Orthogonal and Quasi-Upward Drawings with Vertices of Arbitrary Size*

Giuseppe Di Battista, Walter Didimo, Maurizio Patrignani, and Maurizio Pizzonia
Dipartimento di Informatica e Automazione,
Università di Roma Tre, via della Vasca Navale 79,
00146 Roma, Italy.
{gdb,didimo,patrigna,pizzonia}@dia.uniroma3.it

Abstract

We consider the problem of computing orthogonal drawings and quasi-upward drawings with vertices of arbitrary size. For both types of drawings we present algorithms based on network flow techniques and show that the produced drawings are optimal within a wide class. Further, we present the results of an experimentation conducted on the algorithms that we propose for orthogonal drawings. The experiments show the effectiveness of the approach.

1 Introduction

Orthogonal drawings are extensively used in many application areas and several algorithms for constructing orthogonal drawings have been presented in the literature. A few examples can be found in [13, 4, 7, 8, 14, 17, 12, 18]. However, among such a large set of algorithms and methods, the most popular strategy is probably the so called topology-shape-metrics approach [9].

The topology-shape-metrics approach, originally proposed in [3, 2, 19, 20], has found in the last ten years several variations, implementations, and improvements. Also, it has been the subject of deep theoretical and experimental analysis. However, although such approach has a quite good behaviour in many applications, it still has some drawbacks and unsolved problems. One of the most important of those problems is the difficulty in producing drawings whose vertices have size assigned by the user. Actually, the algorithms based on the topology-shape-metrics approach make either the assumption that the vertices are points of the grid (see, e.g. [19]), or the assumption that all vertices have the same size (see, e.g. [2, 13]).

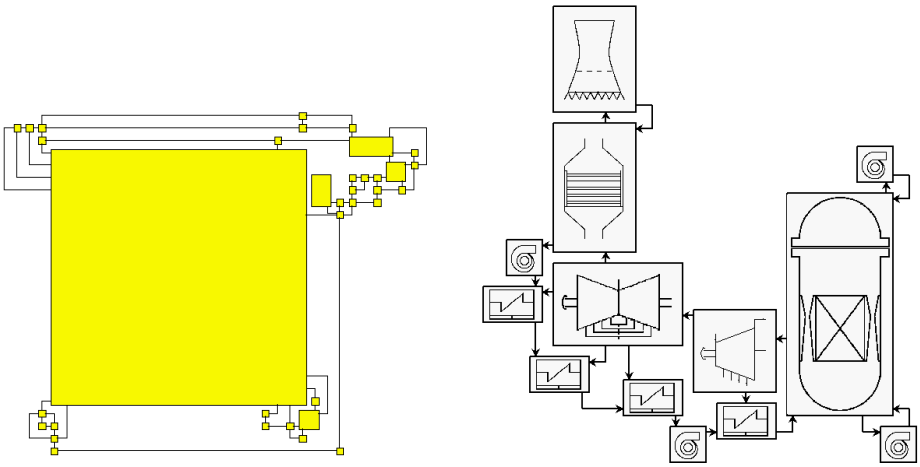
However, the demand of the applications for drawing algorithms that allow to produce orthogonal drawings where the user can control the size of each specific vertex is quite strong. Examples are diagrams with long labels on symbols, diagrams for which the semantics or the importance of each vertex is related to its size, and diagrams whose vertices contain pictures or maybe other diagrams. Further, the capability of drawing vertices of arbitrary size could be a key issue in the realization of new techniques for edges labelling. Namely, fictitious vertices of size suitable to host the labels could be inserted during a preprocessing step and replaced with the corresponding label in the final drawing.

We consider the problem of producing orthogonal drawings with vertices of arbitrary size within the topology-shape-metrics approach. See in Fig. 1 two examples of drawings computed with the techniques described in this paper.

The main contributions of this paper can be summarized as follows.

- We introduce (Section 2) a new drawing convention, called *podavsnef*, for planar orthogonal drawings with vertices of arbitrary size. It is a variation of the *podevsnef* drawing convention presented in [13].
- We present an algorithm for constructing orthogonal drawings with vertices of arbitrary size within the *podavsnef* drawing convention (Section 3). The algorithm first computes a *podevsnef* drawing

*Research supported in part by the CNR Project "Geometria Computazionale Robusta con Applicazioni alla Grafica ed al CAD", and by the ESPRIT LTR Project 20244 (ALCOM-IT)



(a) (b)

Figure 1: (a) A drawing with small and large vertices. (b) The basic blocks of an industrial plant (a boiling water reactor nuclear plant). Both drawings have been computed with an implementation of the techniques described in this paper.

and then expands the vertices. The expansion of vertices is shown to be optimal among a large set of possible expansions. The produced drawings can be further compacted with a post-processing compaction technique at the expenses of a higher computation time. Both the algorithm for vertex expansion and the post-processing are based on network flow techniques.

- We implemented and experimented the algorithms for orthogonal drawings presented in Section 3 on the test suite of graphs used in [10]. The height and the width of the vertices of the graphs have been randomly chosen in a wide interval. The experiments put in evidence the effectiveness of the techniques (Section 4).
- Since the topology-shape-metrics approach has been recently extended to the construction of quasi-upward planar drawings of directed graphs [6], we also study the problem of producing quasi-upward planar drawings of digraphs with vertices of arbitrary size (Section 5). We show an algorithm that is based on the construction of a visibility representation as an intermediate drawing.

2 Preliminaries and Drawing Conventions

We assume familiarity with planarity and connectivity of graphs [11, 16] and with flow networks [1]. We also assume familiarity with graph drawing [9].

An *embedded planar graph* is a planar graph with a specified circular order of edges around vertices and a specified external face, admitting a planar drawing that respects the given embedding. Unless otherwise specified the planar graphs we consider are always embedded. A planar *st-digraph* G is an embedded planar digraph with only one source s and one sink t embedded on the external face. The *dual* st-digraph G^* of G is defined as follows. The faces of G are in one-to-one correspondence with the vertices of G^*, but for the external face of G that corresponds to two vertices s^* and t^* of G^*. For every edge e of G, G^* has an edge (f, g), where f is the face to the left of e and g is the face to the right of e. Digraph G^* may have multiple edges.

A *planar orthogonal drawing* of a planar graph is a planar drawing that maps each edge into a chain of horizontal and vertical segments. A *planar orthogonal grid drawing* is an orthogonal drawing such that vertices and bends along the edges have integer coordinates. A planar graph admits a planar orthogonal (grid) drawing if and only if its vertices have maximum degree four [21]. An *orthogonal representation* (also called *shape*) is an equivalence class of planar orthogonal drawings such that all the drawings of the class have the same sequence of left and right turns (bends) along the edges and two edges incident at a common vertex determine the same angle.

