Fast Compaction for Orthogonal Drawings with Vertices of Prescribed Size*

Markus Eiglsperger and Michael Kaufmann

Universität Tübingen, Wilhelm-Schickard-Institut für Informatik
{eiglsper/mk}@informatik.uni-tuebingen.de

Abstract. In this paper, we present a new compaction algorithm which computes orthogonal drawings where the size of the vertices is given as input. This is a critical constraint for many practical applications like UML. The algorithm provides a drastic improvement on previous approaches. It has linear worst case running time and experiments show that it performs very well in practice.

1 Introduction

Orthogonal drawings of graphs are extensively used in many application areas. Examples are UML class-diagrams in software engineering or ER-diagrams in database management. One critical property in many of these diagrams is that the vertices of the graph have prescribed size. Take again UML class-diagrams, where the size of the vertices is determined by the contained text.

Traditional algorithms for orthogonal layout do not take prescribed vertex sizes into account. Recently, algorithms based on the topology-shape-metrics approach have been proposed for this problem [1], [11], and [5]. The topology-shape-metrics approach introduced in [3] is a very popular algorithm framework and consists of the three steps planarization, layout and compaction. The lengths in the drawing are assigned in the compaction step. Therefore the prescribed size constraints for vertices affect mainly this step. However, the input of the compaction step, also called the shape of the drawing, must ensure that the prescribed size constraints can be fulfilled. Algorithms based on the `Kandinsky` framework [7] guarantee this.

The algorithm of Di Battista et. al. [1] creates in a first step a drawing of the graph where the vertices are represented as points and edges overlap if they are adjacent to the same vertex at the same side. Then each vertical and each horizontal grid-line is expanded individually, i.e., each grid-line is replaced by a set of grid-lines such that the vertices can be assigned their prescribed sizes and the edges can be routed without overlap. This expansion is done by solving a Min-Cost-Flow problem for each horizontal and each vertical grid-line. The authors do not give any bounds on the time complexity of their algorithm. They experienced up to 50 seconds computation time on graphs in the Rome graphs test suite [4]. Klau et al. propose in [11] that their compaction approach

* Partially supported by DFG-Grant Ka812/8-1

P. Mutzel, M. Jünger, and S. Leipert (Eds.): GD 2001, LNCS 2265, pp. 124–138, 2002.

originating from graph labeling can also be used to solve the compaction problem for drawings with prescribed vertex size. The approach relies on branch-and-cut and has, therefore, exponential worst case running time. The authors give neither a detailed description of how the algorithm can be used in the prescribed vertex size setting nor experimental results.

We will present in this work a new compaction algorithm which constructs orthogonal drawings where vertices have prescribed size. The algorithm requires linear running time which improves significantly on the existing algorithms. The design of the algorithm allows us to also improve the quality of the result in the expense of running time. We also provide the results of empirical tests which demonstrate the effectiveness of the algorithm in practice. Since now, experiments for compaction algorithms have only been performed for 4-graphs [9].

This paper is organized as follows: In section 2, we introduce basic concepts of the topology-shape-metrics approach and the `Kandinsky` framework. In section 3, we present a new linear time compaction algorithm for 4-graphs. In section 4 we extend this algorithm for the `Kandinsky` model. In section 5, we show how prescribed vertex sizes can be integrated in this algorithm. Section 6 deals with extensions, implementation issues and concludes with the results of extensive empirical tests.

2 Preliminaries

We assume familiarity with the concept of planarity of graphs. An *embedded planar graph* is a planar graph with a specific circular order of edges around vertices and a specific external face, admitting a planar drawing that respects the given embedding. Unless otherwise specified, the planar graphs we consider are always embedded.

A *planar orthogonal drawing* of a planar graph is a planar drawing that maps each vertex to a point and each edge to a sequence of horizontal and vertical segments. In a *planar orthogonal grid drawing* the vertices and the bends along the edges have integer coordinates. Note that a planar graph admits a planar orthogonal grid drawing if and only if it is a 4-graph, i.e., the vertices of the graph have degree less than or equal to 4. An *orthogonal representation* H is a mapping from the set of faces F of a 4-graph G to clockwise ordered lists of tuples (e_r, a_r, b_r) where e_r is an edge, a_r is the angle formed with the following edge inside the appropriate face, stored as multiple of 90^o, and b_r is the list of bends of the edge. Note that in a planar orthogonal drawing, $1 \leq a_r \leq 4$ holds. If there are no bends in H, we call H *simple*.

A *planar orthogonal box drawing* of a planar graph is a planar drawing that maps each vertex to a box and each edge to a sequence of horizontal and vertical segments. In the corresponding grid drawing the center of the boxes and the bends along the edges have integer coordinates. A *quasi-orthogonal representation* Q is defined analogously to orthogonal representation with the difference that 0^o angles are allowed. A 0^o angle denotes that the following edge is adjacent to the same side of the vertex as the preceding edge. Note that quasi-orthogonal representations are not related to *quasi-orthogonal drawings* as described in [10].

