Orthogonal Compaction using Additional Bends

Michael Jünger[1], Petra Mutzel[2] and Christiane Spisla[2]
[1]*University of Cologne, Cologne, Germany*
[2]*TU Dortmund, Dortmund, Germany*

Keywords: Orthogonal Compaction, Graph Drawing, Total Edge Length, Bends, Flow-based Compaction.

Abstract: Compacting orthogonal drawings is a challenging task. Usually algorithms try to compute drawings with small area or total edge length while preserving the underlying orthogonal shape. We suggest a moderate relaxation of the orthogonal compaction problem, namely the *one-dimensional monotone flexible edge compaction problem with fixed vertex star geometry*. We further show that this problem can be solved in polynomial time using a network flow model. An experimental evaluation shows that by allowing additional bends we were able to reduce the total edge length and the drawing area.

1 INTRODUCTION

The compaction problem in orthogonal graph drawing deals with constructing an area-efficient drawing on the orthogonal grid. Every edge is drawn as a sequence of horizontal and vertical segments, where the vertices and bends are placed on grid points. Compaction has been studied in the context of the *topology-shape-metrics approach* (Batini et al., 1986). Here, in a first phase a combinatorial embedding is computed that determines the topology of the layout with the goal to minimize the number of crossings. In the second phase, a dimensionless orthogonal shape of the graph is determined by fixing the angles between adjacent edges and the bends along the edges. The goal is to minimize the number of bends, which can be done in polynomial time for a fixed embedding (Tamassia, 1987). In the third phase, metrics are added to the orthogonal shape. In this context, first the coordinates of vertices and bends are assigned to grid points so that the given orthogonal shape is maintained. Finally, the (orthogonal) compaction problem asks for a drawing minimizing geometric aestetic criteria, such as the area of the drawing or the total edge length. The shape is not allowed to change.

Since the orthogonal compaction problem is NP-hard (Patrignani, 1999), in practice heuristics are used that fix the x- (or y-, resp.) coordinates and solve the resulting compaction problem in one dimension. Given an initial drawing, the one-dimensional compaction problem with the goal of minimizing the height (or width, resp.) of the layout can be transformed to the longest path problem in a directed acyclic graph. If in addition the total edge length shall be minimized, the problem can be solved by computing a minimum cost flow.

The topology-shape-metrics approach aims at drawings with a small number of crossings, a small number of bends, and a small drawing area. These goals are addressed in this order. And indeed, compared with other drawing methods, the number of crossings and bends is relatively small (Di Battista et al., 1997). However, the layouts often contain large areas of white space. It seems that the goal of getting a small drawing area has not been achieved so far. Consider the drawing in Fig. 1(a) which contains large areas of white space due to shape restrictions. By introducing two bends on one of the edges the drawing area can be reduced drastically. This motivates us to study a compaction problem in which the shape conditions are relaxed.

This brings us back to the origin of the orthogonal compaction problem in VLSI-design (see, e.g., Lengauer (1990)). In contrast to the compaction problem considered in graph drawing, even the permutation of wires along the boundary of a component (and hence, changing the embedding) is allowed.

We suggest a moderate relaxation of the orthogonal compaction problem. More precisely, we suggest to study the *one-dimensional monotone* ***fle****xible* ***ed****ge compaction problem with* ***fi****xed* ***ve****rtex star geometry*, henceforth the *Fled-Five compaction problem*, which asks for the minimization of the vertical (horizontal, resp.) edge length and allows changing the orthogonal shape of edges, but preserves the x-monotonicity (y-

Jünger, M., Mutzel, P. and Spisla, C.
Orthogonal Compaction using Additional Bends.
DOI: 10.5220/0006713301440155
In *Proceedings of the 13th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications (VISIGRAPP 2018) - Volume 3: IVAPP*, pages 144-155
ISBN: 978-989-758-289-9

monotonicity, resp.) of edge segments and prohibits changing the directions of the initial edge segments around the vertices (vertex star geometry). We present a polynomial-time algorithm based on a network flow model that solves the Fled-Five compaction problem to optimality. Our computational results show that repeated application of Fled-Five compaction in *x*- and *y*-direction is able to reduce the total edge length and the drawing area at the expense of additional bends.

This paper is organized as follows. We recall the state of the art in Sect. 2 and some basic definition about orthogonal graph drawing and especially the compaction phase in Sect. 3. We present our new algorithm in Sect. 4 and evaluate it experimentally in Sect. 5.

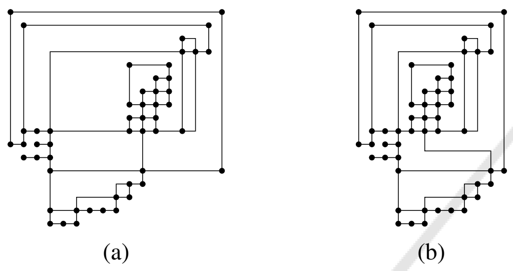


Figure 1: (a) A drawing with large areas of white space due to shape restrictions. (b) Introducing two additional bends to one edge of the drawing leads to a smaller drawing.

2 STATE-OF-THE-ART

Patrignani (1999) has shown that planar orthogonal compaction is in general NP-hard and Bannister et al. (2012) have given inapproximability results for the nonplanar case. But for some special cases there exist polynomial-time algorithms, e.g., if all faces are of rectangular shape (Di Battista et al., 1999) or if all faces are so-called *turn-regular* (Bridgeman et al., 2000) or have a unique completion (Klau and Mutzel, 1999). Klau and Mutzel (1999) suggested a branch-and-cut approach to solve an integer linear program based on extending a pair of constraint graphs.

However, in practice, heuristics are used which iteratively fix the *x*-, and then the *y*-coordinates, and solve the resulting one-dimensional compaction problem. This process is repeated until no further progress is made. One-dimensional compaction algorithms often use either network flow techniques or a longest path method in order to assign integer coordinates to the vertices, see, e.g., Kaufmann and Wagner (2001) for an overview. An experimental comparison of planar compaction algorithms was presented by Klau et al. (2001).

There has been some work to improve the quality of a drawing by changing its shape, e.g., Fößmeier et al. (1998) or Six et al. (1998). The former uses shifting and resizing vertices and modifies the shape of the edges in order to save area and bends, and the latter may additionally change the topology to improve the drawing. But most compaction algorithms for planar 4-graphs take as input an orthogonal representation and try to produce compact drawings with respect to that representation. This can lead to an unnecessarily large drawing area with unused space. Often better results in terms of area and edge length can be achieved if the orthogonal shape can be altered, as we have seen in Fig. 1. On the other hand, it might be desirable to not change a given drawing too much in order to preserve the mental map. In some applications, like layouting data flow diagrams, edges should start and end at specific points or sides of the vertices (Spönemann et al., 2010). These so-called *port constraints* would correspond to fixing the initial edge segments around vertices (vertex star geometry) in an orthogonal representation while leaving flexibility to the edges.

