Non-Planar Orthogonal Drawings with Fixed Topology

Markus Chimani, Gunnar W. Klau
and René Weiskircher

Forschungsbericht / Technical Report

TR–186–1–04–03

25. August 2004

Non-Planar Orthogonal Drawings with Fixed Topology

Technical Report TR 186 1 04 03

Markus Chimani, Gunnar W. Klau, and René Weiskircher

Institute of Computer Graphics and Algorithms,
Vienna University of Technology, Austria
{mch|gunnar|weiskircher}@ads.tuwien.ac.at

Abstract. We present a procedure for calculating the bend minimal shape of non-planar graphs with given topology. The method is an extension of the Simple-Kandinsky drawing standard – a simplification of the more complex Kandinsky standard. For both models, algorithms exist that guarantee bend minimality for planar graphs with given topology. They generate orthogonal drawings with equal node size where multiple edges can be attached to a single side of a node. In contrast to Kandinsky, Simple-Kandinsky has certain restrictions on the split up of such bundles.
In this paper we extend the drawing standard to the class of non-planar graphs and present an algorithm that computes provably bend-minimal drawings within this standard. The extension treats edge crossings in a special way by letting them share identical grid points where appropriate. Hence it allows crossings of whole bundles of edges instead of single edges only. Besides having a reduced number of bends, drawings computed by our algorithm are easier to read and consume less area than those produced by the traditional approaches.
Furthermore, we show a sharp upper bound of the bend count for the heuristic use of Simple-Kandinsky for non-planar graphs and present an extension of the new method that is able to draw non-planar clustergraphs.

1 Introduction

We assume that the reader is familiar with the principles of graph drawing and the 3-phased *topology-shape-metrics* approach, which breaks the drawing problem into three subproblems: The first phase (planarization) calculates a topology for which we optimize the shape of the graph in the second phase (bend-minimization). The final phase (compaction) assigns lengths to the edge segments.

In this paper we propose an extension of the well-known Simple-Kandinsky drawing model, which is also known as Simple-Podevsnef [2]. This standard is a simplification of the more complex Kandinsky/Podevsnef standard (**P**lanar **o**rthogonal **d**rawing with **e**qual **v**ertex **s**izes and **n**on-**e**mpty **f**aces) [6]. Our extension of the original model – which only deals with planar graphs – guarantees true bend-minimality for non-planar graphs (for a given topology). We accomplish this by a special treatment of the artificial *dummy-nodes* that are introduced in the planarization phase.

To our knowledge, our algorithm, which is called SKANPAG (**S**imple-**Kan**dinsky for **N**on-**Pla**nar **G**raphs), is the first algorithm that can draw non-planar graphs with the minimum number of bends in the Simple Kandinsky model. We achieve this by letting dummy-nodes, which represent crossings, share positions on the drawing grid. The main advantages of our new approach are the reduction of bends, reduction of the area consumed by the drawing and an increase in the overall readibility of the resulting drawings.

Although our method is based on an integer linear program (ILP), its running time is very low in practice, even for large and dense graphs – drawing, for instance, the K_{25} takes under 50 seconds. When SKANPAG is applied to planar graphs, it is equivalent to Simple-Kandinsky. Our computational experiments on graphs generated from a widely used set of benchmark graphs show that the number of bends and the area saved by using our new approach is significant for dense graphs compared to the application of the standard Simple

Kandinsky method. We also present a few example drawings that show that the drawings produced using our new approach are easier to read.

The paper is structured as follows: First, we give a brief introduction to the Simple Kandinsky model (Section 2). In Section 3 we present our contribution to the theoretical background of bend minimality for non-planar graphs. Here, we also show the key situations where a straight forward flow approach would fail. Section 4 describes how we circumvent these situations and gives a description of an algorithm to solve the bend minimization problem for non-planar graphs with given topology. In Section 5 we give a sharp quality guarantee bound for the number of bends if we use classic Simple-Kandinsky as a heuristics for non-planar graphs. We present our computational experiments in Section 6 where we show that SKANPAG computes drawings with fewer bends and better area consumption in short computation time. We conclude with Section 7 where we also present our ideas for further research in this field.

Since most proofs are quite technical and involved, we present their main concepts in Appendix A, B, and C. For details, see [3].

2 The Simple-Kandinsky Model

In the following, we assume familiarity with Tamassia's approach to bend minimization [9]. The original method is restricted to graphs with maximum node degree four.

Several attempts have been made to extend the method to the larger class of planar graphs with arbitrary node degrees. We focus on a model where vertices are drawn with uniform size. Fößmeier and Kaufmann have introduced the *Kandinsky* drawing standard in [6]. By using a fine grid for the edges and a coarser grid for the vertices it is possible that several edges emanate from the same side of a node, forming $0°$ angles between them. Furthermore, the standard does not allow faces with angle-sum $0°$. The authors present a minimum-cost flow approach with additional constraints on the flow that can be realized, for instance, using an ILP approach as in, e.g., [5] or in the AGD library [7].

Recently, Bertolazzi, Di Battista, and Didimo have proposed a simplification of this standard – which we will refer to as the *Simple Kandinsky standard* – by adding the following restrictions:

(S1) Only high-degree nodes are allowed to have multiple edges leaving on the same side of a node

(S2) For each two neighboring edges that leave a node on the same side, the first bend on the right-most edge has to be a right bend.

The authors present a polynomial-time algorithm based on a network flow model that computes bend-optimal drawings of planar graphs with fixed topology in this standard. In the remainder of this section, we present a slightly modified but equivalent flow model for this task. Although our model is larger by a constant factor, it is advantageous for our purposes due to its simplicity, consistency, and straightforward extendability.