In order to orthogonally draw graphs of arbitrary vertex degree, different drawing conventions have been introduced. The *podevsnef* (planar orthogonal drawing with equal vertex size and not empty faces) drawing convention was introduced in [13]. In a *podevsnef* drawing (see Fig. 2.a):

1. Vertices are points of the grid but it is easier to think to them in terms of squares of half unit sides centered at grid points.
2. Two segments that are incident on the same vertex may overlap. Observe that the angle between such segments has zero degree.
3. All the polygons representing the faces have area strictly greater than zero.
4. If two segments overlap they are presented to the user as two very near segments.

In [13] an algorithm is presented that computes a *podevsnef* drawing of a planar graph with the minimum number of bends. Further, the authors conjecture that the drawing problem becomes NP-hard when Condition 3 is omitted. The *podevsnef* drawings generalize the concept of orthogonal representation, allowing angles between two edges incident to the same vertex to have a zero degree value. The consequence of the assumption that the polygons representing the faces have area strictly greater than zero is that the angles have specific constraints. Namely, because of Conditions 2 and 3, each zero degrees angle is in correspondence to exactly one bend [13]. An orthogonal representation corresponding to the above definition is a *podevsnef orthogonal representation*.

We also consider a similar drawing convention introduced in [5] called *simple-podevsnef*. In that model, the following constraints are added to the *podevsnef* drawing convention:

1. In order to distribute the edges around a vertex more uniformly, each vertex with degree greater than four has at least one incident edge on each side.
2. Consider a vertex u with $\deg(u) > 4$. Consider two edges (u, v) and (u, w) incident on u and such that (u, w) follows (u, v) in the circular clockwise ordering given by the embedding. If there is a zero degree angle at u between (u, v) and (u, w), then (i) (u, w) contains at least one bend and (ii) the first bend of (u, w) encountered while following (u, w) from u to w causes a right turn.

The *simple-podevsnef* drawing convention has several advantages [5]: the computation of its orthogonal representation can be done with a standard minimum cost flow technique and the final orthogonal drawing can be computed with a standard compaction technique. Observe that a *simple-podevsnef* drawing is a *podevsnef* drawing.

We generalize the concept of *podevsnef* drawing by introducing the *podavsnef* convention. A *podavsnef* (planar orthogonal drawing with assigned vertex size and non-empty faces) drawing is an orthogonal drawing such that (see Fig. 1):

1. Each vertex is a box with its specific height and width (assigned to each single vertex by the user).
2. Two segments that are incident on the same vertex may overlap. Observe that the angle between such segments has zero degree.
3. Consider any side of length $l \geq 0$ of a vertex v and consider the set I of arcs that are incident on such side.
   (a) If $l + 1 > |I|$ then the edges of I cannot overlap.
   (b) Else ($l + 1 \leq |I|$). The edges of I are partioned into $l + 1$ non-empty subsets such that all the edges of the same subset overlap.
4. The orthogonal representation constructed from a *podavsnef* drawing by contracting each vertex into a single point is a *podevsnef* orthogonal representation.

From the above definition it follows that a *podevsnef* drawing is a *podavsnef* drawing such that all its vertices have width and height both equal to zero. On the other hand a *podavsnef* drawing is essentially a *podevsnef* drawing where vertices have specific sizes and where the edges incident on each vertex side are uniformly distributed.

A *quasi-upward drawing* of a digraph [6] is such that the horizontal line through each vertex v (that is drawn as a point) "locally" splits the incoming edges of v from the outgoing edges of v. The term locally is used to identify a sufficiently small connected region properly containing v.

A *pqudavs* (planar quasi-upward drawing with assigned vertex size) drawing (Fig. 10.e) is a quasi-upward drawing such that each vertex is a box with its specific height and width (assigned to each single vertex by the user).

3 Computing *Podavsnef* Drawings

In this section we first show that, given a planar graph and an assignment of height and width for each of its vertices it is always possible to compute a *podavsnef* drawing with the prescribed dimensions for vertices. Second, we show a complete strategy for constructing *podavsnef* drawings that allows trade-offs between effectiveness and efficiency.

Given a *podavsnef* drawing Γ and two vertical (horizontal) lines that do not intersect any vertex, a *vertical strip* (*horizontal strip*) is the set of the vertices of Γ contained in the geometric strip defined by the two lines. A *vertical partition* (*horizontal partition*) of Γ is the partition of the vertices of Γ into vertical (horizontal) strips with the maximum number of strips. We number the strips of the partition left to right (top to bottom).

An *podavsnef* drawing Γ is *splittable* if a vertical and a horizontal partition of the vertices of Γ exist such that: (1) A vertex v of Γ univocally determines one horizontal strip $\sigma_H(i)$ and one vertical strip $\sigma_V(j)$ (we associate to v the pair i, j); and (2) The function associating a pair i, j to each vertex v is injective. The two partitions are a *split* of Γ. Observe that if Γ is splittable, then its split is unique.

Consider two *podavsnef* drawings Γ' and Γ'' of the same graph and with the same *podevsnef* orthogonal representation and with splits σ' and σ'', respectively. Splits σ' and σ'' are *equivalent* if for each vertex v, the pair i, j determined by v in σ' is the same v determines in σ''.

Given a planar graph G we construct a *podevsnef* drawing Γ of G, using one of the algorithms presented in [13, 5]. Roughly speaking, our strategy consists of expanding the vertices inside different strips independently, preserving the shape of the drawing. After the expansion the strips are "glued" together.

Consider a *podevsnef* drawing Γ. Observe that Γ is always splittable. The split of Γ can be constructed as follows. The i-th horizontal (vertical) strip $\sigma_H(i)$ ($\sigma_V(i)$) is obtained considering the vertices of Γ with y-coordinates (x-coordinates) in the interval $[i - 1/2, i + 1/2]$. See Fig. 2.a.

We associate to a vertical strip $\sigma_V(i)$ a flow network $\mathcal{N}_i$ as follows (See Fig. 2.b).

In $\sigma_V(i)$ a vertex u is *up-visible* from a vertex v if $y(u) > y(v)$ and a vertex x does not exist such that $y(u) > y(x) > y(v)$. If u is up-visible from v and u and v are joined by a straight edge, then v is *up-adjacent* to u.

