundle-edges which are not used anyway and reoptimize, thus the new objective value z_{CS} can not be greater than z_{CS2}:

$$z_{CS} \leq z_{CS2} \leq 2z_{opt} \tag{7}$$

The basic variant CS has therefore the approximation factor 2. Since the same solution is obtained in the first iteration of CSS and will be replaced in later iterations only by better ones, CSS does not give worse results.

The running time of step CS2 is linear in $|E|$ as there are two bundles for each edge $e \in E$, and each critical bundle is processed only once with constant expense. Thus the total running time is $O(LP) + O(|E|) + O(MCF)$, hence polynomial. With its constant number of iterations (five in our case) CSS has also a polynomial running time.

5 Experimental Results

We have compared the solutions of our algorithms CS and CSS with two versions TE and TES of the algorithm by Eiglsperger [6], and the optimal solutions obtained by the ILP. A short description of TE and TES is as follows[3]:

TE1:	Solve the KMCF-problem without the bundle capacity constraint
TE2:	Correct overfilled bundles
TE3:	Lock each bundle and solve the resulting MCF-problem.
TES1,TES2:	same as TE1 and TE2
TES3:	Lock each bundle and solve the resulting MCF-problem; keep the solution, if it is better than the previous.
TES4	Lock only those bundles, which have flow value 1 before TES2 and zero flow afterwards. Go to TES1.

Besides the Rome graphs [4] (11,529 graphs with up to 100 nodes) we used a randomly generated data set consisting of about 14,000 planar graphs of several classes with up to 800 nodes. The non-planar Rome graphs were planarised with the *Subgraph Planarizer* of the AGD-Library [1]. The test runs were performed on an Intel Pentium IV 2.8GHz with 2GB RAM using ILOG CPLEX 8.1 [11]. We used the *Mixed Integer Optimizer* for the ILP, the *Network Optimizer* for the minimum-cost flow problems (CS4,CSS3,TE), and the standard settings for the LP (CS1/CSS1).

All instances of the Rome graphs have been solved to optimality by our basic CS algorithm. The optimality has been proven for 99.9% of them by the algorithm itself, i.e., $z_{CS} < z_{CS1} + 1$; only for 10 graphs we needed to calculate the

[3] For a detailed description we refer to [6]. We have slightly modified TE and TES to be conform with the implementations of CS and CSS, improving their results by doing so. For details, see [13].

Table 1. Experimental results for (random) maximal planar graphs

row	$\|V\|$	**100**	**200**	**300**	**400**	**500**	**600**	**700**	**800**
1	$z_{TE} - z_{opt}$	17.83	36.97	56.81	93.27	112.61	138.37	160.26	182.75
2	$z_{TES} - z_{opt}$	7.54	16.08	24.35	30.68	39.40	45.61	52.82	62.61
3	$z_{CS} - z_{opt}$	0.30	1.24	1.36	2.02	3.19	3.67	4.84	5.41
4	$z_{CSS} - z_{opt}$	0.03	0.23	0.36	0.42	0.83	0.86	1.62	1.62
5	z_{opt}	234.52	474.61	715.66	956.08	1196.06	1437.62	1678.63	1918.34
6	$z_{CS1} - z_{opt}$	−0.01	−0.01	−0.03	0	−0.03	−0.01	−0.04	−0.06
7	$z_{TE1} - z_{opt}$	−0.02	−0.03	−0.07	−0.03	−0.07	−0.2	−0.2	−0.23
8	$\#(z_{opt} = z_{CS})$	80	44	35	19	13	6	4	3
9	$\#(z_{opt} = z_{CSS})$	97	82	76	68	47	49	30	26
10	t_{TE}	0.05	0.11	0.19	0.30	0.42	0.57	0.73	0.93
11	t_{CS}	0.09	0.32	0.55	0.89	1.36	1.89	2.57	3.30
12	t_{CSS}	0.1	0.41	0.77	1.32	2.19	3.23	4.61	5.91
13	t_{TES}	0.73	2.21	4.37	7.52	11.68	16.45	22.07	28.70
14	t_{ILP}	0.24	1.57	3.95	8.02	13.38	23.24	41.35	49.91

optimal ILP-value for confirmation. In [6], Eiglsperger has described his experimental results with the successive variant of his algorithm which was able to solve most of the Rome graphs to optimality. However, on 392 instances it has produced 1 additional bend, and on 13 instances 2 additional bends.