Let $G = (V, E)$ be the graph to draw with given topology, characterized by the face set F with outer face f_o. We create an underlying min-cost-flow network with two different types of directed edges: (a) arcs from the vertices $v \in V$ to incident faces $f \in F$ with zero cost, and (b) arcs with unit cost between adjacent faces.

Each unit transported in this network corresponds to a $90°$ bend. Hence, an arc of type (a) without flow implies a $0°$ angle. The capacities of the arcs as well as the supplies and demands of the nodes in the network are straightforward.

To satisfy constraints (S1) and (S2) (and therefore guarantee valid drawings) we have to apply an augmentation to the network for each high-degree node in V (cf. Fig. 1): We add a cyclic substructure of demand-free nodes. Each of these nodes is the target of an arc of type (b) which causes right bends and an arc of type (a). Furthermore, we insert arcs between the new nodes and the faces surrounding the high-degree node; these arcs have a lower flow bound of one unit. The construction guarantees that if two edges leave a node on the same side, forming a $0°$ angle, the right edge of this bundle has to have a right bend.

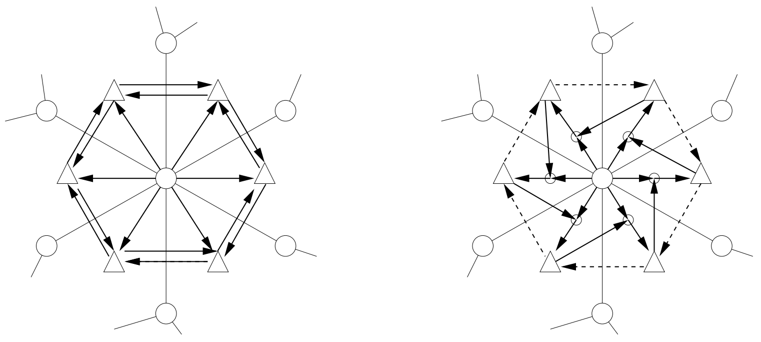


Fig. 1. Underlying network construction (left) and correctly augmented network (right); (circles and triangles represent nodes and faces, resp., dashed arcs are unchanged by the augmentation, irrelevant arcs are not shown)

3 Theory of Hyperfaces

The key to bend-minimality for non-planar graphs is to allow certain dummy-nodes to share a grid point (cf. Fig. 2). Nodes that share a grid point are said to be *merged*. Faces that become empty by such a merge, are said to be *collapsed*.

This section focuses on the cases where dummy merging is possible. Furthermore, it discusses the sources of errors that are potentially introduced when we allow merging. In Section 4 we give details on how we circumvent these errors.

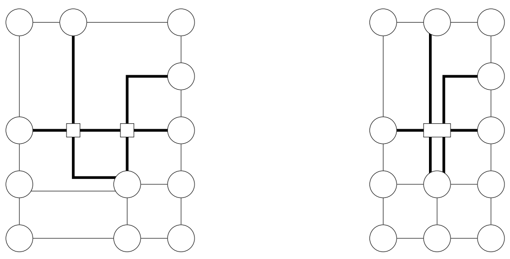


Fig. 2. Example for dummy merging: (left) Simple-Kandinsky, (right) SKANPAG; (original nodes are circles, dummy-nodes are squares)

If not stated otherwise, let $G_{orig} = (V_{orig}, E_{orig})$ be a non-planar, simple, and self-loop-free graph. The graph $G = (V, E)$ denotes the planarization of G_{orig} (based on a topology T_{orig}). Let T be the planar topology of G the drawing should be based on, and F the face set implied by T. We have $V = V_{orig} \cup D$, where D is the set of the dummy-nodes introduced by the planarization. If an edge $e_{orig} \in E_{orig}$ is split up into several subedges $e_i \in E$ during the planarization, we call the set $\{e_i\}$ a *metaedge* (of any e_i).

The most general approach to extend Simple-Kandinsky for non-planar graphs, is to demand its properties only for the underlying non-planar graph G', not for the temporary planarized graph G. This means that we force the right bend on bundles only for metaedges.

This leads to the concept of *hyperfaces* (cf. Fig. 3): A hyperface f^H is an ordered set of faces which starts with a *collapsible triangle* (*"coltri"*), a face f with exactly three incident nodes. One of these nodes is an original node and is called the *source node* of the hyperface. We call the edges incident to this coltri and to the source node e_1 and e_2. After the coltri, a hyperface may contain any number of *collapsible quads* (*"colquads"*). These are faces with four incident nodes, all of which are dummy-nodes. The final face of a hyperface is any face which is neither a coltri nor a colquad. Note that we only discuss simple graphs; hence

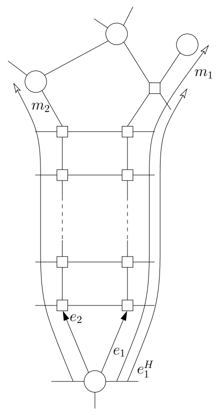


Fig. 3. Metaedges (m_1, m_2), hyperface, and hyperedge (e_1^H); (circles denote original nodes, squares are dummy-nodes)

a hyperface cannot contain two coltris. All subfaces of the hyperface are incident to subedges that are part of the metaedges of e_1 and e_2.

The pairs of dummy-nodes that are shared by two subfaces of a hyperface can potentially merge into a single grid point (cf. Fig. 2). Consider two metaedges leaving a high-degree node on the same side: We can merge the dummy-nodes that lie on the border of the hyperface as long as there are no bends which cause a split up (*"demerge"*).