3 NOTATION AND PRELIMINARY RESULTS

In this section we give basic definitions and notations. For more details on orthogonal drawings and graph drawing in general see, e.g., Di Battista et al. (1999), Kaufmann and Wagner (2001) or Tamassia (2013).

3.1 Orthogonal Graph Drawing

For the rest of this paper we restrict ourselves to undirected *4-graphs*, i.e. graphs whose vertices have at most four incident edges. A graph $G = (V,E)$ with $|V| = n$ and $|E| = m$ is called *planar* if it admits a drawing Γ in the plane without edge crossings. Such a planar drawing of G induces a *(planar) embedding*, which is represented by a circular ordered list of bordering edges for every *face*. The unbounded region of a planar drawing is called *external face*.

An *orthogonal representation H* is an extension of a planar embedding that gives combinatorial information about the orthogonal shape of a drawing. For every edge we provide information about the bends encountered while traversing the edge and the angle formed at vertices by two consecutive edges. If one of these angles is 270° or 360° we associate with v a *reflex corner*. An orthogonal representation is called *normalized* if it has no bends. An *orthogonal grid* *drawing* Γ of G is a drawing in which every edge is drawn as a sequence of horizontal and vertical *edge segments* and every vertex and bend has integer coordinates. Such a drawing induces an orthogonal representation H_Γ and a *star geometry* for every vertex fixing the directions of the initial line segments of its incident edges. If an edge first turns to the right and then to the left, or vice versa, we call this a *double bend* and the edge segment between these two bends a *middle segment*. Every orthogonal representation can be normalized by replacing all bends in H_Γ with dummy vertices of degree two, thus adding vertices to G and Γ.

Since our new approach is based on a network flow model for one-dimensional compaction, we will give a brief introduction to minimum cost flows here. For more information about network flows, see Ahuja et al. (1993). Let $N = (V_N, E_N)$ be a directed graph. Whenever we talk about flows we will call N a *network*, the members of V_N *nodes* and the members of E_N *arcs* (in contrast to vertices and edges in an undirected graph). Every arc a has a *lower bound* $l(a) \in \mathbb{R}^{\geq 0}$, an *upper bound* $u(a) \in \mathbb{R}^{\geq 0} \cup \{\infty\}$ and a nonnegative *cost* $c(a)$. A *demand* $b(n) \in \mathbb{R}$ is associated with every node. We call a function $x : A \to \mathbb{R}^{\geq 0}$ a *flow* if x satisfies the following conditions:

capacity constraint:

$$l(a) \leq x(a) \leq u(a) \qquad \forall\, a \in E_N \qquad (1)$$

flow conservation:

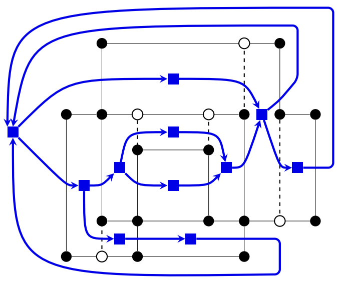
$$\sum_{a=(k,l)} x(a) - \sum_{a=(j,k)} x(a) = b(k) \qquad \forall\, k \in V_N \qquad (2)$$

A *minimum cost flow* is a flow x with minimum total cost $c_x = \sum_{a \in E_N} x(a)c(a)$ under all feasible flows. The minimum cost flows we are interested in can be computed in $O(|V_N|^{3/2} \log |V_N|)$ time (Cornelsen and Karrenbauer, 2012).

3.2 Compaction of Orthogonal Drawings

We focus on the *vertical (orthogonal) compaction problem* that receives as input a planar grid drawing Γ of a graph G with an orthogonal representation H_Γ, and asks for another planar orthogonal drawing Γ' of G realizing H_Γ so that the vertical edge length is minimized subject to fixed x-coordinates. In the following we describe a flow-based method for vertical compaction similar to the coordinate assignment algorithm in Di Battista et al. (1999).

Assume we have an initial grid drawing Γ. In a first step we normalize H_Γ resulting in $\overline{\Gamma}$ and $\overline{H}_\Gamma$. Then we add vertical *visibility edges* (so-called *dissecting edges*). We insert a vertical edge connecting each reflex corner with the vertex or edge that is visible in vertical direction, possibly introducing dummy vertices. This gives us $\widetilde{\Gamma}$ and $\widetilde{H}_\Gamma$. This way we eliminate all reflex corners and have a representation with rectangular faces. Now we are ready to


(a)

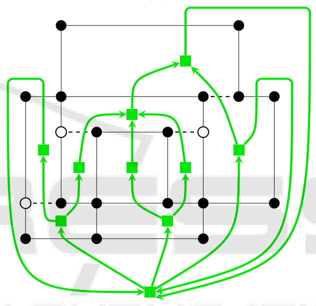

(b)

Figure 2: The flow networks (a) N_y for vertical compaction and (b) N_x for horizontal compaction. White vertices are dummy vertices and dashed edges are dummy edges.

construct the network N_y for vertical compaction. For each face f in representation $\widetilde{H}_\Gamma$ we add a node n_f to N_y with demand $b(n_f) = 0$ and for every vertical edge e with left face f_l and right face f_r we insert an arc $a_e = (n_{f_l}, n_{f_r})$ with lower bound $l(a_e) = 1$ and upper bound $u(a_e) = \infty$. If e is a dummy edge a_e gets zero cost, otherwise a cost of one. See Fig. 2 for an example.

Suppose we have computed a feasible flow. Now a unit of flow corresponds to one unit of vertical edge length. The x-coordinates remain unchanged. The capacity constraint assures that every edge gets a minimum length of one and the flow conservation constraint guarantees that every face is drawn as a proper rectangle. Finally, the visibility edges and dummy vertices can be removed.

Lemma 1. *The above algorithm solves the vertical compaction problem to optimality.*

Proof. In the vertical compaction problem we have to preserve the vertical visibility properties of Γ in order to avoid overlapping graph elements. This is achieved by the newly added visibility edges. Because of the one-to-one correspondence of vertical length of an edge segment in the resulting drawing Γ' and the amount of flow carried by a non-zero-cost arc $a_e \in N_y$, the result of the minimum cost flow gives us a minimal vertical length assignment. □

4 THE FLED-FIVE COMPACTION APPROACH

In this section we study the following relaxation of the one-dimensional compaction problem called the *monotone **fl**exible **ed**ge compaction problem with **fi**xed **ve**rtex star geometry* (*Fled-Five compaction problem*). For vertical compaction it is defined as follows: Given a planar grid drawing Γ of a 4-graph G with an orthogonal representation H_Γ, compute another planar grid drawing of G that minimizes the vertical edge length subject to fixed x-coordinates in which all horizontal edge segments of Γ are drawn x-monotonically and the vertex star geometry for all vertices as well as the planar embedding is maintained.