Different drawing conventions have been proposed for planar orthogonal box drawings. We will concentrate on the so-called `Kandinsky`-models. All `Kandinsky`-models impose the following constraints on the drawing which we call `Kandinsky`-properties: the *bend-or-end property* and the *non-empty face property*. The bend-or-end property is defined as follows: Let G=(V, E) be an embedded graph and Γ be an planar orthogonal box drawing of G. Let e_1 and e_2 be two edges adjacent to the same side of a vertex v, e_1 following e_2 in the embedding. Let f be the face to which e_1 and e_2 are adjacent. Then either e_1 must have a first bend with a 270^o angle in f or e_2 must have a first bend with 270^o angle in f. The non-empty face condition forbids some degenerated cases for triangles in the graph. See [7] for a detailed description of the `Kandinsky`-properties.

There are several variations of `Kandinsky`-models: In the original version all vertices were represented by squares of equal size, arranged on a coarse vertex grid [7]. In the *big-node model* [8], the size of the vertices is determined by the number of edges attached to the different sides of the vertex. We also consider the *point model*, in which the vertices are represented by points. As pointed out above, these drawings might not be valid orthogonal drawings since edge overlap may occur. In [1] the *podavsnef*-model is introduced in which vertices have prescribed size. We refer to this model as the *prescribed-size* `Kandinsky`-model in this work.

We assume that the (quasi-)orthogonal representations, which stem from the second phase of the topology-shape-metrics algorithm have only a linear number of bends. The algorithm works also for cases where the representations have more bends, but the time bounds are no longer valid. The above assumption is justified, since to our knowledge, all algorithms which create (quasi-)orthogonal representations follow this assumption.

3 Compaction of 4-Graphs

In this section, we treat the *compaction problem for orthogonal drawings* and propose a linear time algorithm for it. Our algorithm is a combination of the exact compaction algorithm [12] and the rectangular decomposition technique [15]. Rectangular decomposition already leads directly to a linear time algorithm, but the insight that we gain in combining the two techniques is the base for the following sections. The problem is stated as follows:

Problem 1: Given an embedded 4-graph G with orthogonal representation H, find a drawing Γ of G with orthogonal representation H.

Note that every orthogonal representation can be reduced to a simple orthogonal representation by replacing the bends in H by dummy vertices. We therefore assume for the rest of the section that H is simple. We first review the exact compaction algorithm and then derive our algorithm from it.

3.1 The Approach of Klau/Mutzel

Given an embedded graph G with orthogonal representation H. We calculate from G and H a mapping $dir : E \rightarrow \{r, u\}$ and an orientation of G such that

there exists an orthogonal drawing Γ of G with orthogonal representation H where the directed edge e points upward if $dir(e) = u$ and points to the right if $dir(e) = r$. We can calculate this in linear time. Note that this assignment is not unique, but the set of solutions are rotations of each other. Note also that for each edge, there are two entries in H. In one of these entries, the edge orientation is the same as the traversal direction of the face and in one of these entries the traversal direction is the opposite of the edge orientation. From this fact we can derive in which absolute direction $d \in \{up, down, left, right\}$ an edge is traversed in a face-traversal and append this information directly to the tuple in H. We assume for the rest of the section that G is directed according to the orientation stated above.

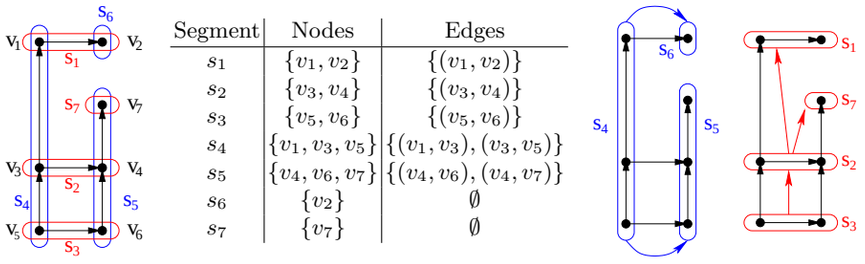
We denote with $G_r = (V, E_r)$ the subgraph of G which contains only horizontal edges, i.e., $E_r = \{e \in E : dir(e) = r\}$ and with $G_u = (V, E_u)$ the subgraph of G which contains only vertical edges, i. e., $E_u = \{e \in E : dir(e) = u\}$.

The connected components of G_r, resp. G_u, are directed paths and form a line in a drawing of G with orthogonal representation H. We denote with S_r, resp. S_u, the set of connected components of G_r, resp. G_u, and call the elements of it *horizontal segments*, resp. *vertical segments*. We say that two segments are *adjacent* if they share a point. We denote with $\alpha(s)$ the start node of the path which forms a segment s and with $\omega(s)$, the endpoint of this path. Every edge e is contained in exactly one segment $seg(e)$ and each node v is contained in exactly one vertical segment $vert(v)$ and one horizontal segment $hor(v)$.

The concept of *constraint graph* describes the ordering of the segments of one type. The edge sets of the constraint graphs $D_u = \{S_r, A_u\}$, resp. $D_r = \{S_u, A_r\}$, are defined as:

$$A_u = \{(hor(v), hor(w)) : (v, w) \in E_u\}, \text{ and}$$
$$A_r = \{(vert(v), vert(w)) : (v, w) \in E_r\}$$

The *shape description* $\mathcal{S} = (D_r, D_u)$ combines two constraint graphs.