If not stated otherwise, we will give directions on the hyperface in its *natural orientation*, where the coltri is on the bottom (as in Fig. 3). We assume that m_1 (the metaedge of e_1) is on the right side of the hyperface. To satisfy the Simple-Kandinsky constraints, we have to assure that m_1 has at least one right bend before any other bend, and before m_2 (the metaedge of e_2) has a right bend.

It is clear that such a right bend has to happen before we would be forced to merge a dummy-node with an original node. Hence we can specify the constraint that such a right bend has to happen on the *hyperedge*. The hyperedge e_1^H is an ordered subset of the metaedge m_1 and contains all the edges of the metaedge that are on the boundary of subfaces of the hyperface (see Fig. 3). We call the analogously defined subset of m_2 the *partner edge* of the hyperedge.

We can summarize these observations: SKANPAG has to force a right bend on each hyperedge, if its respective coltri has an opening angle of 0°. We can merge dummy-nodes if and only if their two metaedges leave a node on the same side, and did not have any bends that would cause a split up of the bundle.

3.1 Sources of Errors in Hyperfaces

While in Simple-Kandinsky the simple right-bend rule is enough to guarantee valid drawings, this is no longer true for SKANPAG . This section discusses the situations that could render a correct drawing of hyperfaces impossible. Section 4 will describe how we can eradicate these errors. Note that if all hyperfaces are drawn correctly, the complete graph will be drawn correctly (see the proof of Simple-Kandinsky's validity in [2]).

This section uses the min-cost-flow approach described earlier; it uses the Simple-Kandinsky augmentation presented in Section 2 only for faces adjacent to high-degree vertices that do not belong to a hyperface. For all hyperedges, it contains the demand for a right bend (in case of a 0° opening angle). Details are given in Section 4.

Let f^H be a hyperface, consisting of a coltri f_0, the colquads $f_1, \ldots, f_{n-1}$, and a final face f_n. Let $e_0, \ldots, e_n$ be the respective subedges of f_0's hyperedge e_H and $p_0, \ldots, p_n$ their partner subedges.

If the coltri has an opening angle of 90°, the drawing of the hyperface is always valid (this follows from correctness of Simple-Kandinsky). Hence we will only consider the case of a 0° opening angle.

Note that in this case, the coltri is balanced by default: It has a demand of two units, which are automatically sent by the incident dummy-nodes (dummy-nodes have a degree of four, and therefore send one unit into each incident face). The colquads are also balanced by default: their demand of four units is satisfied by their four incident dummy-nodes. Hence each unit that enters a coltri (colquad) from an adjacent face, must leave it into an adjacent face.

We show the possible erroneous flows by iterative enumeration based on the length of the hyperface. A hyperface is drawable, if each of its elements (*subfaces*) is drawable. If a subface f_i *demerges* (its lower pair of dummy-nodes is merged, but its higher pair is not), all faces $f_{>i}$ start demerged and are therefore drawable. Hence errors can only occur in collapsed faces and in the demerging face.

Iteration Start A coltri is drawable, if it either demerges or if its incident edges contain no bends. Note that – following the Simple-Kandinsky paradigm – every right bend on an edge will happen prior to any left bends.

The bends that would demerge a coltri are a right bend on the right side or a left bend on the left side. If either of these bends happen "early enough", it generates enough space for any other bends that might follow. Hence the bends that could invalidate the shape are: a left bend on the right side, a right bend on the left side, and any bend between the coltri and its subsequent colquad.

As mentioned earlier, each bend corresponds to a flow. Figure 4(a) shows a matrix of the interesting cases: The first line considers left bends on the right side, the second line right bends on the left side. The third line shows *upflows* (flows from f_0 into f_1), and the fourth *downflows* (flows from f_1 into f_0). Figure 4(c) gives the corresponding explanations regarding their validity. The solid arrows represent the flow that could potentially invalidate the resulting shape. The erroneous cases are encircled.

Note that there are cases when a face sends a unit to its adjacent face which sends it back again. Although this situation seems like a contradiction, it occurs in bend-minimal drawings (both in SKANPAG and in Simple-Kandinsky) to generate necessary right bends. We call such a situation a *FlowReFlow (FRF)*. Figure 4(a) shows the two types of FRFs: the situations in the center of row 3 and 4 are *updown-FRFs* since they contain an up- and a downflow; the FRF in the second row is a *left-FRF*.

Hence we know the two different kinds of errors that can invalidate a coltri with an opening angle of 0°: a downflow into the coltri, and a left-FRF.

Iteration Step We can detect an error at a colquad f_i, if there either was an error on f_{i-1}, or if f_i contains an error itself. The errors that can happen at a colquad are basically the same as the ones at the coltri: the matrix in Fig. 4 has to be extended by an additional column with either a down- or an upflow between f_{i-} and f_i. These situations would result in an error detected in f_{i-1} or in a demerge of f_{i-1}, respectively. Hence colquads have the same error types as coltris.