In contrast to the classical compaction problem, it is not required here to realize the entire orthogonal representation, but only the local geometric surrounding of each vertex. The fixed x-coordinates and the x-monotonicity prohibits the lengthening of the total horizontal edge length (see Fig. 3).

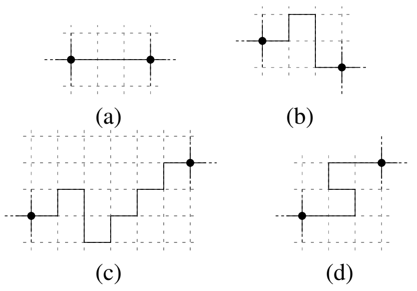


Figure 3: Modifications in Fled-Five. (a) The original edge of length three. (b) Modification allowed in Fled-Five. (c) Not allowed because of changed x-coordinates and (d) not allowed, because x-monotonicity is violated.

We adapt the network flow approach described in Sect. 3.2. But now we are able to introduce or remove double bends of certain edges in order to improve the vertical edge length. We illustrate the concept by the following example. Figure 4(a) shows an optimally compacted drawing with respect to its orthogonal representation. The blue arcs show the arcs of the vertical compaction network (compare Fig. 2). For better readability we omitted the network nodes belonging to the faces. In Fig. 4(b), the same graph is shown, but with a different orthogonal representation. The new double bend in the middle edge leads to a smaller drawing. In this drawing it is possible to send flow between the two internal faces, since they are separated by a vertical edge segment. In the corresponding flow network there is a new network arc (red, dashed) connecting the upper and the lower face. This leads to the key idea of our approach. Introducing arcs in the vertical flow network between horizontally separated face nodes enables us to shift flow between them and therefore exchange vertical edge length at the expense of a double bend (see Fig. 4(c)). We can also reverse this thought. If there is an unnecessary double bend we can get rid of it by not sending flow over the arc corresponding to its middle segment and thus removing the middle segment from the drawing.

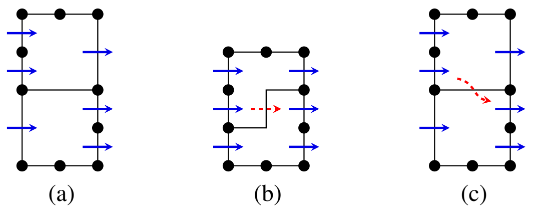


Figure 4: A graph and the arcs of the corresponding flow network (a,b) of the traditional algorithm and (c) in our algorithm.

But we have to be careful here. First, we are compacting in vertical direction, so we cannot change the x-coordinates. If an edge has a double bend, it needs to have a horizontal expansion of at least two. Thus we will not consider edges of length one as possible candidates for getting a double bend. Second, adding a double bend to e introduces two reflex corners, one in both adjacent faces of e. Two double bends may even cause an edge overlap, see Fig. 5(a). We need to ensure that the computed flow corresponds to a feasible drawing. Therefore we will treat each grid point along an edge that could be part of a double bend as a potential reflex corner, which we will eliminate by dissecting. We will now describe the algorithm in detail.

Our algorithm works in three phases: dissection, computation of a minimum cost flow and transforming the flow into a new drawing. First we normalize H_Γ to $\overline{H}_\Gamma$. For dissection we split the horizontal edges of $\overline{\Gamma}$ by placing an artificial *bend vertex* on every inner grid point of an horizontal edge. The bend vertices may later be transformed to double bends. Then we dissect $\overline{\Gamma}$ as described in Sect. 3 and treat every

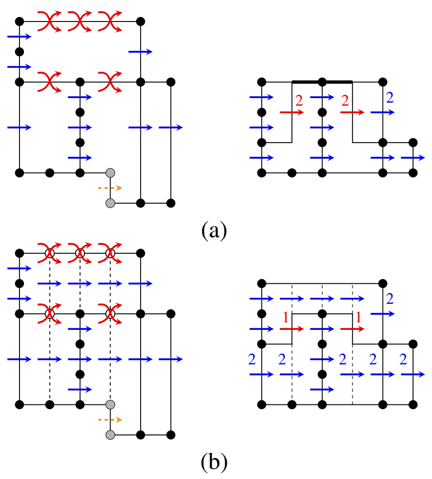


Figure 5: Input graph with modified flow network and a minimum cost flow (all unlabeled arcs carry one unit of flow) with the resulting drawing (a) without additional dissecting edges and (b) including them. The bold edge in (a) indicates an edge overlap and the grey vertices result from normalization, the white vertices in (b) are bend vertices. Dissecting edges are indicated in (b).

bend vertex as a reflex corner in its adjacent faces (see Fig. 5(b)). That means, we may dissect the drawing in vertical stripes of unit length. Doing so, we solve both of the problems mentioned above. Since we consider only edges of length at least two, we guarantee that the edge will be long enough for a double bend and by inserting the visibility edges we keep the vertical separation of graph elements intact. This results in the extensions $\widetilde{\Gamma}$ and $\widetilde{H}_\Gamma$ and $\widetilde{G}$.

Observation 1. *After this dissection method the number of vertices and edges in $\widetilde{G}$ is $O(A)$, where $A = h_\Gamma \cdot w_\Gamma$ and h_Γ (w_Γ) is the height (width) of Γ.*

Next we construct the network from Sect. 3.2. Additionally to the already introduced arcs (straight blue arcs in Fig. 5) we add two arcs $a_v^\uparrow$ and $a_v^\downarrow$ for every bend vertex (curved, red arcs in Fig. 5). Notice that every such bend vertex v has four adjacent faces, one at the lower and upper left and right. If it lies on the external face, two of the adjacent faces may coincide. Arc $a_v^\uparrow$ goes from the lower left face of v to its upper right face and $a_v^\downarrow$ goes from the upper left to the lower right face. Flow on one of these arcs will lead to a double bend in the edge segment of G that is split by v. The lower bound of these arcs is zero, the upper bound is set to infinity and the cost is one. For every edge $e \in \widetilde{G}$ that is a middle segment in G we decrease the lower bound of its arc a_e to zero, since this is a vertical edge segment, that we may delete (dashed, orange arc in Fig. 5).

After computing a minimum cost flow in this network, we interpret the result in the following way: For every vertical edge e of $\widetilde{G}$ we translate the amount of flow on the arc a_e of N_y to the length of e. Let $a_v^\uparrow$ be an arc corresponding to a bend vertex v carrying k units of flow. Let e be the split horizontal edge and x_v the x-coordinate of v. Then e will start at its left endpoint in horizontal direction, bend downwards at x-coordinate x_v, proceed for k units in y-direction and then continue to the right to its other endpoint. If we deal with an arc of the form $a_v^\downarrow$ the corresponding edge will do an upward bend at x_v. Let $e' \in \widetilde{G}$ be an edge that corresponds to a middle segment and let $a_{e'}$ be the corresponding arc. If $a_{e'}$ carries no flow we do not assign any vertical length to e', hence the double bend to which e' belongs collapses. Notice that flow on every non-zero-cost arc corresponds to vertical edge length. Theoretically, it is possible that both $a_v^\uparrow$ and $a_v^\downarrow$ carry flow in a feasible, but not optimal, solution. In this case we can always augment the flow in a way that at most one of $a_v^\uparrow$ or $a_v^\downarrow$ carries flow. Finally, we remove all dummy edges and vertices. See Fig. 5 for an example and Algorithm 1 for pseudocode.