- The nodes of $\mathcal{N}_i$ are: Three nodes n^l_v, n^c_v, and n^r_v, for each vertex v of $\sigma_V(i)$. Two nodes $n^l_{u,v}$ and $n^r_{u,v}$ for each vertex u that is up-adjacent to a vertex v. One node $n^c_{u,v}$ for each vertex v that is up-visible from a vertex u that is not up-adjacent to v. One *source-node* s_i and one *sink-node* t_i.
- For each vertex u that is up-adjacent to a vertex v we introduce the arcs $(n^c_u, n^r_{u,v})$, $(n^c_u, n^l_{u,v})$, $(n^r_{u,v}, n^c_v)$, $(n^l_{u,v}, n^c_v)$, $(n^l_u, n^l_{u,v})$, $(n^r_u, n^r_{u,v})$, $(n^l_{u,v}, n^l_v)$, and $(n^r_{u,v}, n^r_v)$.
- For each vertex v that is up-visible from a vertex u that is not up-adjacent to v we introduce the arcs $(n^c_u, n^c_{u,v})$, $(n^c_{u,v}, n^c_v)$, $(n^l_u, n^c_{u,v})$, $(n^r_u, n^c_{u,v})$, $(n^c_{u,v}, n^l_v)$, and $(n^c_{u,v}, n^r_r)$.
- Let u be the bottommost (topmost) vertex of $\sigma_V(i)$. We introduce the arcs (s_i, n^l_u), (s_i, n^c_u), and (s_i, n^r_u) $((n^l_u, t_i)$, (n^c_u, t_i), and $(n^r_u, t_i))$.

The units of flow correspond to units of width of the strip. We denote by $lb(\cdot)$ and $ub(\cdot)$ lower and upper bounds, respectively. Each node n^c_u has $lb(n^c_u) = ub(n^c_u) = w \geq 0$, where w is the width assigned to u. The lower bounds of arcs $(n^l_{u,v}, n^c_v)$ and $(n^r_{u,v}, n^c_v)$ are used to preserve enough space for the edges incident on v from below. Namely, consider the sets I^l and I^r of arcs incident from below to the left and to the right of the straight edge connecting u to v, respectively. If $w + 1 > |I^l| + |I^r|$ then, according to the *podavsnef* convention, the incident edges can not overlap. Hence, we assign $lb(n^l_{u,v}, n^c_v) = |I^l|$ and $lb(n^r_{u,v}, n^c_v) = |I^r|$. Else $(w + 1 \leq |I^l| + |I^r|)$, w is not enough large to draw the incident edges all non-overlapping. In this case we assign $lb(n^l_{u,v}, n^c_v) = 0$, $ub(n^l_{u,v}, n^c_v) = |I^l|$, $lb(n^r_{u,v}, n^c_v) = 0$, and $ub(n^r_{u,v}, n^c_v) = |I^r|$. Analogous bounds are assigned to $(n^c_u, n^l_{u,v})$ and $(n^c_u, n^r_{u,v})$. Lower and upper bounds not specified are set to 0 and to ∞, respectively. All arcs have an ∞ capacity and 0 cost. See Fig. 2.

Property 1 *Network $\mathcal{N}_i$ is a planar st-digraph.*

We define a network $\mathcal{N}_V$ associated with the vertical partition of Γ. It is obtained by the networks $\mathcal{N}_i$ as follows. A new source s_V and a new sink t_V are defined. For each i the edges (s_V, s_i) and (t_i, t_V) are added with $lb(s_V, s_i) = lb(t_i, t_V) = 0$ and $ub(s_V, s_i) = ub(t_i, t_V) = \infty$.

We apply an analogous construction to the horizontal partition of Γ, defining a network $\mathcal{N}_H$.

Property 2 *Networks $\mathcal{N}_V$ and $\mathcal{N}_H$ are planar st-digraphs with $O(n)$ nodes, where n is the number of vertices of G.*

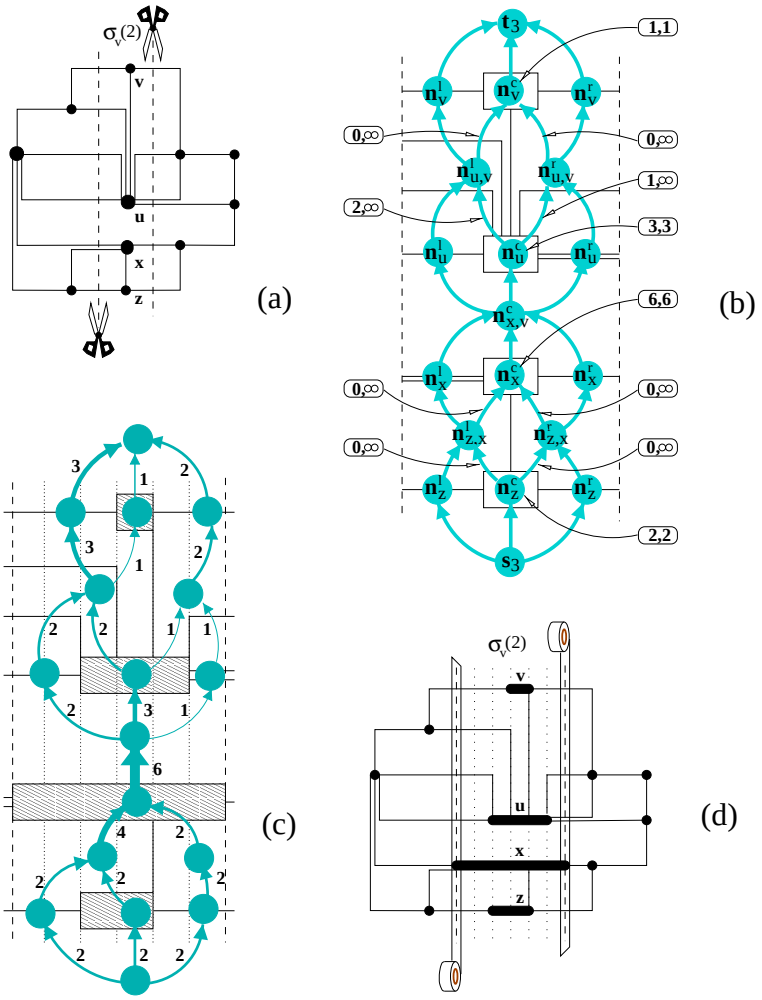


Figure 2: (a) A *podevsnef* drawing and its vertical strip $\sigma_V(2)$. (b) The flow network $\mathcal{N}_2$. Labels show the most important lower and upper bounds. (c) A minimum flow in the network $\mathcal{N}_2$. The thickness of the arcs is proportional to their flow; arcs and nodes with 0 flow are omitted. (d) Widths and positions of vertices in $\sigma_V(2)$.