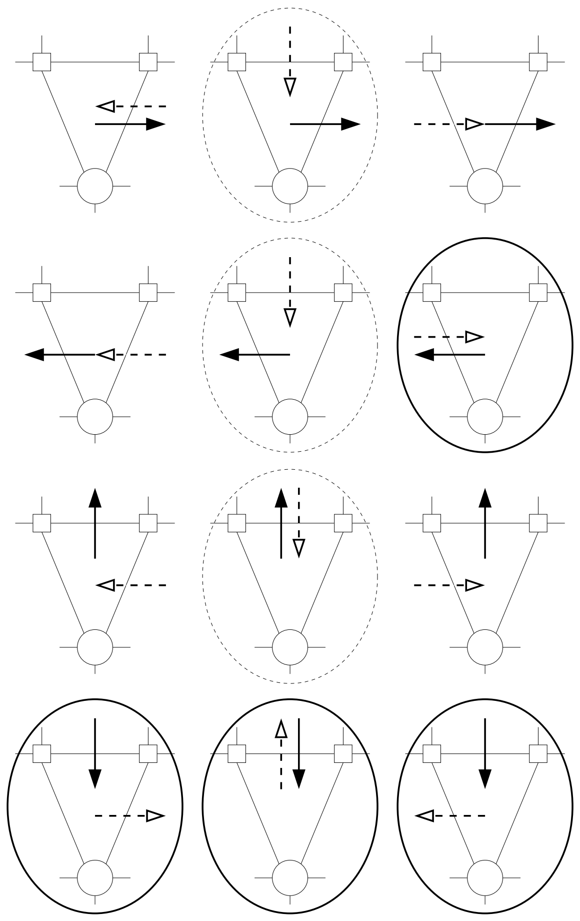
Iteration Stop We assume that there are no errors in the subfaces $f_{<n}$. The final subface f_n can only produce errors if it starts collapsed (its pair of dummy-nodes are merged). This implies that all subfaces $f_{<n}$ are collapsed. Since we demand a right bend on the hyperface, and such a bend would demerge a face, we know that this bend has to happen on f_n. Such a bend generates enough room for any following bends and therefore demerges f_n in a valid way. Thus f_n can never generate errors.

Summary of Erroneous Flows The above enumeration shows that the cases which we have to prohibit to guarantee valid drawings of non-planar graphs are the following:

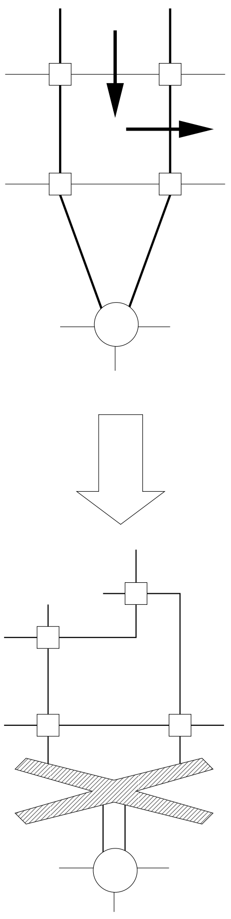
- Downflows into a collapsed subface of any hyperface
- left-FRFs on a collapsed subface of any hyperface

4 Solution Scheme

To solve the bend minimization problem, we use an ILP that models the underlying min-cost-flow network and contains the right-bend-on-hyperedge constraint. It also contains two additional types of constraints



(a) The erroneous flows in a coltri are encircled; since analogous errors recur, one of each type is encircled with a solid line.



(b) Example: A downflow generates an undrawable shape

row		left		center		right
1	V	right bend on e	E	bend on q, left bend on e	V	left bend on p
2	V	right bend on e	E	bend on q, right bend on p	E	right bend on p before left bend
3	V	right bend on e	E	bends on q	V	left bend on p
4	E	bend on q, left bend on e	E	bends on q	E	bend on q, right bend on p

(c) Explanations for the situations depicted in (a); V = valid, E = error; e = subedge of hyperedge, p = partner(e), q = top edge of coltri

Fig. 4. Sources of errors

to prohibit most of the erroneous situations described above. These two classes can also be described as augmentations in a standard min-cost-flow network. We use an ILP instead of a flow network because of the right-bend-on-hyperedge constraints, which we cannot model in such a network.

After solving the ILP, we apply a polynomial time repair-function on the solution. It guarantees that the objective value remains the same, but all remaining situations with erroneous flows are eliminated.

As an alternative to the repair-function, we could include additional constraints in the ILP, but this would introduce new 0/1-variables. The repair-function provides a much better runtime performance than solving the ILP with the additional variables.

The Fundamentals of the ILP We formulate the underlying min-cost-flow network (see Section 2) as a linear program, and add the following constraint for each hyperface: The sum of the flow that defines the opening angle of the face and the flows that generate right bends on the respective hyperedge has to be greater or equal than 1.

We have to demand that the variables involved in this constraints are integral. For all faces surrounding high-degree nodes that are not collapsible we demand the classic Simple-Kandinsky right-bend property.

Prohibiting Effective Downflows The first additional class of constraints excludes so-called *effective* downflows. These are downflows that are not part of an updown-FRF and therefore effectively transport units into the hyperface's subedge. By careful enumeration of the possible cases (see Appendix A), we can show that effective downflows can be prohibited without losing all optimal solutions. We can show that for each valid solution that contains effective downflows, we can find a related valid solution without effective downflows having the same number of bends.

Therefore, we add the constraint that for every position on all hyperfaces, the upflow has to be at least as strong as the downflow. This constraint can also be incorporated in a standard flow network by making sure that only the flow on an upflow arc can use a corresponding downflow arc.

Prohibiting Left-FRFs As shown in Section 3.1, we know that left-FRFs cause errors if they are attached to a collapsed subface of a hyperface. Hence we can only allow them on subfaces where the hyperface demerges or above this position. Additional proofs (summarized in Appendix B) show, that we can narrow the allowed positions further, without loosing all optimal and valid solutions: It suffices to allow left-FRFs only on positions where the affected subface f_i either contains a right bend on its right side or a "flow from below". The definition of the latter depends on f_i: if $i = 0$ it means an opening angle of 90°, otherwise it is the effective upflow (upflow minus downflow) from f_{i-1}.