Let us assume that the width and height of Γ are bounded by the number of its vertices and bends. Otherwise there would be a grid line without any vertex or bend on it. We can delete such grid lines iteratively until we reach our bound.

Theorem 1. *Let Γ be a planar grid drawing of a 4-graph G with an orthogonal representation H_Γ. Let $\bar{n}$ be the number of vertices and bends of Γ. Then the above described extended network-based compaction algorithm takes $O(\bar{n}^3 \log \bar{n})$ time and solves the vertical Fled-Five compaction problem to optimality.*

Proof. Because of the visibility edges we will maintain visibility properties of Γ. Every vertical edge segment of G that is not a middle segment gets a minimum length of one due to the lower bound of the corresponding arc in N_y. Every bend vertex can safely be turned to a double bend if its corresponding arc carries flow, since we add visibility edges to the top and bottom of it. By this modification we do not change the vertex star geometry of Γ, because bend vertices are not part of G. Every middle segment can be removed if there is no flow on the corresponding arc. Because its edge $e \in \Gamma$ has a horizontal expansion of at least two, e will still have a proper length and due to the dissection phase visibility properties are maintained. This modification also does not affect the vertex star geometry of Γ, since a middle segment is not adjacent to a vertex of G. So every face will have a rectangular shape, no matter what modification we apply, and due to flow conservation every face and thus the entire graph is drawn consistently. Similar to Lemma 3.2 the

```
Input  : orthogonal drawing $\Gamma$
Output: optimal solution $\Gamma'$ to the Fled-Five
        compaction problem
$\overline{\Gamma} \leftarrow$ normalize($\Gamma$);
$\widetilde{\Gamma} \leftarrow$ addBendVerticesAndVertVisEdges($\overline{\Gamma}$);
// construct network N
for every face $f$ do
    $N \leftarrow$ node $n_f$;
    demand[$n_f$]=0;
for every vertical edge $e$ do
    node $n1$ = leftFaceNode($e$);
    node $n2$ = rightFaceNode($e$);
    if isMidSegment(e) then
        lowerBound[$a_e$]=0;
    else lowerBound[$a_e$]=1;
    upperBound[$a_e$]=$\infty$;
    if isVisEdge(e) then cost[$a_e$] = 0;
    else cost[$a_e$]=1;
for every bend vertex $v$ do
    node $n1$ = lowerLeftFaceNode($v$);
    node $n2$ = upperRightFaceNode($v$);
    $N \leftarrow$ arc $a^{\uparrow} = (n1, n2)$;
    lowerBound[$a^{\uparrow}$]=0; upperBound[$a^{\uparrow}$]=$\infty$;
    cost[$a^{\uparrow}$]=1;
    node $n3$ = upperLeftFaceNode($v$);
    node $n4$ = lowerRightFaceNode($v$);
    $N \leftarrow$ arc $a^{\downarrow} = (n3, n4)$;
    lowerBound[$a^{\downarrow}$]=0; upperBound[$a^{\downarrow}$]=$\infty$;
    cost[$a^{\downarrow}$]=1;
// compute length assignment
$x \leftarrow$ computeMinimumCostFlow($N$);
for every vertical edge $e$ do
    length($e$) $\leftarrow x$(networkArc($e$));
for every bend vertex $v$ do
    edge $e \leftarrow$ horizontalEdge($v$);
    int $xc \leftarrow$ xCoordinate($v$);
    if $x(a_v^{\uparrow}) \neq 0$ then
        addDownwardMidSegment($e, xc, x(a_v^{\uparrow})$);
    if $x(a_v^{\downarrow}) \neq 0$ then
        addUpwardMidSegment($e, xc, x(a_v^{\downarrow})$);
$\Gamma' \leftarrow$ RemoveBendVerticesAndVisEdges($\widetilde{\Gamma}$);
```

Algorithm 1: verticalFledFive.

length of vertical segments of Γ' is equal to the cost of the computed minimum cost flow. Since the initial drawing can be interpreted as a feasible flow, we get a drawing Γ' with total edge length less or equal to that of Γ. The horizontal edge segments maintain their x-monotonicity, because we only add vertical segments to the drawing. Hence the horizontal edge lengths and the x-coordinates of the vertices in Γ stay the same. Because the minimum cost flow gives us a minimal vertical length assignment we have an optimal solution for the vertical Fled-Five compaction problem.

For the running time we first consider the dissection phase. By Observation 1 we end up with $O(\bar{n}^2)$ vertices. For inserting dummy edges based on visibility properties we can use a sweep-line algorithm that runs in $O(\bar{n}^2 \log \bar{n})$ time in our case. The construction of the flow network runs in linear time in the size of $\widetilde{G}$. The network is planar and has $O(\bar{n}^2)$ nodes. For planar flow networks with n nodes and $O(n)$ arcs there exists a $O(n^{3/2} \log n)$ time algorithm for computing a minimum cost flow (Cornelsen and Karrenbauer, 2012). Therefore we have a total running time of $O(\bar{n}^3 \log \bar{n})$. □

4.1 Short Remarks

Running Time. The cubic part of the running time comes from the possibly quadratic number of bend vertices and dissecting edges. So if we restrict the number of bend vertices to be linear, we can decrease the running time to $O(\bar{n}^{3/2} \log \bar{n})$.

Controlling the Number of New Bends. Although the above compaction approach may reduce the total edge length, the number of bends in the resulting drawing may increase. A possible approach to control the number of new bends could be to bound the number of specific arcs used by a feasible flow. This is known as the binary case of the *budget-constrained minimum cost flow problem* for which Holzhauser et al. (2016) showed strong NP-completeness. In fact, in this model we have no other control over the number of additional bends than restricting the number of bend vertices. Each such vertex can generate two new bends. By assigning higher cost to arcs $a_v^{\uparrow}$ and $a_v^{\downarrow}$ it becomes more unlikely for these arcs to carry much flow, but it does not limit its number used by the flow. It makes no difference in terms of total cost if we send, e.g., 10 units of flow over $a_v^{\uparrow}$ or one unit of flow over 10 such arcs, while the former produces two bends and the latter 20 bends.