From the above discussion we have:

Lemma 1 *Each pair of feasible flows one of $\mathcal{N}_V$ and the other of $\mathcal{N}_H$ determines a* podavsnef *drawing that admits a split equivalent to the one of Γ. Conversely, for each* podavsnef *drawing admitting a split equivalent to the one of Γ there exists a pair of feasible flows one of $\mathcal{N}_V$ and the other of $\mathcal{N}_H$.*

It is easy to see that networks $\mathcal{N}_V$ and $\mathcal{N}_H$ always have a feasible flow.

Because of Lemma 1 that and since a *podevsnef* drawing of a planar graph always exists [13], we have:

Lemma 2 *Let G be a planar graph and suppose an assignment of widths and heights is given for the vertices of G. A* podavsnef *drawing of G always exists with vertices of the given sides.*

The values of flow in $\mathcal{N}_V$ and $\mathcal{N}_H$ are in one-to-one correspondence with the width and the height of the corresponding *podavsnef* drawings. Hence a minimum width and height drawing can be computed by computing minimum flows on $\mathcal{N}_V$ and $\mathcal{N}_H$. Hence, we have:

Theorem 1 *Let G be a planar graph and let Γ be a* podevsnef *drawing of G. Suppose an assignment of widths and heights is given for the vertices of G. A* podavsnef *drawing of G can be computed that has minimum width and minimum height among those admitting a split that is equivalent to the one of Γ.*

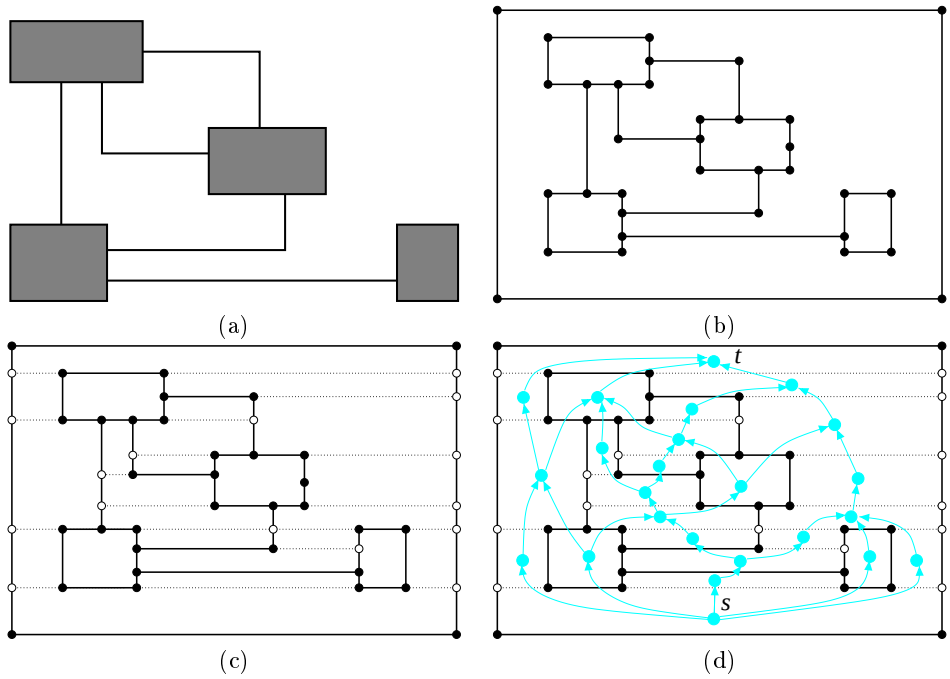


Figure 3: (a) The starting *podavsnef* drawing Γ. (b) The result Γ' of the preprocessing step. (c) The drawing $\Gamma^{\#}$. (d) The flow network $\mathcal{N}_V$.

Observe that a minimum flow in $\mathcal{N}_V$ can be computed by solving a min-cost-flow problem on $\mathcal{N}_V$ augmented with arc (s_V, t_V) (with cost equal to zero) and by setting a cost equal to one on arcs (s_V, s_i). In such network we can "pump" any feasible flow of $\mathcal{N}_V$. The minimum flow in $\mathcal{N}_V$ is obtained by subtracting from the feasible flow the value of flow through (s_V, t_V).

It is clear that, even if the drawings produced with the above strategy are minimal in their equivalence class, they can be often further reduced in size if we allow vertices of a strips (vertical or horizontal) to enter another strip. This is not possible in the above framework.

If the user is interested in smaller drawings, even at the expenses of a higher computational time, a more effective technique can be adopted. The idea is to use the *podavsnef* drawing produced so far as a starting point for successive compaction steps.

We adopt a heuristic similar to the one used in the VLSI compaction [15], and that generalizes the "moving" technique presented in [12]. Namely, at each step we "squeeze" as much as possible the drawing in one direction. Then, we do the same in the opposite direction. We continue alternating the two steps until the drawing cannot be further squeezed. The steps should not modify the size of vertices. In the following we describe the algorithm for compacting with respect to the horizontal direction. The vertical case is analogous.

First (preprocessing step), we reduce the problem of horizontally compacting the *podavsnef* drawing Γ to the problem of horizontally compacting a drawing Γ' with all zero size vertices and constraints on the length of a subset of its edges. It is worth noting that the techniques described hereunder can be used in any case an analogous reduction can be found. The drawing Γ' is obtained from Γ as follows (see Fig. 3.a and Fig. 3.b).

- Each box of Γ representing a vertex is replaced with a box-shaped cycle in which the corners of the box and the attach points of the edges are represented with zero size vertices. We call *vertex-boundary edges* the edges created in this way.
- Each bend is replaced by a zero size vertex.
- The whole drawing is enclosed in a box consisting of four additional vertices and edges, in order to make easy to handle the external face in the subsequent steps.