Segment	Nodes	Edges
s_1	$\{v_1, v_2\}$	$\{(v_1, v_2)\}$
s_2	$\{v_3, v_4\}$	$\{(v_3, v_4)\}$
s_3	$\{v_5, v_6\}$	$\{(v_5, v_6)\}$
s_4	$\{v_1, v_3, v_5\}$	$\{(v_1, v_3), (v_3, v_5)\}$
s_5	$\{v_4, v_6, v_7\}$	$\{(v_4, v_6), (v_4, v_7)\}$
s_6	$\{v_2\}$	$\emptyset$
s_7	$\{v_7\}$	$\emptyset$

Fig. 1. Examples of segments and the corresponding constraint graph.

Definition 1. *Let s_r be a vertical segment and s_u be a horizontal segment which are not adjacent. We call s_r and s_u to be* separated *if one of the following conditions hold:*

$$1.\ s_u \xrightarrow{*}_{D_r} ver(\alpha(s_r)) \qquad 2.\ ver(\omega(s_r)) \xrightarrow{*}_{D_r} s_u$$
$$3.\ s_r \xrightarrow{*}_{D_u} hor(\alpha(s_u)) \qquad 4.\ hor(\omega(s_u)) \xrightarrow{*}_{D_u} s_r$$

A shape description is called *complete* if every pair of segments with opposite direction is separated.

We define the linear program (LP) as:

$$\begin{aligned} x_b - x_a &\geq 1 \ \ \forall (a,b) \in A_r \\ y_b - y_a &\geq 1 \ \ \forall (a,b) \in A_u \\ x_s &\geq 0 \ \ \forall s \in S_u \\ y_s &\geq 0 \ \ \forall s \in S_r \end{aligned}$$

Lemma 1. *[12] If $\mathcal{S}$ is complete a feasible solution (x, y) of the linear program (LP) induces an orthogonal drawing $\Gamma : V \to \mathbb{N} \times \mathbb{N}$ with $\Gamma(v) = (x_{vert(v)}, y_{hor(v)})$ of G with orthogonal representation H.*

We can find a feasible solution of (LP) in linear time by solving a longest path problem [13]. When the shape description is not complete, there may be feasible solutions of the (LP) which induce non-valid orthogonal drawings. In [12], it is shown that there always exists a superset of $\mathcal{S}$ which is a complete shape description. We call this superset $\mathcal{S}_{ext}$ a *complete extension* of $\mathcal{S}$. Klau/Mutzel [12] propose a branch and cut algorithm which finds the complete extension such that the resulting drawing has minimal edge length. This approach has the disadvantage that it has exponential worst-case running time. We cannot hope to do better when we try to optimize the edge length, since this problem is NP-hard[14]. If we drop the goal to achieve optimality, we can get a much faster algorithm by searching a complete shape extension by heuristics.

3.2 A Fast Heuristic

We propose the following strategy to solve the compaction problem:

1. Calculate orientation and *dir* of G and H.
2. Calculate shape description $\mathcal{S} = (D_r, D_u)$.
3. Calculate a complete extension of $\mathcal{S}$.
4. Solve corresponding longest path problem on D_r and D_u.
5. Assign coordinates according to longest distances.

It remains to be shown how we can find a complete extension. The heuristics we use is based on the technique of rectangular decomposition[15]. The starting point for the rectangular decomposition strategy is the observation that if all faces of the graphs are rectangles, we can easily solve the compaction problem by applying longest path or network flow algorithms to it. The idea is to subdivide those faces which are not rectangular into rectangles and then solve the problem on this subdivision. This induces a valid embedding on the original graph. What remains is to perform this subdivision efficiently, which can be done by searching certain patterns of angles on the face. We denote 90^o angles on a face with a

$'0'$ and 270^o angles with a $'1'$. Every time we find the pattern 100, we cut a rectangle from the face and continue to search the pattern on the remaining face. See Fig. 2 for an illustration. We terminate if there are no more patterns in any face. Using a list, rectangle decomposition can be done in linear time.

Function `init-list`(*Face* f)

List $l \leftarrow \epsilon$;
for *each* $(e = (v, w), a, \epsilon, d)$ *in* f **do**
  // Let d' be the direction obtained by rotating d by 90^o.;
  if $a = 1$ **then** append $(0, seg(e), d)$ to l;
  if $a = 3$ **then** append $(1, seg(e), d)$ to l;
  if $a = 4$ and $e \in E_u$ **then** append $(1, seg(e), d), (1, hor(w), d')$ to l;
  if $a = 4$ and $e \in E_r$ **then** append $(1, seg(e), d), (1, ver(w), d')$ to l;
end
return l

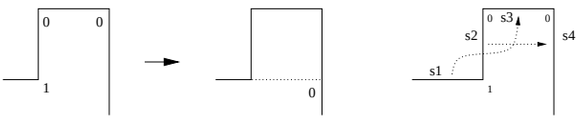


Fig. 2. Decomposition of a face into a rectangle and a remaining face.