We can ensure that these positions are allowed (and prohibit all other positions) by the following constraints for every position on all hyperfaces: The flow that generates a right bend on the left side has to be smaller or equal than the sum of the "flow from below" and the flow that causes a right bend on the right side. Again, this constraint – together with the prohibition of effective downflows – can be modeled with normal min-cost-flow techniques.

Updown-FRFs and the Repair-Function The ILP containing the additional constraints we just described already prohibits most types of errors. The only remaining type is a downflow that is part of an updown-FRF. Such an FRF is generated by the need for a right bend on an additional hyperedge/hyperface that is orthogonal to the one considered. Note that several updown-FRFs may occur *stacked* on each other and have to be repaired simultaneously to guarantee validity.

The repair-function itself – as well as the corresponding proofs, which are based on a detailed study of the possible cases – is quite technical and complex. We will give a short overview on the techniques used; details can be found in Appendix C.

The idea behind the repair-function is to detect stacks of invalid updown-FRFs and to try to move them to another position. Such a move has to ensure that all bend properties and constraints are still satisfied afterwards, but that the FRFs do not invalidate the drawing anymore. There are certain situations where a simple movement does not suffice. In these cases, the repair function may split some FRFs up into two distinct flows and move them separately.

Note that all operations by the repair-function only involve movement of already defined bends. They do not introduce new bends. Hence the objective value and the optimality of the solution remains.

The repair-function has an upper time bound of $O(h^2 l_{max}^2)$, where h is the number of hyperfaces and l_{max} the cardinality of the longest hyperface. This bound can be estimated more generously as $O(E^2)$.

5 Quality Guarantee of Simple-Kandinsky

Prior to SKANPAG, the only way to draw non-planar graphs was to use a planar drawing standard as a heuristics. We look at the special case of using Simple-Kandinsky, because of its strong relationship to SKANPAG . Furthermore, Simple-Kandinsky seems to be a good compromise between simplicity, execution time, and quality. Hence it is quite interesting to assess the quality difference between Simple-Kandinsky used as a heuristics and SKANPAG on non-planar graphs.

Note that applying Simple-Kandinsky or SKANPAG to planar graphs results in drawings with the same number of bends. Since there are no hyperfaces in planar graphs, there are no additional constraints, and hence no difference in the solution.

Also note that Simple-Kandinsky can never generate better solutions than SKANPAG: Each valid Simple-Kandinsky solution defines a valid SKANPAG solution with the same number of bends because Simple-Kandinsky's right-bend constraint is a specialization of the corresponding constraint for hyperfaces in SKAN-PAG.

Theorem 1. *For any given planarized graph $G = (V, E)$ and its planar topology T with h hyperfaces, Simple-Kandinsky needs at most h more bends than* SKANPAG*.*

Proof. We assume that SKANPAG has calculated a valid shape. The following observation holds true for every hyperface h_i: If the opening angle of h_i is 90° or if there is a right bend on the first subedge of h_i, this shape is valid for Simple-Kandinsky, too.

Now we assume that the coltri has an opening angle of 0° and there is no right bend on the first subedge of the hyperedge of h_i. Hence this shape is not a valid Simple-Kandinsky shape (Fig. 5(a), left). But we can achieve a similar Simple-Kandinsky shape by increasing the opening angle from 0° to 90° and adding one right bend on the bundle partner of h_i (Fig. 5(a), right).

This transformation is not influenced by collapsed coltris to the left of h_i, since these have to be extended by analogous bends themselves to become Simple-Kandinsky compliant. Neither does the transformation influence its surrounding area, since its overall shape remains the same.

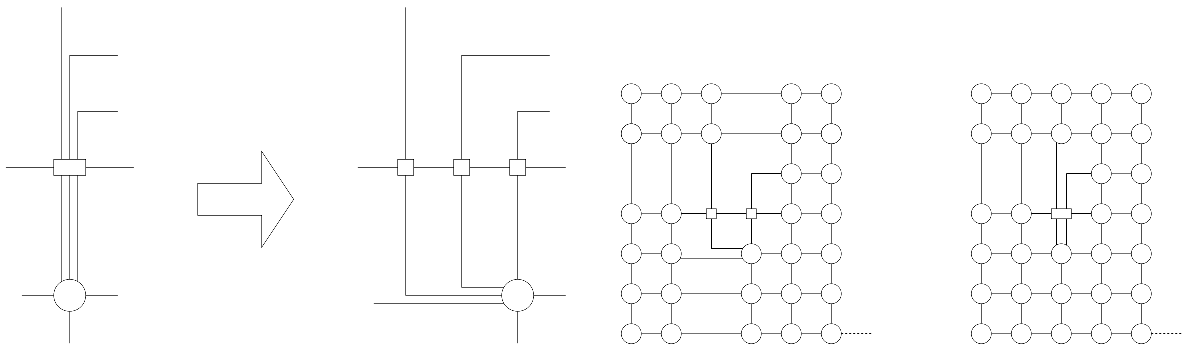
Hence we need at most one more bend for each hyperface to transform the shape generated by SKANPAG into a valid Simple-Kandinsky drawing. □