Table 1 shows our experimental results for the maximal planar graphs that turned out to be the most difficult instances in our randomly generated test set (see [13]). Rows 1-5 show the average absolute errors of the four algorithms and the average number of bends in the optimal solution, each averaged over 100 instances of the same size. Expectedly the successive variants perform much better. Note that the average relative error of our CSS algorithm does not exceed 0.10%, while it is between 3.15% and 3.40% for TES. Moreover, the average absolute error of CSS is less than 2 for all tested instances. Thus the question arises if it is worth the effort of solving the ILP for getting the exact optimum. Though the ILP has a tolerable running time for small instances, it becomes unacceptable for larger ones because of its exponential increase (row 14).

The objective values obtained by the infeasible solutions of the first steps CS1 and TE1, resp., provide the starting values for the cycle correction steps of the approximation algorithms. Therefore, we were interested in the underestimation for the number of bends obtained by CS1 and TE1 (see rows 6 and 7). It can be observed that the objective value z_{CS1} is very close to the optimal value z_{opt}, and always closer than z_{TE1}. The latter observation is always true, since for computing z_{CS1} we only omit the integer constraints whereas for computing z_{TE1}, the bundle constraints are omitted and with them automatically the

integer constraints. Rows 8 and 9 show how often our algorithms CS and CSS have found the optimal solutions (out of 100 instances per size), and rows 10-14 give the running times of the four algorithms and the ILP approach.

References

1. *AGD User Manual*, 1999. `http://www.ads.tuwien.ac.at/AGD/`
2. Ahuja, R.K., Magnanti, T.L., Orlin, J.B.: Network Flows. *Prentice-Hall*, 1993.
3. Batini, C., Nardelli, E., Tamassia, R.: *A Layout Algorithm for Data Flow Diagrams.* IEEE Trans. Softw. Eng., SE-12(4), pages 538-546, 1986.
4. Di Battista, G., Garg, A., Liotta, G., Tamassia, R., Tassinari, E., Vargiu, F.: *An Experimental Comparison of Three Graph Drawing Algorithms.* Proc. 11th Ann. ACM Symp. Comput. Geom., pages 306-315, 1995.
5. Bertolazzi, P., Di Battista, G., Didimo, W.: *Computing orthogonal drawings with the minimum number of bends.* IEEE Trans. Comput., 49(8), pages 826-840, 2000.
6. Eiglsperger, M.: *Automatic Layout of UML Calss Diagrams: A Topology-Shape-Metrics Approach.* PhD thesis, Eberhard-Karls-Universität zu Tübingen, 2003.
7. Fößmeier, U., Kaufmann, M.: *Drawing high degree graphs with low bend numbers.* In F.J. Brandenburg, editor, Proc. 3rd Int. Symp. on Graph Drawing (GD'95), volume 1027 of LNCS, pages 254-266. Springer, 1996.
8. Fößmeier, U.: *Orthogonale Visualisierungstechnicken für Graphen.* PhD thesis, Eberhard-Karls-Universitä t zu Tübingen, 1997.
9. Fößmeier, U., Kaufmann, M.: *Algorithms and Area Bounds for Nonplanar Orthogonal Drawings.* In G. Di Battista, editor, Proc. 5th Int. Symp. on Graph Drawing (GD'97), volume 1353 of LNCS, pages 134-145. Springer, 1997.
10. Garg, A., Tamassia, R.: *A New Minimum Cost Flow Algorithm with Applications to Graph Drawing.* In S.C. North, editor, Proc. 4th Int. Symp. on Graph Drawing (GD'96), volume 1190 of LNCS, pages 201-216. Springer, 1997.
11. ILOG CPLEX 8.1: `http://www.ilog.com/products/cplex/`
12. Tamassia, R.: On embedding a graph in the grid with the minimum number of bends. *SIAM Journal on Computing*, 16(3), pages 421–444, 1987.
13. Yildiz, C.: *Knickminimales Orthogonales Zeichnen Planarer Graphen im Kandinsky Modell.* PhD thesis, Vienna University of Technology, 2006.