From this algorithm we can directly derive a completion heuristic. Assume that we are in the situation illustrated in Fig. 2. Instead of introducing a dummy node and a dummy edge in the graph, we simply add edges to the constraint graph. In the case above, we insert the edge (s_1, s_3) in D_u and the edge (s_2, s_4) in D_r. We handle the other three cases symmetrically. The function `define-box` describes the four cases:

Function `define-box`(*Shape description* $\mathcal{S}$, *Direction* d, *Segments* s_1, s_2, s_3, s_4)

Let $\mathcal{S} = ((S_u, A_r)(S_r, A_u))$;
If $d = up$ **then** $A_u \leftarrow A_u \cup (s_2, s_4)$, $A_r \leftarrow A_r \cup (s_3, s_1)$;
If $d = down$ **then** $A_u \leftarrow A_u \cup (s_4, s_2)$,$A_r \leftarrow A_r \cup (s_1, s_3)$;
If $d = left$ **then** $A_u \leftarrow A_u \cup (s_1, s_3)$,$A_r \leftarrow A_r \cup (s_2, s_4)$;
If $d = right$ **then** $A_u \leftarrow A_u \cup (s_3, s_1)$,$A_r \leftarrow A_r \cup (s_4, s_2)$;

The completion algorithm executes on every face first `init-list` and then `decompose`.

Lemma 2. *With the above algorithm, we can calculate a complete shape extension of size* $O(n)$ *in linear time.*

Function `decompose`(*Shape description* , *List* l)

```
while size(l) > 4 do
    // denote with t_i = (a_i, s_i, d_i) the i-th tuple in l;
    if (a_1 = 1) and (a_2 = 0) and (a_3 = 0) then
        define-box(S, d_1, s_1, s_2, s_3, s_4);
        replace t_1 with (0, s_1, d_1);
        remove t_2 and t_3 from l;
    else
        move t_1 to the rear of l
    end
end
define-box(S, d_1, s_1, s_2, s_3, s_4);
```

Proof. We first show that the complete shape extension has size $O(n)$. The initial shape description has linear size by Euler's formula. Since the rectangle decomposition introduces $O(n)$ rectangles and we insert two edges into it per rectangle, the complete shape extension has linear size. The linear running time follows immediately from this fact, too. It remains to be shown that the extension is complete. Take a drawing of G produced by the conventional rectangle decomposition algorithm and take a vertical segment s_r and a horizontal segment s_u which are not adjacent. Because the drawing is planar, s_r and s_u do not cross, so one of the four following cases must hold: s_r is above s_u, s_r is below s_u, s_r is left of s_u or s_r is right of s_u. Assume w.l.o.g. that s_u is above s_r and that s_u is not to the left of s_r. The other cases are symmetric. Since s_u is above s_r, $s' = hor(\alpha(s_u))$ is also above s_r. Assume that one of the following cases holds:

1. There is a $s_v \in S_u$ such that the intersection of the projection of s_v and s_u on the y-axis is non-empty and there is a path from $vert(\omega(s_r))$ to s_u in D_r.
2. There is a segment $s_h \in S_r$ such that the intersection of the projection of s_h and s' on the x-axis is non-empty and there is a path from s_r to s_h in D_u.

If the assumption is true, we are done. To see this, assume that the second case holds. Then just take a line parallel to the y-axis with x-coordinate in the intersection of the projections. From the rectangles which are intersected by the parallel line, we can now easily construct a path from s_h to s' in D_u.

We give a constructive proof that either s_v or s_h exists. Start at segment s_r and go to the lowest rectangle to the right of it, if such a rectangle exists. In this case, go from this rectangle to the leftmost rectangle above. Iterate until a segment is found which induces an intersection of the projections. Because we proceed monotonically increasing in x- and y-coordinates, such a segment must exist for monotonicity reasons. The existence of the path follows from how we traverse the rectangles.

Theorem 1. *The above algorithm solves the Problem 1 in linear time.*

4 Compaction in the Kandinsky Point Model

Before we present the compaction algorithm for prescribed vertex sizes in the next section, we take a step in between and consider the compaction problem for the `Kandinsky` point model, which is presented in this section.

Problem 2: Given an embedded graph G with simple quasi-orthogonal representation Q having the `Kandinsky` properties. Find a drawing Γ of G with quasi-orthogonal representation Q, in which the vertices of G are represented as points.

We assume as in the previous section that G is oriented, marked with direction tags and that bends are replaced by dummy vertices.

In a quasi-orthogonal representation there may be more than one edge adjacent to a side of a vertex. As a consequence a segment does not represent a directed path of edges with the same direction tags as in the previous section. See Fig.3 for such a segment.

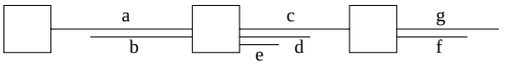


Fig. 3. A segment in a quasi-orthogonal representation, where vertices are denoted as rectangles.

We therefore refine the segment definition and introduce the concept of *sub-segment*. We call two edges (u, v) and (v, w) a *right-join* (*left-join*) if they have the same direction and between them in the cyclic order (reverse cyclic order) there are only edges with different directions. In Fig. 3, edges b and e form a right-join and edges a and c a left-join.