Extensions to Other Models. While in a drawing model for 4-graphs a node occupies exactly one grid point, there are other models for drawing graphs with maximum degree greater four, e.g., the *Kandinsky* model (Fößmeier and Kaufmann, 1995), where essentially nodes spread over several grid points (big nodes) or the *quasi-orthogonal* model (Jünger et al., 2004), where edges are allowed to leave the grid around vertices. Our approach can be extented to

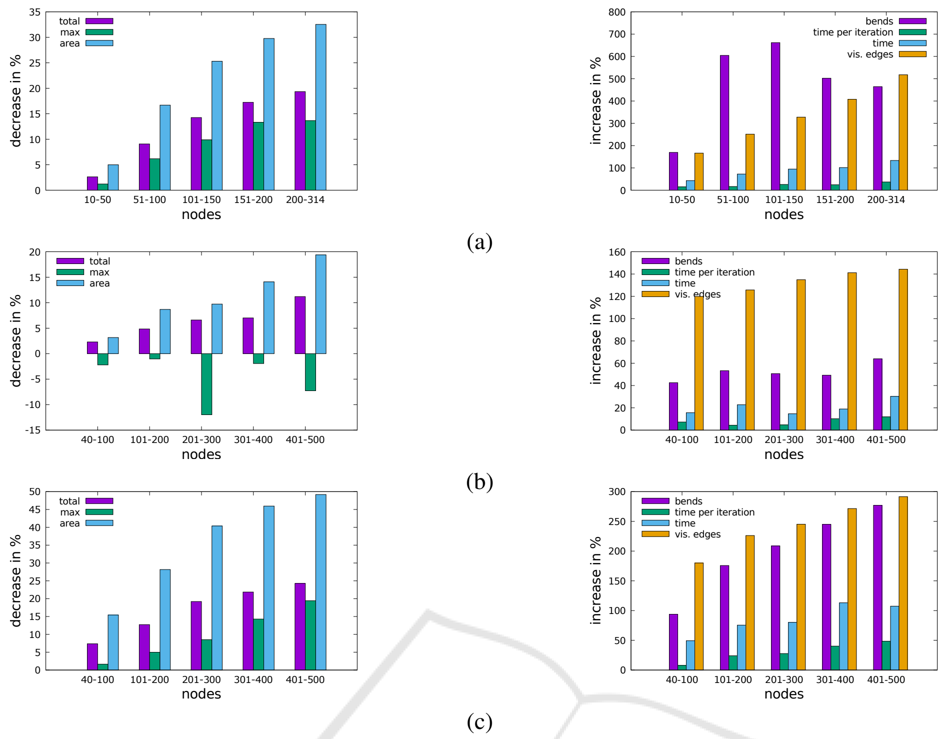


Figure 6: Average change of the total edge length, the maximum edge length, the area, the number of bends, the time per iteration, the total running time and the number of visibility edges from TRAD to FF in percent for (a) the Rome graphs, (b) the quasi trees, and (c) the biconnected graphs.

these models in a straightforward way by modeling big nodes as rectangular faces of suitable size for all the incident edges and prohibit changing the shape of the dummy edges that participate in these faces. Since in the quasi-orthogonal model, until the last step of actual drawing, high-degree nodes are also handled as faces, the extension works for this model as well.

5 EXPERIMENTAL EVALUATION

In this section we evaluate our algorithm FF focusing on the total edge length and area improvement. Klau et al. (2001) compared 12 compaction heuristics that maintain the orthogonal representation and we chose the best of these traditional heuristics (T2FL in Klau et al. (2001)), which we call TRAD, for comparison in order to demonstrate the benefit of slightly changing the orthogonal shape. This method was within one percent of the optimal value for practical instances. We refrain from comparing to improvement algorithms like the 4M algorithm (Fößmeier et al., 1998) or the refinement method from Six et al. (1998) since they change the vertex star geometry, the topology and even vertex sizes. The moving operation of the 4M algorithm is the only operation that fits our model, but would not change any layout generated by a flow-based compaction algorithm. The only applicable modification from Six et al. (1998) (superfluous bends) does not have an impact on bend minimal drawings that we use as input.

The algorithms were implemented in C++ using the OGDF library (Chimani et al., 2013). For both algorithms we start with a bend minimal orthogonal representation. For TRAD we apply a constructive compaction method, that transforms the orthogonal representation into a turn-regular one and then runs a flow-based coordinate assignment in order to get a feasible planar grid drawing. Then we repeatedly apply horizontal and vertical compaction steps with a flow-based improvement heuristic until no further improvement can be achieved, see heuristic T2FL in Klau et al. (2001). For FF we use the same constructive heuristic and then apply the modified flow technique from Sect. 4 with a cost of one for the network arcs causing double bends to achieve best possible shortening, also repeatedly in horizontal and vertical direction. Recall that although both approaches are optimal for the one-dimensional compaction problems, repeated application of horizontal and vertical compaction steps does in general not lead to drawings with optimal total edge length for the two-

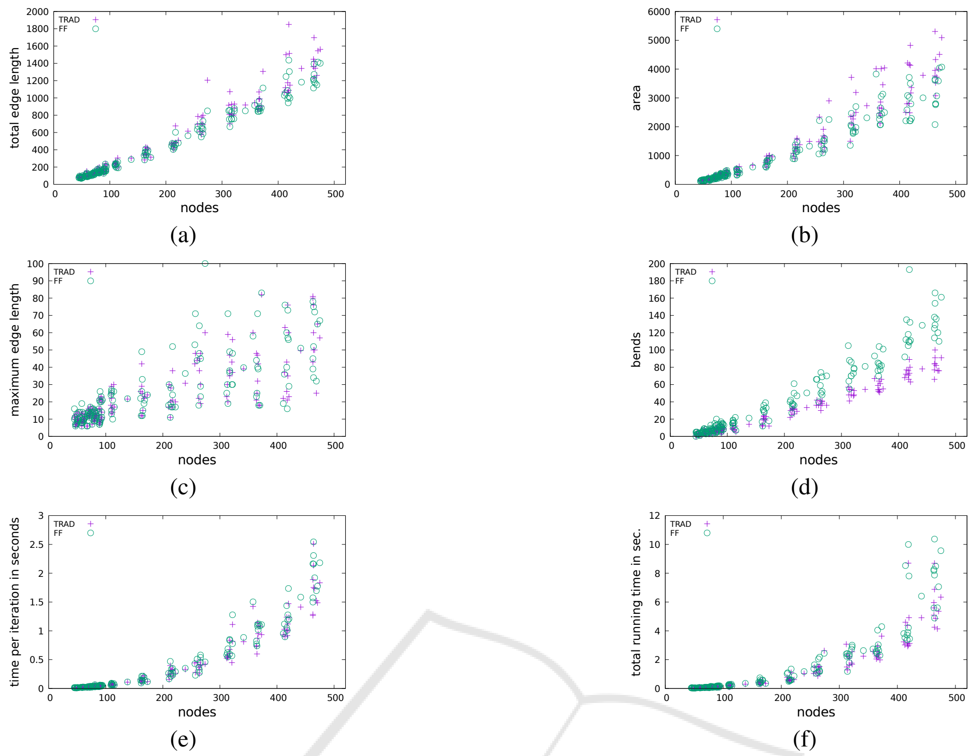


Figure 7: Absolute results of TRAD and FF for the quasi-trees: (a) total edge length, (b) area, (c) maximum edge length, (d) number of bends, (e) time per iteration, (f) total running time.

dimensional compaction problems. Notice also that due to the alternating repetitions in FF the monotonicity of edge segments may no longer be maintained.