Corollary 1. *The bound given in Theorem 1 is sharp.*

Proof. Fig. 5(b) shows the graph used as a building block. We can put an arbitrary number of these blocks next to each other (joined by a simple edge, represented by a dotted line) and every building block has exactly one hyperface. For every block, Simple-Kandinsky needs two bends, but SKANPAG needs only one. Hence Simple-Kandinsky needs exactly h bends more than SKANPAG . Note that each block is optimally planarized, since there exists no other planarization generating less than two dummy-nodes per block. □

6 Computational Results

The *Rome graphs* [4] are a well established collection of about 11600 graphs based on 112 graphs taken from real-world applications. They have between 10 and 100 nodes each, and most of them are non-planar. But even the non-planar graphs are quite sparse and have therefore very few dummy-nodes after the planarization


(a) A valid hyperface in SKANPAG may need one more bend in Simple-Kandinsky

(b) Example where the heuristic solution is as bad as it gets

Fig. 5. Simple-Kandinsky as a heuristics

step. Nearly 7000 graphs do not have any hyperface at all and 1500 others contain only one. Thus we know from Theorem 1 that we cannot expect big differences between Simple-Kandinsky and SKANPAG . The implementation of our new method solves all these graphs without any need for the repair-function, nor does the LP-relaxation ever produce non-integer solutions.

Since this test suite is too sparse to show the difference between SKANPAG and Simple-Kandinsky, we *squared* each of the Rome graphs. The resulting graphs are still not extremely dense, but have more hyperfaces: only about 1200 still have none, and there are graphs with up to 280 hyperfaces.

The statistics compare SKANPAG to Simple-Kandinsky, if not otherwise stated. As Fig 6(a) shows, we need nearly 10% less bends in the average case for big graphs, with peak values of up to 25%. The size of the drawing (Fig 6(b))is reduced by over 20% on average. In some cases, we reduce the area consumption by 60%. As Fig 6(c) shows, the runtime performance of SKANPAG is acceptable even for large and dense graphs.

We also tested SKANPAG with complete graphs (up to K_{30}), to demonstrate the quality advantage for dense graphs. Note that the planarization of K_{30} has nearly 11000 dummy nodes. Fig 6(d) shows that we can save 15%-20% of bends for all such graphs with over 11 nodes; the area savings are even higher (up to 50%, see Fig. 6(e)). Figure 6(f) shows the runtime performance of SKANPAG and Simple-Kandinsky.

Figure 7 shows an example of a quite dense graph, drawn by Kandinsky, Simple-Kandinsky, and SKANPAG (equally scaled). More examples and details on the statistics can be found in [3].

7 Conclusions and Further Work

This paper presented a new approach for drawing non-planar graphs that takes into account the special properties of dummy nodes. By the use of an integer linear program, it guarantees the minimum number of bends for any given topology following the Simple-Kandinsky properties. The algorithm is the first to tackle this problem and due to our polynomial time repair function, the runtime is acceptable even for large and dense graphs.

Note that our algorithm can also be used for drawing clustered graphs orthogonally. These are usually drawn by modeling the cluster boundaries as circles consisting of dummy nodes and dummy edges [1, 8]. By treating the dummy nodes on the cluster boundaries just like we treat the dummy nodes introduced in the planarization, we can achieve savings in bends and area compared to producing drawings with the previously known orthogonal drawing algorithms.

Unfortunately, the complexity of producing bend-minimal drawings in the SKANPAG model is still unknown. This – as well as the complexity proof for the Kandinsky model itself – will be an interesting field of future study.

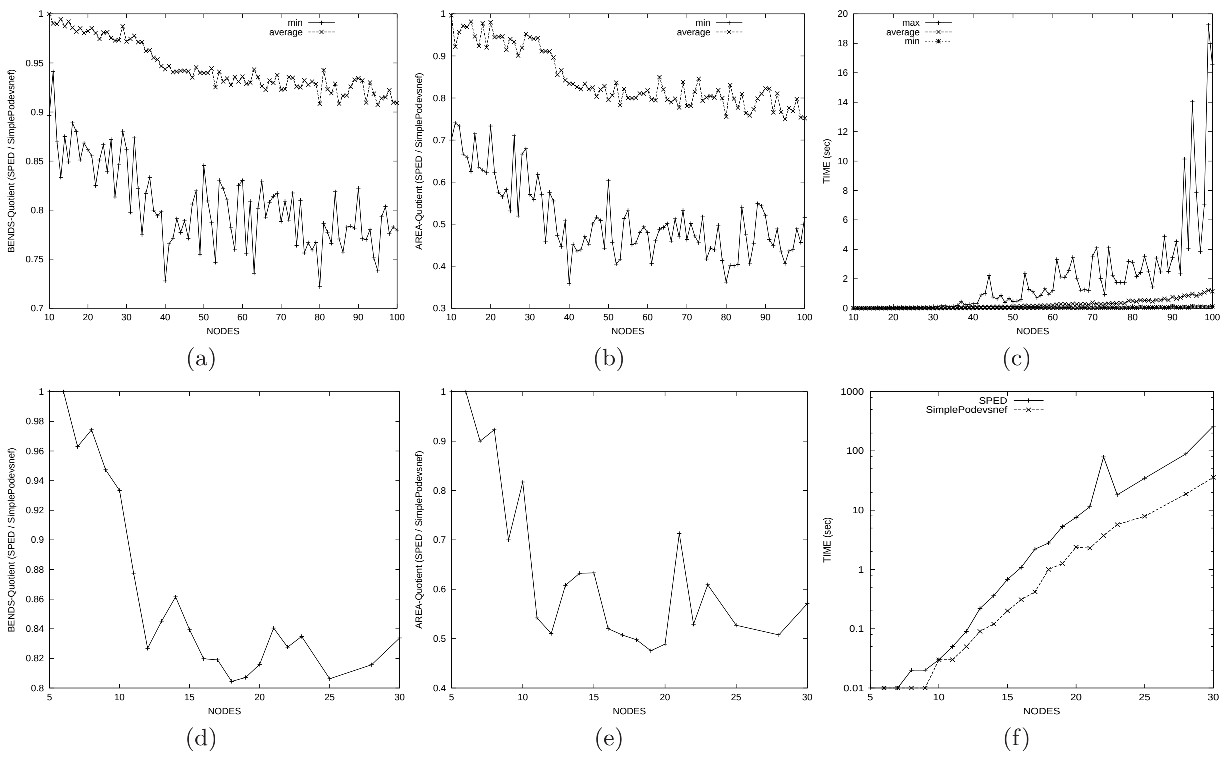
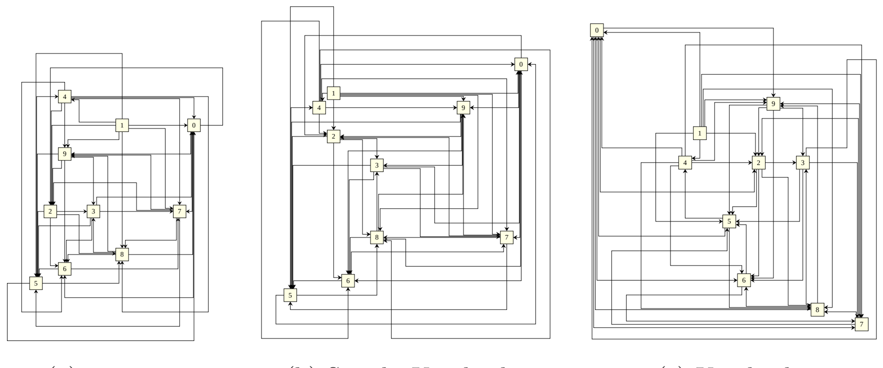