Definition 2. *Let p be a directed path consisting of edges with the same direction tag. The path p is a* sub-segment *if every two consecutive edges in the path are a right-join or every two consecutive edges in the path are a left-join or if p consists of one single edge. The path p is a* maximal sub-segment *if there is no sub-segment containing it.*

Because we will only consider maximal sub-segments, we will omit the word 'maximal' for the rest of this work. Note that the terms sub-segment and maximal-sub-segment have a different meaning as in [12]. The sub-segments of the segment shown in Fig. 3 are (a, c, g),(b, e),(d) and (c, f). An edge is part of at most two sub-segments. As for segments, we distinguish between vertical sub-segments and horizontal sub-segments. A vertex may be adjacent to an arbitrary number of sub-segments. If s is a sub-segment, we denote with $seg(s)$ the segment which contains s. Note that α and ω are no longer well defined on segments, but are well-defined for sub-segments. We define $\mathcal{S} = (D_r, D_u)$ and (LP) analogous to the previous section.

Definition 3. *Let s_r be a vertical sub-segment and s_u be a horizontal sub-segment which are not adjacent. We call s_r and s_u* separated *if one of the following conditions hold:*

$$\textit{1. } seg(s_u) \xrightarrow{*}_{D_r} seg(\alpha(s_r)) \qquad \textit{2. } seg(\omega(s_r)) \xrightarrow{*}_{D_r} seg(s_u)$$
$$\textit{3. } seg(s_r) \xrightarrow{*}_{D_u} seg(\alpha(s_u)) \qquad \textit{4. } seg(\omega(s_u)) \xrightarrow{*}_{D_u} seg(s_r)$$

We call a shape description *complete* if every pair of sub-segments with opposite direction is separated unless the overlap of these two segments cannot be avoided because of the representation of vertices as points.

With the same argumentation as in Lemma 1, it follows that if $\mathcal{S}$ is complete, a feasible solution (x, y) of (LP) induces an orthogonal drawing $\Gamma : V \to \mathbb{N} \times \mathbb{N}$ with $\Gamma(v) = (x_{vert(v)}, y_{hor(v)})$ of G with quasi-orthogonal representation Q. As in the previous section a feasible solution of (LP) can be found with a longest path algorithm in linear time. It remains to be shown how we can calculate a complete shape extension for $\mathcal{S}$. We again use rectangle decomposition to perform this, but this time we need two rounds.

Let f be a face in Q. In the first round, we eliminate all 0^o angles from f. We modify for this purpose the `init-list` function described in the previous section in two points: we denote a 0^o angle with a '-1' in the cyclic-list and we store the sub-segments rather than the segments in the list entries. Then, we perform a modified version of the `decompose` function on this list. The first modification is that we search for patterns described in Fig. 4. The choice of rule (c) or (d) depends on some technical constraints, see [6] for a detailed description. The `define-box` function takes sub-segments instead of segments as arguments. It connects the segments in the constraint graph which correspond to the sub-segments unless it would connect a segment with itself. We terminate when we have traversed the entire cyclic list. Let l' denote the resulting cyclic list. In the

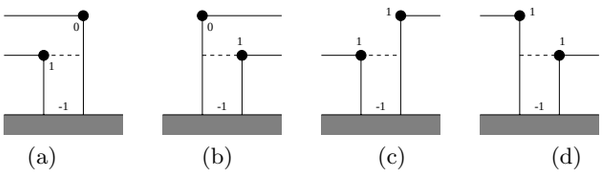


Fig. 4. Rules to eliminate zero degree angles

second round, we simply perform `decompose` on l'.

From the correctness and time bounds proven in the previous section, together with the fact that the traditional version of rectangle decomposition described in [6] is correct, the following theorem follows:

Theorem 2. *The above algorithm solves Problem 2 in linear time.*

5 Compaction in the Prescribed-Size Kandinsky Model

In this section we provide a compaction algorithm for the prescribed-size `Kandinsky`-model. We denote with $width(v)$ the prescribed width of a vertex in G and with $height(v)$ the prescribed height. We assume that the number of edges adjacent to the top/bottom side of a vertex v is less than $width(v) + 1$ and the number of edges adjacent to the left/right side is less than $height(v) + 1$.

Problem 3: Given an embedded graph G with quasi-orthogonal representation Q having the `Kandinsky` properties. Find a drawing Γ of G with orthogonal representation Q, in which each vertex v of G is represented by a box of the size $(width(v), height(v))$.

We assume as in the previous section that G is oriented, marked with direction tags and that bends are replaced by dummy vertices.

The algorithm proceeds as follows: It first replaces each non-dummy vertices by a rectangular face. The result of this transformation is a 4-graph with a simple orthogonal representation. It creates for this orthogonal representation a complete shape description which is compatible with the vertex size constraints. Finally, it calculates the drawing from the complete extension. We now give a detailed description of the algorithm.