We state the following hypotheses regarding the results of our new approach FF compared to those of TRAD:

- **(H1)** The total edge length and therefore the area of the drawings will decrease. We will also examine the change of the maximum edge length. It is hard to predict the behaviour, because on the one hand adding a double bend lengthens an edge, but it could also lead to shorter edges in other places.
- **(H2)** The number of bends will rise significantly.
- **(H3)** Although there will be many more dissecting edges, the running time will not increase drastically.

We tested our algorithm on three test suites. The standard benchmark set called *Rome graphs* introduced in Di Battista et al. (1997) consists of about 11,000 real-world and real-world like graphs with 10 to 100 vertices. The second set of graphs are *quasi-trees* which are known to be hard to compact. They have already been used in the compaction literature (e.g., Klau et al. (2001), Klau (2002)). From this, we selected a subset of 155 graphs with 40 to 450 vertices. The last set consists of 240 randomly generated biconnected planar 4-graphs with 40 to 500 vertices. All used graphs were, if necessary, initially turned into planar 4-graphs by planarizing them with methods of OGDF and replacing vertices with $k > 4$ outgoing edges with faces of size k. This results in graph sizes of up to 314 vertices for the Rome test suite and up to 475 vertices for the quasi-trees before the orthogonalization step. The input instances are available on https://ls11-www.cs.tu-dortmund.de/mutzel/compaction. All tests were run on an Intel Xeon E5-2640v3 2.6GHz CPU with 128 GB RAM.

(H1) Figure 6 shows the average decrease of the total and maximum edge length as well as of the area in percent. In all three graph classes the total edge length as well as the area has improved. We were able to decrease the total edge length by up to 36.1% and the area by up to 68.7%. In general, larger graphs allow a larger improvement. It turned out that for the majority of the instances we were also able to reduce the maximum edge length. Only for the quasi-trees the longest edges produced by FF are longer on average. But in general the results are mixed, reaching from a lengthening by 87.5% to a shortening by 58.1%. Figures 7,

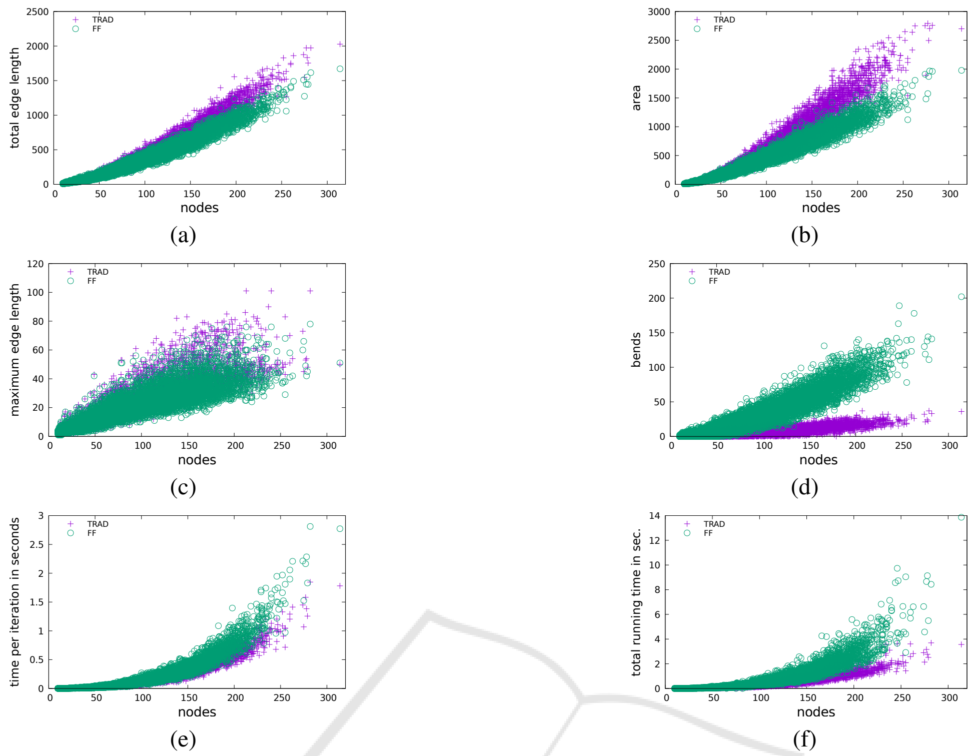


Figure 8: Absolute results of TRAD and FF for the Rome graphs: (a) total edge length, (b) area, (c) maximum edge length, (d) number of bends, (e) time per iteration, (f) total running time.

8 and 9 (a) - (c) show absolute values for the total edge length, the area and the maximum edge length measured for the three graph classes.

(H2) Figure 6 displays the average increase of the number of bends. If an instance had no bends after TRAD, we use the number of bends after FF as relative increase to take also these instances into account for computing the average increase of bends. As expected the drawings produced by FF have many more bends, especially for the Rome graphs. In one of the worst cases the number of bends went up from zero to 72. Although the relative increase of bends is very high, the average number of bends per edge in the drawings produced by FF is 0.18 for the Rome graphs, 0.13 for the quasi trees and 0.30 for the biconnected graph, respectively. The highest ratio over all instances is 0.52. Recall, that we started with bend minimal orthogonal representations. Figures 7, 8 and 9 (d) show the absolute values for the number of bends measured for the test graphs. Figure 10 gives the relation between the number of invested bends and the improvement in terms of area and total edge length for the biconnected graphs. The data for the other two graph classes can be found in Jünger et al. (2017). Figure 11 plots the relation between the number of bends and the number of edges.