From Γ' we construct a drawing $\Gamma^{\#}$ such that all the faces are rectangles. This is done by splitting

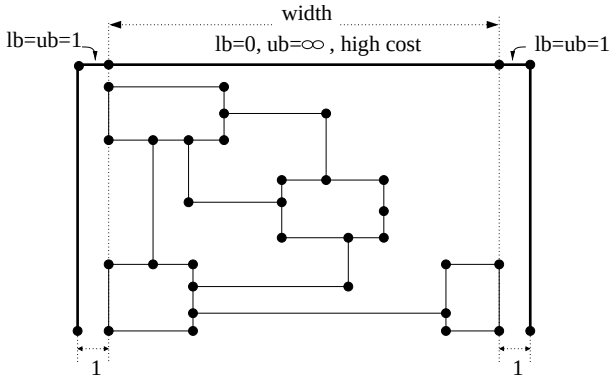


Figure 4: The pinch is a technique that allows to control the width of a drawing. Setting the cost of edge e allows to consider in the objective function the width of the drawing.

the faces of Γ' with a suitable set of new horizontal edges. See for example Fig. 3.c. This operation requires, in general, the introduction of fictitious vertices (white colored in Fig. 3.c) to split some vertical edges.

We compute the compacted drawing with a flow technique. Namely, we determine the new lengths to be assigned to the horizontal edges by solving a min-cost-flow problem on a network $\mathcal{N}$ built as follows (see Fig. 3.d).

- The nodes are the internal faces of $\Gamma^{\#}$.
- We insert an arc for each pair of vertically-adjacent faces, with the only exception of the adjacencies with the external face. In other words, the horizontal edges of $\Gamma^{\#}$ that are not on the external face are in one-to-one correspondence with the arcs. Each arc is directed from the bottom face to the upper face.

Each unit of flow corresponds to one unit of length. When the solution is computed the vertex coordinates can be easily assigned using a depth first search visit of the drawing. In order to preserve the size of vertices and to minimize the total length of the horizontal edges, we set bounds and costs on each arc a of $\mathcal{N}$ corresponding to edge e of $\Gamma^{\#}$ as follows.

- If e is a vertex-boundary edge, then we set $lb(a) = ub(a)$ equal to the value of the length of e. Since the value of the flow through a is fixed, its cost is irrelevant.
- If e belongs to $\Gamma^{\#}$ but it is not an edge of Γ', then $lb(a) = 1$, $ub(a) = \infty$, and the cost of a is set to 0.
- If e belongs to Γ' and it is not a vertex-boundary edge, then $lb(a) = 1$, $ub(a) = \infty$, and the cost of a is set to 1.

Property 3 *Network $\mathcal{N}$ is a planar st-digraph with $O(n + b)$ nodes, where n is the number of vertices and b is the number of bends of Γ. The source s and the sink t of $\mathcal{N}$ are the bottom and the top faces of $\Gamma^{\#}$, respectively.*

Network $\mathcal{N}$ admits a feasible flow. The value of flow produced by s is equal to the width of the box enclosing the drawing. It is easy to see that the solution of a min-cost-flow problem on $\mathcal{N}$ corresponds to a minimization of the total length of the horizontal edges.

It is also possible to exploit network $\mathcal{N}$ to minimize the total width of the drawing. This is done by using what we call the *pinch* technique. During the construction of Γ', before building the enclosing box we add to Γ' the structure depicted in Fig. 4. In such a construction the length of e measures the width of the drawing. Then, in $\mathcal{N}$, we set the cost associated with such edge to infinity.

It is worth noting that, even though the techniques described above considerably reduce the length of the edges, we still have space for improvements. In fact, the described procedure does not allow the attach points of the edges to "slide" along the boundaries of the vertices because the length of each vertex-boundary edge in Γ' is fixed. Such constraint can be relaxed.

In order to obtain the desired dimension for the vertices we can fix only the sum of the lengths of the vertex-boundary edges along the top and bottom sides of each vertex, leaving the attach points free to move along the boundary. Namely, the flow network $\mathcal{N}$ can be modified as follows (see Fig. 5.d):

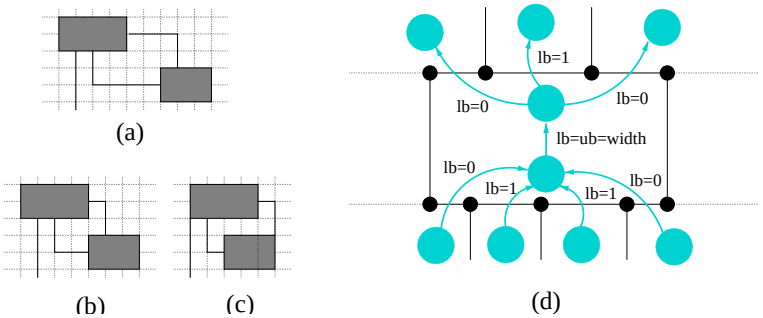


Figure 5: Given an initial configuration (a), figures (b) and (c) show the results of the compaction without and with the refined strategy illustrated in figure (d).

- For each box-shaped face f of $\Gamma^{\#}$ representing a vertex v of Γ, we split the node of $\mathcal{N}$ associated with f into two nodes v_{out} and v_{in} and add a directed arc $e = (v_{in}, v_{out})$. The incoming edges of v become the incoming edges of v_{in} while the outgoing edges of v become the outgoing edges of v_{out}.
- We set $lb(e) = ub(e)$ equal to the width assigned to v, and cost of e equal to 0.
- For each vertex-boundary edge e' of f, we set the lower bound of the corresponding arc of $\mathcal{N}$ equal to 0 if e' is incident to a "corner" of f, and equal to 1 otherwise. The value of $ub(e')$ is set to ∞ and the cost of e' is set to 0.

The benefits deriving from the above modifications are put in evidence in Fig. 5.a, 5.b, and 5.c.

4 Experiments with GDToolkit

We implemented the algorithms described in Section 3, that compute a *podavsnef* drawing from a *podevsnef* one. In particular our implementation constructs a *podavsnef* drawing starting from a *simple-podevsnef* drawing. The implementation uses the *GDToolkit* library[1].

We tested the algorithm over a set of more than 8000 (not-necessarily planar) graphs in the data-base used in [10]. Such graphs represent data from real-world applications. The number of vertices of the tested graphs is in the range $10 - 100$.

Our experimental setting is as follows. For each graph G in the test-suite:

- We have randomly chosen the width w and the height h of every vertex v of G with a uniform probability distribution in the range $0 - 9$. The area covered by v (in terms of grid-points) is $(w + 1)(h + 1)$. Fig. 7.a shows how the sum of the areas of the vertices increases with the number of vertices of G.
- We have computed a *simple-podevsnef* drawing Γ of G, and a *podavsnef* drawing Γ' of G starting from Γ. In the *simple-podevsnef* Γ all vertices have zero width and height, each one covering one unit of area. Both for Γ and for Γ' we measured the ratio between the total area covered by the vertices and the area of the drawing (both in terms of grid-points). Fig. 7.b shows the curves of these two values. The curve for *podavsnef* drawings indicates a good usage of the area of the drawing for representing vertices.