5.1 Simplification

Let $B \subset V$ denote the dummy vertices representing bends. The elements in B have zero size. In a first step, the vertices with non-zero size are replaced by rectangular faces. Let v be such a vertex. For each edge adjacent to v, a new node $p(e, v)$ is created which represents the port of e on v. Also, four corner nodes $nw(v)$, $ne(v)$, $sw(v)$ and $se(v)$ are created. See Fig. 5 for an example. Each node face has four adjacent *node-segments*: the top segment $t(v)$, the bottom segment $b(v)$, the left segment $l(v)$, and the right segment $r(v)$. The result of this

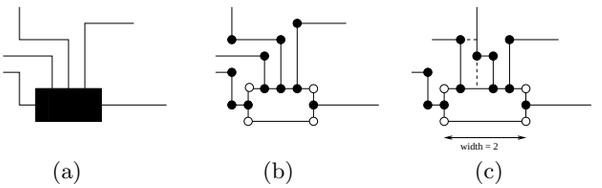


Fig. 5. Transformation of a vertex into a face and an invalid compaction.

simplification step is a 4-graph $G_S = (V_S, E_S)$ with orthogonal representation H_S. The mapping $simple : E \to E_S$ maps every edge in E to the corresponding edge in E_S. G_S has $4|V| + 2|E| + |B|$ nodes. Since G and G_S are planar it follows with Eulers formula that the above transformation causes a constant blow up.

5.2 Problem Description

We now try to find a drawing of G_S which induces a valid drawing on G. We call these drawings *valid drawings* of G_S. There are two differences between a valid drawing of G_S and a valid drawing of a 4-graph described in Sec. 3:

1. The edges adjacent to a corner node may have zero length.
2. The segments of the node-faces have prescribed distance.

We extend the algorithm of the previous section to match these requirements. This is done by refining the constraint graph definition of the previous section by adding a length function to the constraint graphs, and introducing auxiliary edges which denote the vertex size. These refinings are similar to the ones in [11]. We define the edge-set of the constraint graph D'_u as $A'_u = A_u \cup N_u^+ \cup N_u^-$, with

$$N_u^+ = \{(b(v), t(v)) : v \in V\}, \text{ and } N_u^- = \{(t(v), b(v)) : v \in V\}$$

The constraint graph D'_r is defined analogously, and $\mathcal{S}' = (D'_r, D'_u)$ is the corresponding shape description. Additionally, we define the length function length : $A'_u \cup A'_r \to Z$ in the following way:

$$\text{length}(\mathbf{e}) = \begin{cases} 0 & \text{if } e \in A_u, \text{e is induced by an edge adj. to a corner} \\ \text{height}(v) & \text{if } e \in N_u^+ \\ -\text{height}(v) & \text{if } e \in N_u^- \\ 1 & \text{otherwise} \end{cases}$$

The values of length for A'_r are defined analogously. This leads to the `Kandinsky` linear program (KLP):

$$\begin{aligned} x_b - x_a &\geq \text{length}(e) \ \ \forall e = (a,b) \in A'_u \\ y_b - y_a &\geq \text{length}(e) \ \ \forall e = (a,b) \in A'_r \\ x_s &\geq 0 \ \ \forall s \in S_u \\ y_s &\geq 0 \ \ \forall s \in S_u \end{aligned}$$

As in the previous sections, we need a characterization of the shape descriptions whose solution of (KLP) induce valid drawings. But, this time, it is not enough to demand that all segments have to be separated, since there are cases where all segments are separated but there is no feasible solution for (KLP). The reason for this is that we might violate the maximum length condition for node faces, see Fig. 5(c) for an example.

Definition 4. *A shape description $\mathcal{S}'$ is* length-complete *if $\mathcal{S}'$ is complete and every cycle in the constraint graphs of $\mathcal{S}'$ has non-positive length.*

Lemma 3. *Let $\mathcal{S}'$ be a complete shape description. (KLP) has a feasible solution if and only if every cycle in the constraint graphs of $\mathcal{S}'$ has non-positive length.*

Proof. It is shown in [12] that if a constraint graph in $\mathcal{S}$ has a positive length cycle, then (KLP) is not feasible. We can transform (KLP) to a *system of difference constraints* by multiplying each constraint by -1. This transforms every positive length cycle in a negative length cycle and vice versa. It holds now that a system of difference constraints is feasible if and only if it has no negative length cycle, see, i.e., [2] for a proof.

We cannot apply the longest path algorithms of the previous sections to find a feasible solution of (KLP) because there are cycles and negative edges in the constraint graphs. In the remainder of this section, we will present an algorithm which finds a feasible solution of (KLP) in linear time.

A segment $s \in Q$ induces a set of segments $simple(s)$ in H_S. Note that every negative length edge in a constraint graph connects two segments in H_S which belong to the same segment in Q. Our algorithm considers one segment of Q at a time. Let s be a vertical segment in Q. A critical role is played by the edges in s which are not adjacent to bends. We denote these edges by $nb(s)$ and the corresponding edges in H_S by $nb_S(s)$. Because of the bend-or-end property the edges in $nb(s)$ form a path in s. Let $e = (v, w) \in nb(s)$, $e_S = simple(e)$. We denote with $d(v, e)$ the minimal distance from $left(v)$ to e in D_r. The same holds for w and $d(w, e)$. This value can be calculated in amortized constant time from the shape. Given an x-value $\bar{x}$ for one arbitrary edge $e_S \in nb_S(s)$, where $e = (v, w)$ is the corresponding edge in Q. We define $x(left(v)) = x(e_S) - d(e_S, v)$ and $x(left(w)) = x(e_S) - d(e_S, w)$. Let e'_S be the following edge of e in the path $nb(s)$, and e''_S the preceding edge. We define $x(e'_S) = x(left(v)) + d(e'_S, v)$ and $x(e''_S) = x(left(w)) + d(e'_S, w)$. We proceed recursively until x is determined for all left sides of nodes and all edges in nb_S. Given $e \in nb_S(s)$ and $\bar{x} = 0$ and the rest of the x-values calculated as described above. Then we let $\text{off}(e, s) = \min_{e' \in nb_S(s)}\{x(e')\}$.