As mentioned before, we cannot limit the number of bends added by FF, but we decrease the likelyhood to send flow over an arc of the form $a_v^{\uparrow}$ or $a_v^{\downarrow}$ by assigning higher costs to these arcs. We evaluated the number of additional bends, the area and total edge length for different arc costs on the subset of the Rome graphs with 100 to 150 vertices. These instances showed the greatest increase regarding the number of bends. Since we focus on the impact of the arc cost here, we apply only one compaction step in each direction. As expected the number of new bends as well as the achieved improvement in terms of area and total edge length compared to TRAD decreases with increasing arc costs, see Fig. 12. With a slightly higher arc cost we can reduce the number of additional bends without drastically affecting the area and total edge length improvement. The average relative increase of bends went down from 663.7% for an arc cost of one to 324.6% for an arc cost of two and 234.3% for an arc cost of three. The average improvement of area (total edge length) is 24.7% (13.8%), 22.5% (11.4%) and 20.5% (10.2%), respectively. The differences between higher arc costs in terms of additional bends become smaller. The data regarding the number of bends scatter over a wide range (notice the different scale on the right in Fig. 12 for the change of bends), especially for low arc cost, therefore it is

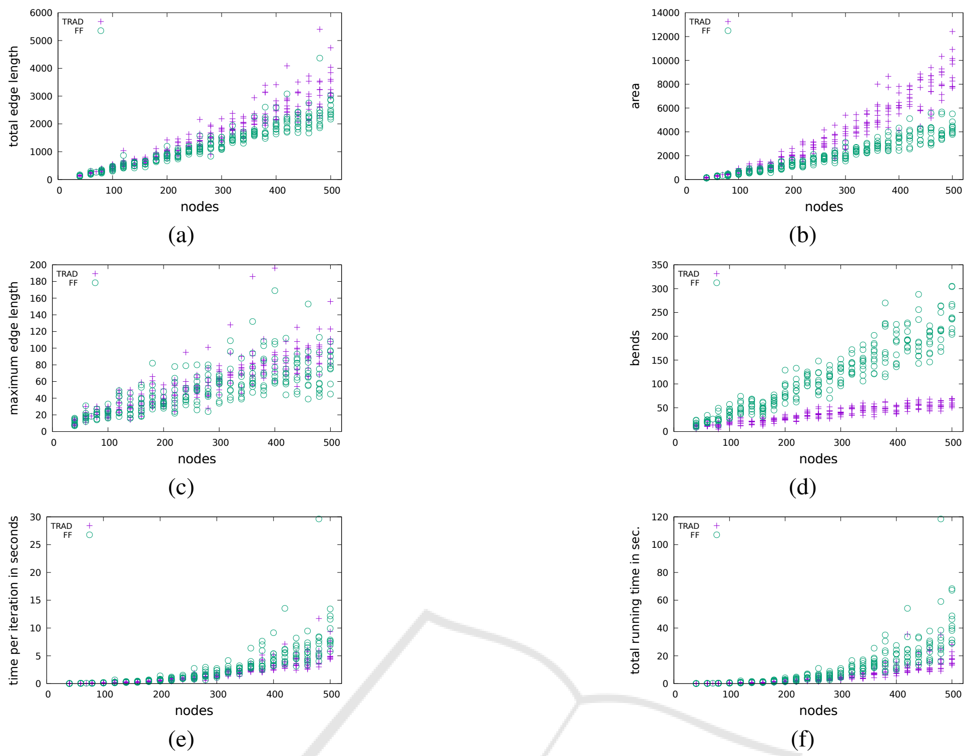


Figure 9: Absolute results of TRAD and FF for the biconnected graphs: (a) total edge length, (b) area, (c) maximum edge length, (d) number of bends, (e) time per iteration, (f) total running time.

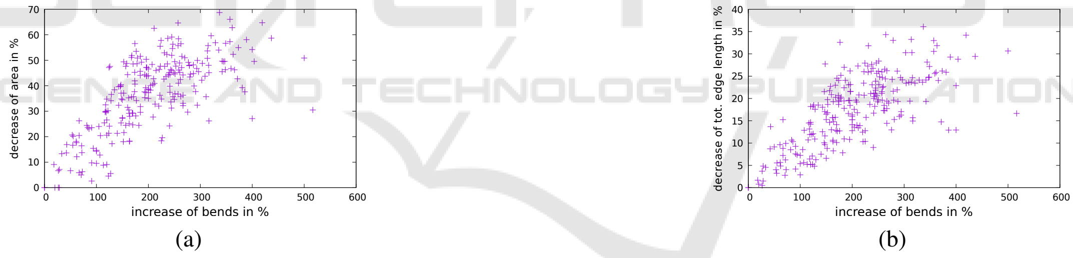


Figure 10: Relation between additional bends and (a) improvement of area and (b) total edge length for the biconnected graphs.

hard to determine which arc cost is suitable to keep the number of additional bends low, but still yields good geometric results.

(H3) We measured the total running time and the number of performed compaction iterations. For better comparison we also display the running time per iteration, because FF tends to do more iterations. The reason for that is that a compaction step of FF changes the drawing and therefore the flow network of the next step more significantly, leading to more new possibilities in compacting. The right side of Figure 6 displays the average increase of the running time and the number of visibility edges. In all cases the number of additional dissecting edges has increased significantly. The running time per iteration has also increased, but not more than 50% on average. Only for 252 instances of the Rome graphs and one biconnected graph the running time per iteration has more than doubled.

The results support the hypotheses. Our new approach is able to improve the drawing area and the total edge length by introducing additional bends. Figure 13 shows two examples drawn with TRAD and FF, respectively.

6 CONCLUSION

We have presented a new compaction heuristic for orthogonal graph drawing that alters the orthogonal rep-

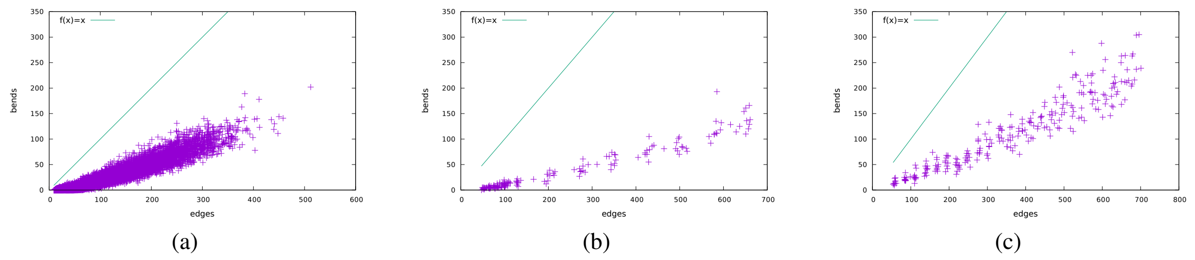


Figure 11: Number of bends in the drawings produced by FF for (a) the Rome graphs, (b) the quasi trees and (c) the biconnected graphs.

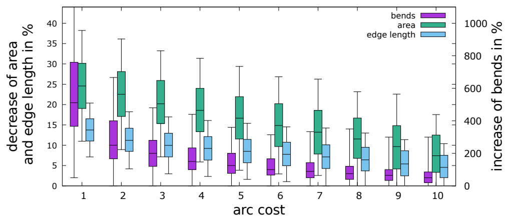


Figure 12: Relative change of the number of bends, area and total edge length with increasing arc costs for a subset of the Rome graphs. The whiskers cover 90% of the data, no outliers are depicted for better readability.

resentation. We were able to reduce the total edge length and drawing area, but at the expense of additional bends. Unfortunately, in our model we have no control over the number of bends added, but if the focus of interest lies on area and edge length, the results show a very positive effect. Future research could focus on studying the trade-off between bends and area or bends and total edge length as a bicriteria optimization problem.