The experiments show that the compaction strategies described in the previous section make an effective usage of the area of the drawing, increasing the percentage of area used to represent vertices. Also, from the point of view of the CPU time, all the computations were performed within an acceptable amount of time. Namely, the largest graphs required less that 50 seconds on a PC Pentium II (350 MHz) with Linux and C++ code compiled with GNU g++.

In Fig. 8 a *podavsnef* drawing is shown of a 40-vertices graph of the test-suite, computed by our implementation.

[1] `http://www.dia.uniroma3.it/~gdt`

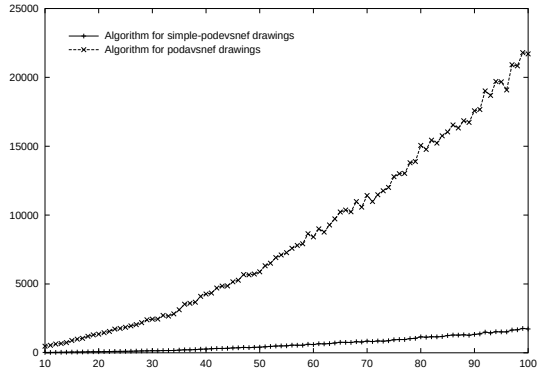


Figure 6: Area of the drawings.

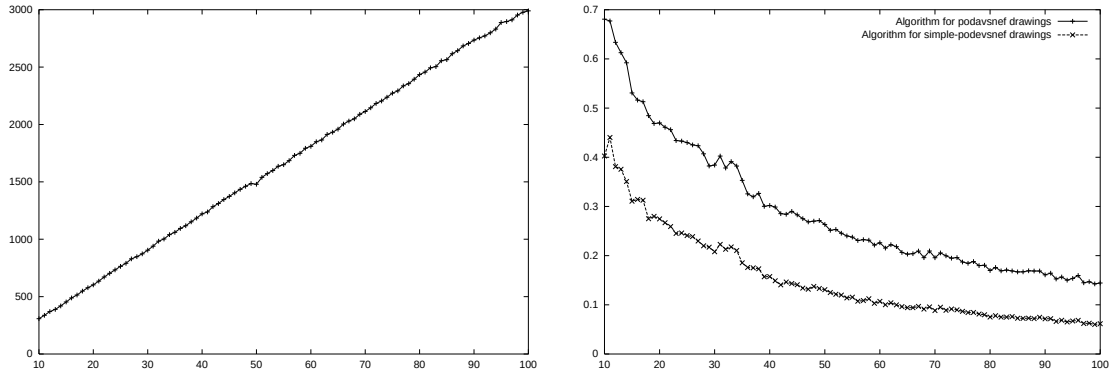


Figure 7: (a) Total vertex area with respect to the number of vertices. The width and the height of the vertices of any graph in the test-suite have been chosen by using a randomly probability distribution in the range $0 - 9$. (b) Percentage of the area of the drawing occupied by the vertices.

5 Computing *Pqudavs* Drawings

In this section we describe how to compute a *pqudavs* drawing of a planar digraph. Since the strategy for computing a planar quasi-upward drawing relies on the construction of a planar upward drawing of a planar st-digraph derived from it [6], we focus on the problem of constructing a *pqudavs* drawing of a planar st-digraph. The techniques can be straightforwardly applied to draw general planar digraphs.

Planar upward drawings of planar st-digraphs can be constructed by using, as intermediate drawing, a visibility representation [9]. *Visibility representations* map vertices to horizontal segments (*vertex-segments*) and edges to vertical segments (*edge-segments*). The vertical segment representing edge (u, v) has its endpoints on the horizontal segments representing vertices u and v, and does not intersect with any other horizontal segment.

The basic technique [9] to compute coordinates of the vertex-segments of a visibility representation of an st-digraph G consists of computing two optimal topological numberings, one performed on the st-digraph G and the other performed on its dual st-digraph G^*. We recall that a *topological numbering* of an st-digraph is an assignment of integer numbers to its vertices, such that, for each edge (u, v), the number assigned to v is greater than the one assigned to u. The numbering is *optimal* if the range of numbers assigned to the vertices is minimized. The y-coordinate of the vertex-segment representing vertex v is the number assigned to v by the optimal topological numbering of G. The left and right x-coordinates of the vertex-segment representing vertex v are: (left x-coordinate) the lowest number assigned by the optimal topological numbering of G^* to the vertices of the face associated with v; (right x-coordinate) the highest number assigned by the optimal topological numbering of G^* to the vertices

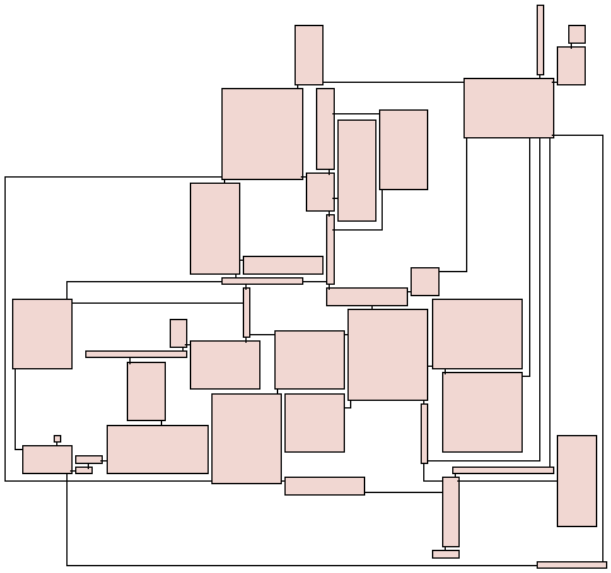


Figure 8: A podasvsnef drawing of a 40-vertices graph in the test-suite.

of the face associated with v minus one (See Fig. 9).