Fig. 6. Performance of SKANPAG, relative to Simple-Kandinsky. Top row: squared Rome graphs; bottom row: complete graphs. The peak at (f) is because the ILP-solver had to do several branches



(a) SKANPAG (b) Simple–Kandinsky (c) Kandinsky

Fig. 7. Graph with 10 nodes and 42 edges, drawn using three different drawing standards

References

1. G. D. Battista, W. Didimo, and A. Marcandalli. Planarization of clustered graphs. In *Graph Drawing (Proc. GD 2001)*, volume 2265 of *LNCS*, pages 60–74, 2001.
2. 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.
3. M. Chimani. Bend-minimal orthogonal drawing of non-planar graphs. Master's thesis, Vienna University of Technology, Department of Computer Science, Austria, 2004.
4. G. Di Battista, A. Garg, and G. Liotta. An experimental comparison of three graph drawing algorithms (extended abstract). In *Proceedings of the eleventh annual symposium on Computational geometry*, pages 306–315. ACM Press, 1995.
5. M. Eiglsperger, U. Fößmeier, and M. Kaufmann. Orthogonal graph drawing with constraints. In *10th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 1999)*, pages 3–11, 1999.
6. U. Fößmeier and M. Kaufmann. Drawing high degree graphs with low bend numbers. In F. J. Brandenburg, editor, *Proc. of the 3rd Int. Symp. on Graph Drawing (GD 1995)*, volume 1027 of *Lecture Notes in Computer Science*, pages 254–266, Passau, Germany, 1996. Springer.
7. M. Jünger, G. W. Klau, P. Mutzel, and R. Weiskircher. *Graph Drawing Software*, chapter AGD: A Library of Algorithms for Graph Drawing, pages 149–172. Mathematics and Visualization. Springer, 2003.
8. D. Lütke-Hüttmann. Knickminimales Zeichnen 4–planarer Clustergraphen. Master's thesis, Saarland University, Department of Computer Science, Saarbrücken, Germany, 2000.
9. R. Tamassia. On embedding a graph in the grid with the minimum number of bends. *SIAM J. Comput.*, 16(3):421–444, 1987.

A Proof: Effective Downflows can be forbidden

We know that any downflow into a collapsed face introduces an error. Hence they only arise in valid solutions if the lower of the involved faces is demerging or demerged. We can distinguish between several classes of downflows per hyperface (Fig. 8):

Class 1 Downflows:	Downflow that moves from one side to the other (from left to right/from right to left)
Class 2 Downflows:	Downflow attached only to the left/right
Class 3 Downflows:	Downflow coming from the end face and leaving to the left/right

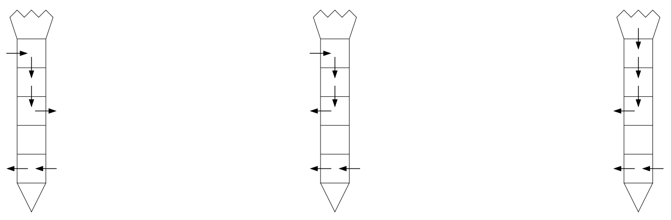

(a) Class 1 (b) Class 2 (c) Class 3

Fig. 8. Classes of valid non-FRF downflows (Nodes not drawn for clearity reasons)

We can show the following lemmata:

Lemma 1. *For any valid and optimal solution with class 1 (2) downflows, there exists a solution with equal objective value but without any class 1 (2) downflow. Furthermore, such a related solution does not introduce new class 2 (1) downflows.*

Lemma 2. *For any valid and optimal solution with class 3 downflows, there exists a solution with equal objective value but without any class 3 downflow. Furthermore, such a related solution does neither introduce new class 1 nor new class 2 downflows.*

The proofs for Lemmata 1 and 2 are straight forward and are based on giving the possible moves to transform every solution with the downflow of the specified class into a solution without these. Details are given in [3] From these lemmata we can deduce:

Lemma 3. *Only downflows that occur as a part of FRFs have to be allowed.*

Hence we can add a constraint that each upflow has to be at least as strong as its corresponding downflow. This constraints does not cut of all optimal solutions: at least one remains.