The algorithm to calculate x works now as follows: First, we determine the vertical segments of Q and calculate a topological numbering $s_1, \ldots, s_k$ of them. We denote with $U_i \subseteq S'_u, 0 \leq i < k$ the set of vertical segments in H_S whose corresponding segment in Q is s_i. We then calculate a topological numbering on the vertical segments of H_S where the U_i appear in an interval. For every i, we do the following: We perform the standard DAG algorithm for longest paths on U_i. Then we search the edge $e \in nb_S(s_i)$ with maximal x-value. We add $\text{off}(e, s_i)$ to this x-value and apply the above algorithm to determine the x-values of all other segments in s_i. This algorithm has linear running time.

Lemma 4. *A feasible solution of (KLP) can be found in time $O(n)$.*

5.3 Rectangular Decomposition

We will again use the rectangular decomposition method to obtain a length-complete shape extension.

Definition 5. *Let $v \in V \setminus B$ and $d \in \{up, down, left, right\}$. Then $seg(v, d)$ is defined the following way:*

$$seg(v,d) = \begin{cases} left(v) & \text{if } d = up \\ right(v) & \text{if } d = down \\ top(v) & \text{if } d = right \\ bottom(v) & \text{if } d = left \end{cases}$$

The equivalent to a sub-segment in Q is a *meta-segment* in H_S.

Definition 6. *Let $s = e_0, e1, \ldots, e_r$ be a sub-segment, with $e_0 = (v_0, v_1)$, $e_2 = (v_1, v_2), \ldots e_r = (v_r, v_{r+1})$ and $d \in \{up, down, left, right\}$. The corresponding meta-segment $meta(s, d)$ is defined as the set:*

$$\{seg(v_0, d), simple(e_0), seg(v_1, d), simple(e_1), \ldots, simple(e_r), seg(v_{r+1}, d)\}$$

On every face f we perform rectangle decomposition in the following way: We first execute the first round of the decomposition algorithm of section 4 on f and obtain a list of three-tuples l. Then we insert for each tuple $t = (a, s, d)$ one *meta-segment node* $mn(s)$ in the corresponding constraint graph which represents the boundaries of the meta-segment $meta(s, d)$ at f. If the face is on the right side of $mn(s)$ we create for every segment s' in $meta(s, d)$ an edge $(mn(s), s')$, if the face is on the left side we create edges $(s', mn(s))$. The created edges have zero length. Then we separate adjacent meta-segments by introducing edges of length 1 between one meta-segment node and one segment node. This is illustrated in Fig. 6(b). As a last step we apply function `decompose` on l, where we use a special version of function `define-box` which connects the appropriate meta-segment nodes in the constraint graphs as illustrated in Fig. 6(a).

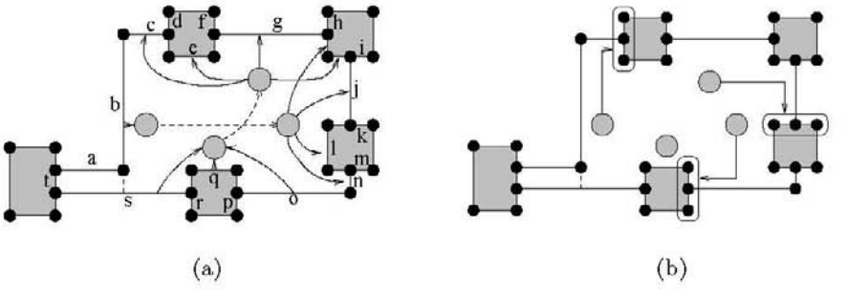


Fig. 6. A face with meta-segments $(a), (b), (c, e, g, i), (h, j, l, m), (o, q, s), (t)$.

Lemma 5. *The algorithm described above makes the shape-description length-complete. The shape description has linear size.*

Proof. (Sketch) We introduce at most $2|E|$ meta-segment nodes. Every segment in H_S is contained in at most two meta-segments and is therefore connected

to a meta-segment node at most twice. From this it follows that H_S has linear size. All segments which take part in a decomposition step are separated. With a similar argument as in Lemma 2 then follows that all non-adjacent segments in Q are separated.

Theorem 3. *The above algorithm solves Problem 3 in linear time.*

6 Practical Issues and Experiments

We assumed in the previous section that vertices have sufficient width resp. height to connect all edges adjacent to a certain side. But this assumption may be not satisfied by the layout algorithm which provides the input for the compaction. In this case, we define in the simplification step the distances between certain edges adjacent to the same side of a vertex as zero such that the size constraint can be fulfilled. With a slight modification the non-bending edge on a vertex side can be placed in the middle of the side which yields nice drawings.