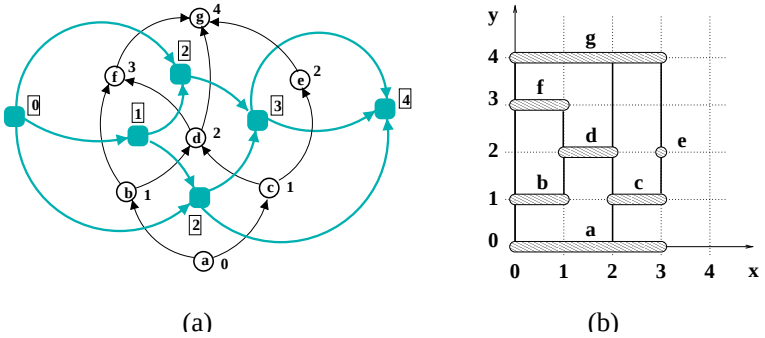


Figure 9: (a) An st-digraph (colored black and drawn upward) and its dual (colored grey). Labels are the result of two optimal topological numberings. (b) The corresponding visibility representation.

Given an st-digraph G and an assignment of specific height and width to each of its vertices, we compute a *pqudavs* drawing of G by using, as intermediate drawing, an *expanded* visibility representation. In the expanded visibility representation a vertex with assigned height h and width w is represented by a box (*vertex-box*) of height h and width greater than or equal to w, while an edge (u, v) is represented as usual by a vertical segment connecting the boxes representing vertices u and v, and not intersecting with any other vertex-box.

We assign the y-coordinates (x-coordinates) of the vertex-boxes of the expanded visibility representation by means of a network flow technique, computed on the flow network $\mathcal{N}_y$ ($\mathcal{N}_x$) defined as follows.

1. Starting from G, we construct a digraph G' (see Fig. 10.a) by replacing each vertex v of G with two new vertices v_{in} and v_{out} and with a new directed edge (v_{in}, v_{out}). The incoming edges of v become the incoming edges of v_{in} while the outgoing edges of v become the outgoing edges of v_{out}. Observe that G' is a planar st-digraph.
2. We compute the dual st-digraph G'^* of G'.
3. The nodes and arcs of $\mathcal{N}_y$ are (see Fig. 10.b) the vertices and edges of G'^*. Let v be a vertex of G with assigned height h. Let e be the edge of G'^* associated with (v_{in}, v_{out}). We set in $\mathcal{N}_y$

$lb(e) = ub(e) = h$. All the other arcs of $\mathcal{N}_y$ have lower bound equal to one and upper bound equal to ∞.

4. The nodes and arcs of $\mathcal{N}_x$ are the vertices and edges of G'. Let v be a vertex of G with assigned width w. We set in $\mathcal{N}_x$ $lb(v_{in}, v_{out}) = w + 1$. All the other arcs of $\mathcal{N}_x$ have lower bound equal to one. All the arcs have upper bound equal to ∞.

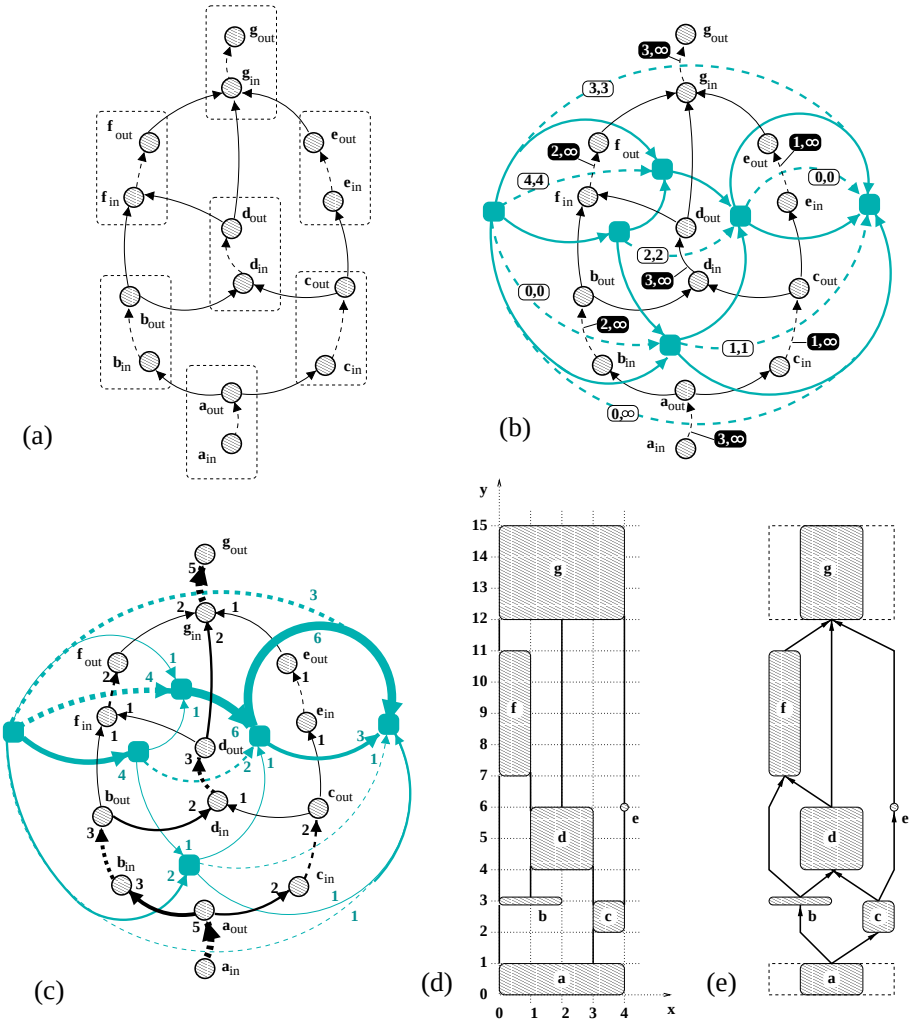


Figure 10: (a) The digraph G' obtained by splitting each vertex of the input graph G (the digraph of Fig. 9). (b) The network flows $\mathcal{N}_x$ (black arcs) and $\mathcal{N}_y$ (grey arcs). Black (white) labels show the most important lower and upper bounds of $\mathcal{N}_x$ ($\mathcal{N}_y$). Unspecified lower bounds and upper bounds are assumed to be 1 and ∞, respectively. The lower and upper bounds on dashed arcs are used to specify the vertex dimensions. (c) Two minimum flows for $\mathcal{N}_x$ and $\mathcal{N}_y$. The thickness of the arcs is proportional to their flow. Arcs with flow 0 are omitted. (d) The expanded visibility representation of G corresponding to the flow values in (c). (e) A *pqudavs* drawing of the digraph G with the vertices of the specified height and width.