B Proof: Valid Positions of Left-FRFs

Because the ILPs objective function minimizes the bend count, such an FRF can only be caused by a second hyperface that lies directly to the left of the first one. Thus the halfedge on which the FRF lies is not only part of a partner edge, but also an element of an additional hyperedge. While such an FRF demerges the second hyperface, it may invalidate the first one (to its right), if that hyperedge is still collapsed at this stage.

So we have to assure that such a left-FRF only happens if the hyperedge is already demerged at this stage.

Lemma 4. *For every valid and optimal solution with a left-FRF on a hyperface with an opening angle of $0°$, there exists a solution with equal objective value, where the FRF lies on the same height as the first right bend on the hyperedge.*

Proof. We show that we can move the left-FRF to the stage where the (right) hyperface demerges. Such a move does not effect hyperfaces orthogonal thereto (if there are any): the FRF only looks like an updown-FRF to them, which we already considered irrelevant for now (see Appendix C).

The demerge of the (right) hyperface can either happen by a right bend on its hyperedge or by a left bend on its bundle partner. Thus in the first case we can clearly move the FRF downards to the place specified by the lemma.

If the right hyperface demerges by a left bend on the bundle partner, the left hyperface demerges at exactly the same stage or even below since it has no non-FRF downflows. Hence the left-FRF is not needed to assure a valid demerge but only satisfies the demand for at least one right bend on every hyperedge. Thus we can simply move this FRF to the position we want, as specified by the lemma. □

Hence it is certainly enough to allow left-FRFs to happen either:

1. On the same face as a right bend of the hyperedge (if the opening angle is 0°)
2. Directly on the coltri (if the opening angle is 90°)
3. On a face that has an effective upflow (= upflow that is not used in an FRF) coming from the face below (if the opening angle is 90°)

This restriction can be further weakened by allowing case 3 also to happen if the opening angle is 0°, because we know that effective upflows only happen if the face below is demerged.

Fortunately, such a constraint is quite easy to implement: The “bad” part of the left-FRF is the right bend on the partner edge; thus we only have to restrict it by demanding that the flow for this bend comes either from a right bend on the hyperedge or from “below” (as defined in Section 4).

C Proof: Applicability of a Repair-Function

We move each invalid updown–FRF somewhere where it does not invalidate any hyperface, but still correctly demerges the hyperface that needs it. The best place for such an FRF would be the first edge of the hyperedge they lie on:

Lemma 5. *Moving an FRF that causes an invalid downflow to the hyperface's subedge that is incident to the coltri (if this coltri is unambiguous) has the effect of removing any invalid downflows caused by it. (But it may generate an invalid left-FRF.)*

Proof. By moving it there, the corresponding hyperface demerges earlier. Hence such a movement does not introduce errors on the hyperface the FRF is needed for. Since the target subedge can never be an inner edge of any hyperface (coltris cannot be end faces of a hyperface), such a shift cannot result in an updown-FRF. □

So we only have to pay attention not to introduce an invalid left-FRF by such a shift. We also have to watch out for situations where such a shift is not possible at all: if the FRF causing the downflow-error is needed by two opposing hyperfaces (hence the coltri to move the FRF onto is ambiguous), it may not be possible to shift it to any coltri without introducing errors on the opposing hyperface.

For the first case we have to recognize that FRFs may be dependent on each other: they occur *stacked*, and thus have to be treated simultaneously.

There are three different types of FRF stacks; the third one is the conjunction of the other two (Fig. 9):

1. A bundle of neighboring hyperfaces with their respective coltris to the right of the hyperface containing the erroneous updown–FRFs
2. A bundle of neighboring hyperfaces with their respective coltris to the left of the hyperface containing the erroneous updown–FRFs
3. A block of hyperfaces that starts with a bundle of the first type and continues with a bundle of the second. The last of the first, and the first of the second bundle are opposing hyperfaces; both of them need the FRF.

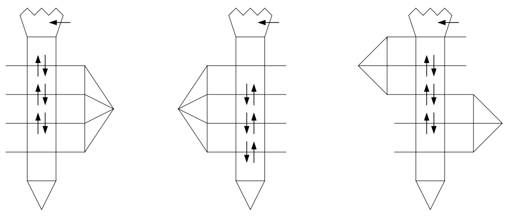


Fig. 9. The three types of FRF stacks

Note that bundles can consist of a single hyperface. Also note that a block consisting of a type 2 stack first, and followed by a type 1 stack, can never have a continuous sequence of FRFs due to minimization reasons.

It is lengthy, but straight forward to proof that type 1 and type 2 stacks can be moved to guarantee a valid and optimal solution. The proof for the repair of invalid type 3 stacks is somewhat more complicated, because it requires a split up of FRFs in certain situations. Furthermore it might "unveil" invalid updown-FRFs which has been valid before. But a careful consideration of the possible cases leads to a polynomial repair scheme even for this more complicated stack type [3].

Hence we can repair all invalid updown-FRFs within polynomial time bounds and without increasing the number of bend. Hence we guarantee a valid and optimal solution.