We performed no empirical comparison with [1], but we are convinced that the quality of the drawings with respect to the drawing area is as least as good as in [1]. The edge length may be worse, but to further improve the quality of the drawing, we can solve the occurring linear programs with a Min-Cost Flow algorithm instead of using longest path algorithms. Of course this increases the running time. The solution obtained by using the Min-Cost Flow algorithm has minimal edge length with respect to the given complete shape extension (see, e.g., [12]). To improve an existing drawing, we can apply one-dimensional compaction, known from VLSI-design (see, e.g., [13]), as postprocessing. We can reuse for this case the compaction algorithm; we only have to replace the rectangle decomposition step by a separation function based on the visibility of segments in the input drawing.

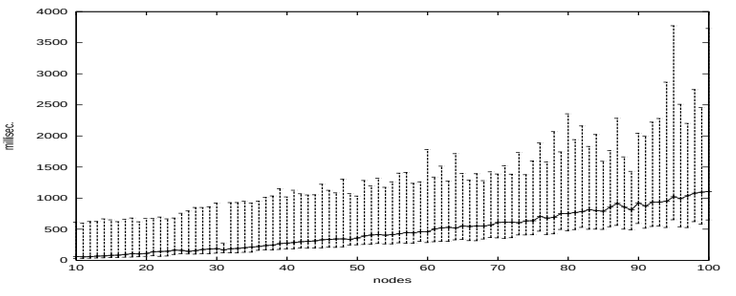


Fig. 7. Average running time of the algorithm on the Rome graphs.

We tested the algorithm on the so called Rome graphs test suite [4]. We assigned to each vertex a random width and height between zero and ten. The

input of the compaction algorithm was generated by an implementation of the `Kandinsky` bend-minimizing algorithm [7]. The running time of the algorithm was always below 4 seconds using the virtual machine provided in the JDK 1.2.2 of Sun Microsystems, with a 64MB memory limit on a Sun Ultra 5 with 333 MHz. The average running time of the algorithm is shown in Fig. 7.

References

1. G. Di Battista, W. Didimo, Maurizo Patrignani, and Maurizio Pizzonia. Orthogonal and quasi-upward drawings with vertices of prescribed size. In J. Kratochvil, editor, *Proceedings of the 7th International Symposium on Graph Drawing (GD'99)*, volume 1731 of *LNCS*, pages 297–310. Springer, 1999.
2. T.H. Cormen, C.E. Leiserson, and R.L. Rivest. *Introduction to Algorithms*. McGraw-Hill, 1990.
3. G. Di Battista, P. Eades, R. Tamassia, and I. G. Tollis. *Graph Drawing: Algorithms for the Visualization of Graphs*. Prentice Hall, 1999.
4. G. Di Battista, A. Garg, , G .Liotta, R. Tamassia, E. Tassinari, and F. Vargiu. An experimental comparison of four graph drawing algortihms. *Comput. Geom. Theory Appl.*, 7:303–325, 1997.
5. M. Eiglsperger. Constraints im Kandinsky-Algorithmus. Master's thesis, Universität Tübingen, 1999.
6. U. Fößmeier. *Orthogonale Visualisierungstechniken für Graphen*. PhD thesis, Eberhard-Karls-Universität zu Tübingen, 1997.
7. U. Fößmeier and M. Kaufmann. Drawing high degree graphs with low bend numbers. In F. J. Brandenburg, editor, *Proceedings of the 3rd International Symposium on Graph Drawing (GD'95)*, volume 1027 of *LNCS*, pages 254–266. Springer, 1996.
8. U. Fößmeier and M. Kaufmann. Algorithms and area bounds for nonplanar orthogonal drawings. In G. Di Battista, editor, *Proceedings of the 5th International Symposium on Graph Drawing (GD'97)*, volume 1353 of *LNCS*, pages 134–145. Springer, 1997.
9. K. Klein G. W. Klau and P. Mutzel. An experimental comparison of orthogonal compaction algorithms. In *Proceedings of the 8th International Symposium on Graph Drawing (GD'2000)*, number 1984 in LNCS, pages 37–51, 2001.
10. G. W. Klau and P. Mutzel. Quasi-orthogonal drawing of planar graphs. Technical Report 98-1-013, Max-Planck-Institut für Informatik, Saarbrücken, 1998.
11. G. W. Klau and P. Mutzel. Combining graph labeling and compaction. In J. Kratochvil, editor, *Proceedings of the 7th International Symposium on Graph Drawing (GD'99)*, number 1731 in LNCS, pages 27–37. Springer, 1999.
12. G. W. Klau and P. Mutzel. Optimal compaction of orthogonal grid drawings. In *Integer Programming and Combinatorial Optimization (IPCO'99)*, number 1610 in LNCS, pages 304–319, 1999.
13. T. Lengauer. *Combinatorial Algorithms for Integrated Circuit Layout*. Applicable Theory in Computer Science. Wiley-Teubner, 1990.
14. M. Patrignani. On the complexity of orthogonal compaction. *Computational Geometry: Theory and Applications*, 19(1):47–67, 2001.
15. R. Tamassia. On embedding a graph in the grid with the minimum number of bends. *SIAM Journal on Computing*, 16(3):421–444, 1987.