ACKNOWLEDGEMENTS

This work was supported by the DFG under the project *Compact Graph Drawing with Port Constraints* (DFG MU 1129/10-1). We would like to thank Gunnar Klau for providing us with most of our test instances and anonymous reviewers for helpful remarks on a previous version of this paper. We also gratefully acknowledge helpful discussions with Martin Gronemann, Sven Mallach, and Daniel Schmidt.

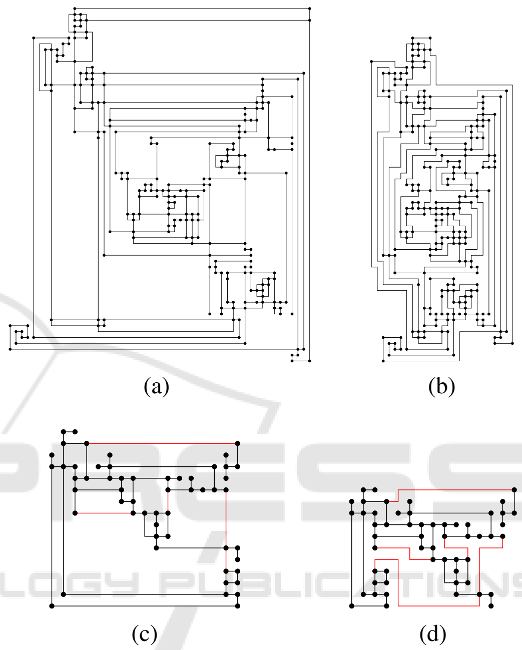


Figure 13: Biconnected graph with 220 vertices and 304 edges and Rome graph with 55 vertices and 66 edges (after planarization and expanding high degree vertices): (a) TRAD: total edge length 1459, area 3060, bends 31 (b) FF: total edge length 1107, area 1320, bends 133 (c) TRAD: total edge length 168, area 240, bends 1 (d) FF: total edge length 140, area 140, bends 13. Edges with additional bends are red.

REFERENCES

Ahuja, R. K., Magnanti, T. L., and Orlin, J. B. (1993). *Network Flows: Theory, Algorithms, and Applications*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA.

Bannister, M. J., Eppstein, D., and Simons, J. A. (2012). Inapproximability of orthogonal compaction. *J. Graph Algorithms Appl.*, 16(3):651–673.

Batini, C., Nardelli, E., and Tamassia, R. (1986). A layout algorithm for data flow diagrams. *IEEE Transactions on Software Engineering*, SE-12(4):538–546.

Bridgeman, S. S., Di Battista, G., Didimo, W., Liotta, G., Tamassia, R., and Vismara, L. (2000). Turn-regularity and optimal area drawings of orthogonal representations. *Comput. Geom.*, 16(1):53–93.

Chimani, M., Gutwenger, C., Jünger, M., Klau, G. W., Klein, K., and Mutzel, P. (2013). The Open Graph Drawing Framework (OGDF). In Tamassia, R., editor, *Handbook of Graph Drawing and Visualization*, chapter 17, pages 543–569. CRC Press.

Cornelsen, S. and Karrenbauer, A. (2012). Accelerated bend minimization. *J. Graph Algorithms Appl.*, 16(3):635–650.

Di Battista, G., Eades, P., Tamassia, R., and Tollis, I. G. (1999). *Graph Drawing: Algorithms for the Visualization of Graphs*. Prentice-Hall, Upper Saddle River, NJ, USA.

Di Battista, G., Garg, A., Liotta, G., Tamassia, R., Tassinari, E., and Vargiu, F. (1997). An experimental comparison of four graph drawing algorithms. *Comput. Geom.*, 7:303–325.

Fößmeier, U., Heß, C., and Kaufmann, M. (1998). On improving orthogonal drawings: The 4m-algorithm. In Whitesides, S., editor, *Graph Drawing, 6th International Symposium, GD'98, Proceedings*, volume 1547 of *LNCS*, pages 125–137. Springer.

Fößmeier, U. and Kaufmann, M. (1995). Drawing high degree graphs with low bend numbers. In Brandenburg, F., editor, *Graph Drawing, Symposium on Graph Drawing, GD '95, Proceedings*, volume 1027 of *LNCS*, pages 254–266. Springer.

Holzhauser, M., Krumke, S. O., and Thielen, C. (2016). Budget-constrained minimum cost flows. *J. Comb. Optim.*, 31(4):1720–1745.

Jünger, M., Klau, G. W., Mutzel, P., and Weiskircher, R. (2004). AGD - A library of algorithms for graph drawing. In *Graph Drawing Software*, pages 149–172.

Jünger, M., Mutzel, P., and Spisla, C. (2017). Orthogonal compaction using additional bends. *CoRR*, abs/1706.06514.

Kaufmann, M. and Wagner, D., editors (2001). *Drawing Graphs, Methods and Models*, volume 2025 of *LNCS*. Springer.

Klau, G. W. (2002). *A combinatorial approach to orthogonal placement problems*. PhD thesis, Saarland University, Saarbrücken, Germany.

Klau, G. W., Klein, K., and Mutzel, P. (2001). An experimental comparison of orthogonal compaction algorithms. In Marks, J., editor, *Graph Drawing, 8th International Symposium, GD 2000, Proceedings*, volume 1984 of *LNCS*, pages 37–51. Springer.

Klau, G. W. and Mutzel, P. (1999). Optimal compaction of orthogonal grid drawings. In Cornuéjols, G., Burkard, R. E., and Woeginger, G. J., editors, *Integer Programming and Combinatorial Optimization, 7th International IPCO Conference, 1999, Proceedings*, volume 1610 of *LNCS*, pages 304–319. Springer.

Lengauer, T. (1990). *Combinatorial Algorithms for Integrated Circuit Layout*. John Wiley & Sons, Inc., New York, NY, USA.

Patrignani, M. (1999). On the complexity of orthogonal compaction. In *Algorithms and Data Structures, 6th International Workshop, WADS '99, Proceedings*, volume 1663 of *LNCS*, pages 56–61. Springer.

Six, J. M., Kakoulis, K. G., and Tollis, I. G. (1998). Refinement of orthogonal graph drawings. In Whitesides, S., editor, *Graph Drawing, 6th International Symposium, GD'98, Proceedings*, volume 1547 of *LNCS*, pages 302–315. Springer.

Spönemann, M., Fuhrmann, H., von Hanxleden, R., and Mutzel, P. (2010). Port constraints in hierarchical layout of data flow diagrams. In Eppstein, D. and Gansner, E. R., editors, *Graph Drawing, 17th International Symposium, GD 2009. Revised Papers*, volume 5849 of *LNCS*, pages 135–146. Springer.

Tamassia, R. (1987). On embedding a graph in the grid with the minimum number of bends. *SIAM J. Comput.*, 16(3):421–444.

Tamassia, R., editor (2013). *Handbook on Graph Drawing and Visualization*. Chapman and Hall/CRC.