Each unit of flow on an arc of the network $\mathcal{N}_x$ corresponds to a unit of width required by the corresponding edge or vertex-box of G. Each unit of flow on an arc of the network $\mathcal{N}_y$ corresponds to a unit of height of the corresponding edge or vertex-box of G. Therefore, from a pair of feasible flows of $\mathcal{N}_y$ and $\mathcal{N}_x$ the coordinates of the vertex-boxes and of the edge-segments of the expanded visibility representation are easily computed (see Fig. 10.c and 10.d).

Property 4 *Networks $\mathcal{N}_y$ and $\mathcal{N}_x$ are planar st-digraphs with $O(n)$ nodes, where n is the number of vertices of G.*

Observe that, since all the upper bounds on the arcs of $\mathcal{N}_x$ are set to ∞, there always exists a feasible flow in such network. Further, in the network $\mathcal{N}_y$ the only arcs with a bounded upper capacity are those corresponding to the edges of G' derived from the splitting of vertices. However, for each of these arcs there is a directed path connecting its end vertices, composed by arcs with infinite upper bound. Then ([1]) there always exists a feasible flow in $\mathcal{N}_y$.

Since networks $\mathcal{N}_y$ and $\mathcal{N}_x$ always have a feasible flow, and since from an expanded visibility representation a *pqudavs* drawing can be easily computed with the techniques illustrated in [9] and in [6], we have:

Theorem 2 *Let G be a quasi-upward planar digraph and suppose an assignment of widths and heights is given for the vertices of G. A* pqudavs *drawing of G always exists.*

The amounts of flow in $\mathcal{N}_y$ and $\mathcal{N}_x$ are in one-to-one correspondence with the height and width of the expanded visibility drawings. Hence, to construct drawings with limited width and height we can compute minimum flows on $\mathcal{N}_y$ and $\mathcal{N}_x$.

A further level of optimization of the drawing can be reached by assigning one unit of cost to the arcs of $\mathcal{N}_y$ and performing a min-cost-flow on $\mathcal{N}_y$ with the above minimum flow as feasible flow. Since units of flow represent lengths of the edge-segments, this technique allows to obtain drawings with reduced total edge length. Unfortunately, since there are many degrees of freedom in obtaining the st-digraph G from the input digraph, the reduced size of the drawing of the st-digraph does not imply the reduced size of the *pqudavs* drawing of the input graph.

Acknowledgements

We are grateful to Francesco Matera for useful discussions on the drawings of industrial plants.

References

[1] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. *Network Flows: Theory, Algorithms, and Applications.* Prentice Hall, Englewood Cliffs, NJ, 1993.

[2] C. Batini, E. Nardelli, and R. Tamassia. A layout algorithm for data flow diagrams. *IEEE Trans. Softw. Eng.*, SE-12(4):538–546, 1986.

[3] C. Batini, M. Talamo, and R. Tamassia. Computer aided layout of entity-relationship diagrams. *Journal of Systems and Software*, 4:163–173, 1984.

[4] P. Bertolazzi, G. Di Battista, and W. Didimo. Computing orthogonal drawings with the minimum number of bends. In F. Dehne, A. Rau-Chaplin, J.-R. Sack, and R. Tamassia, editors, *Proc. 5th Workshop Algorithms Data Struct.*, volume 1272 of *Lecture Notes Comput. Sci.*, pages 331–344. Springer-Verlag, 1997.

[5] P. Bertolazzi, G. Di Battista, and W. Didimo. Computing orthogonal drawings with the minimum number of bends. manuscript, 1998.

[6] P. Bertolazzi, G. Di Battista, and W. Didimo. Quasi-upward planarity. In S. H. Whitesides, editor, *Graph Drawing (Proc. GD '98)*, volume 1547 of *Lecture Notes Comput. Sci.*, pages 15–29. Springer-Verlag, 1998.

[7] T. Biedl and G. Kant. A better heuristic for orthogonal graph drawings. *Comput. Geom. Theory Appl.*, 9:159–180, 1998.

[8] T. C. Biedl. Relating bends and size in orthogonal graph drawings. *Inform. Process. Lett.*, 65:111–115, 1998.

[9] G. Di Battista, P. Eades, R. Tamassia, and I. G. Tollis. *Graph Drawing.* Prentice Hall, Upper Saddle River, NJ, 1999.

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

[11] S. Even. *Graph Algorithms.* Computer Science Press, Potomac, Maryland, 1979.

[12] U. Fößmeier, C. Hess, and M. Kaufmann. On improving orthogonal drawings: The 4M-algorithm. In S. Whitesides, editor, *Graph Drawing (Proc. GD '98)*, volume 1547 of *Lecture Notes Comput. Sci.*, pages 125–137. Springer-Verlag, 1999.

[13] U. Fößmeier and M. Kaufmann. Drawing high degree graphs with low bend numbers. In F. J. Brandenburg, editor, *Graph Drawing (Proc. GD '95)*, volume 1027 of *Lecture Notes Comput. Sci.*, pages 254–266. Springer-Verlag, 1996.

[14] U. Fößmeier and M. Kaufmann. Algorithms and area bounds for nonplanar orthogonal drawings. In G. Di Battista, editor, *Graph Drawing (Proc. GD '97)*, volume 1353 of *Lecture Notes Comput. Sci.*, pages 134–145. Springer-Verlag, 1998.

[15] T. Lengauer. *Combinatorial Algorithms for Integrated Circuit Layout*. Wiley-Teubner, 1990.

[16] T. Nishizeki and N. Chiba. Planar graphs: Theory and algorithms. *Ann. Discrete Math.*, 32, 1988.

[17] A. Papakostas and I. G. Tollis. Algorithms for area-efficient orthogonal drawings. *Comput. Geom. Theory Appl.*, 9(1–2):83–110, 1998. Special Issue on Geometric Representations of Graphs, G. Di Battista and R. Tamassia, editors.

[18] J. M. Six, K. G. Kakoulis, and I. G. Tollis. Refinement of orthogonal graph drawings. In S. Whitesides, editor, *Graph Drawing (Proc. GD '98)*, volume 1547 of *Lecture Notes Comput. Sci.*, pages 302–315. Springer-Verlag, 1999.

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

[20] R. Tamassia, G. Di Battista, and C. Batini. Automatic graph drawing and readability of diagrams. *IEEE Trans. Syst. Man Cybern.*, SMC-18(1):61–79, 1988.

[21] L. Valiant. Universality considerations in VLSI circuits. *IEEE Trans. Comput.*, C-30(2):135–140, 1981.