Labeling Heuristics for Orthogonal Drawings*

Carla Binucci, Walter Didimo, Giuseppe Liotta, and Maddalena Nonato

Università di Perugia ({binucci,didimo,liotta,nonato}@diei.unipg.it).

Abstract. This paper studies the problem of computing an orthogonal drawing of a graph with labels along the edges. Labels are not allowed to overlap with each other or with edges to which they are not assigned. The optimization goal is area minimization. We provide a unified framework that allows to easily design edge labeling heuristics. By using the framework we implemented and experimentally compared several heuristics. The best performing heuristics have been embedded in the topology-shape-metrics approach.

1 Introduction

The labeling placement problem has a long tradition in the computational geometry, computational cartography, and geographic information system communities. Several papers have been published in the literature presenting algorithms that receive as input a drawing Γ of a graph together with a set of text or symbol labels for the vertices and/or edges and produce as output a labeled drawing with some "good readability" qualities. For example, if Γ is a city map, then the streets must be easy to identify by their names.

Kakoulis and Tollis [12] define three basic requirements that a good labeling of a drawing should have: (i) For any label it must be easy to identify the edge or vertex to which the label is assigned. (ii) A label assigned to a vertex or edge cannot overlap other labels, vertices or edges. (iii) A label must be placed in the best possible position among all acceptable positions.

Depending on the application domain, the above three requirements are expressed in terms of different geometric constraints and optimization goals. Unfortunately, most labeling problems have been proved to be NP-hard in general even in their simplest forms where the given drawing consists of just a set of distinct points or a set of distinct straight-lines; efficient heuristics and polynomial solutions for restricted versions of these problems have been designed. These results have in common that the problem constraints do not allow to change the geometry of the drawing that must be labeled. The interested reader is referred to the on-line bibliography by Wolff and Strijk [17] for references on the subject.

In recent years the labeling problem has been receiving increasing attention also in the graph drawing community (see e.g. [2,6,13,10,11,15,14]). As several

* Research supported in part by the CNR Project "Geometria Computazionale Robusta con Applicazioni alla Grafica ed al CAD", the project "Algorithms for Large Data Sets: Science and Engineering" of the Italian Ministry of University and Scientific and Technological Research (MURST 40%).

authors remark (see e.g. [2,12,13]), the problem allows more flexibility in the graph drawing context where one can change the geometry of a drawing so to free up space for label insertions. In this area a challenging research direction is that of designing algorithms whose input is a graph G with a set of labels for its vertices or edges and whose output is a labeled drawing of G. Namely, most graph drawing algorithms have very poor support for computing labeled drawings and there is a clear need of integrating effective labeling strategies within graph drawing techniques in order to enlarge their range of applications.

For example, the well-known and widely used topology-shape-metrics approach for computing orthogonal drawings of planar graphs is not equipped with effective technology for inserting labels along the edges. One possible solution is to model the labels as dummy vertices and to apply the topology-shape-metrics approach that computes an orthogonal drawing where the dummy vertices are constrained to have fixed size [3]. However, it is not clear how to guarantee in this framework that the resulting labeled drawing has good readability qualities. Namely, the described strategy does not allow any control on choosing the best position for a label along an edge; if for example the graph to be drawn is large and optimizing the area of the drawing is a critical issue, choosing a segment or another for placing a label on an edge may deeply affect the readability of the output. The drawing on the right-hand side of Figure 1 is computed with a random segment selection strategy.

A pioneering work aiming at integrating labeling techniques with the topology-shape-metrics approach is due to Klau and Mutzel [13]. They study the problem of computing a grid drawing of an orthogonal representation with labeled vertices and minimum total edge length. Klau and Mutzel show an elegant ILP formulation of the problem and present the first branch-and-cut based algorithm that combines compaction and labeling techniques.

The present paper makes a further step in the direction defined by Klau and Mutzel by investigating one of the questions that they leave as open. Namely we focus on integrating the topology-shape-metrics approach with algorithms for edge labeling. A precise description of the problem we deal with is as follows: Let G be a planar graph, let H be an orthogonal representation of G, and let L be a set of labels for the edges of G, where each edge is associated with at most one label. We want to compute an orthogonal grid drawing Γ of G such that the edges of Γ are labeled and have the shape defined in H. A label is modeled as an axis parallel rectangle of given integer width and height.

Our geometric constraints and optimization goals are as follows: (i) A label is drawn with one of its sides as a proper subset of a segment of the corresponding edge (the label is "glued" to the edge, so the assignment is unambiguous); (ii) Each label λ associated with a segment s cannot overlap any other label, vertex, or segment except s; (iii) A good placement for our labels is a placement that minimizes the area of the drawing.

We observe that finding an optimal solution for our problem can be too expensive in practice, since the instance where every edge has no associated label coincides with the well known NP-hard compaction problem [16] for orthogo-

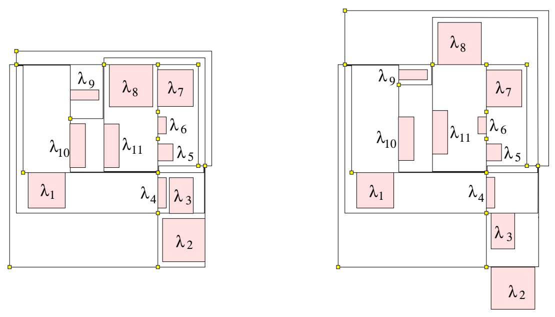

Fig. 1. Two different labeled drawings of the same labeled orthogonal representation. The one the right is computed by choosing randomly the segment and the direction of each label.The one on the left is computed by choosing the segment and the direction of each label with one of our heuristics.

nal representations. We investigate the edge labeling problem with an approach different from that followed by Klau and Mutzel for vertex labeling: Instead of looking for an ILP formulation, we implement simple and robust heuristics and experimentally compare their performances. A more detailed description of our results is given below:

- We define a general framework for constructing heuristics to solve the edge labeling problem for orthogonal representations. The proposed framework is based on a greedy strategy that computes a drawing by inserting a label at a time. The optimization goal is area minimization.
- By using the above framework we designed different algorithms for the edge labeling problem and experimentally compared their performances. We used techniques such as local search and greedy randomized adaptive search procedures (GRASP) [7] to investigate the effectiveness of our heuristics in practice.
- We embedded the best performing heuristics within the topology-shape-metrics approach. Namely, the implementation of these heuristics has enriched the topology-shape-metric technology of the GDToolkit library [9] for computing orthogonal drawings. We designed and implemented two new algorithms, called `Fast Labeler` and `Slow Labeler` that receive as input an edge labeled graph (with vertices of unbounded degree) and produce a compact orthogonal drawing of G. The drawing on the left-hand-side of Figure 1 is computed with the `Slow Labeler` and is the same graph as the one on the right-hand side.

2 Preliminaries

We assume familiarity with basic definitions on graph connectivity, graph planarity, and graph drawing [4]. An *orthogonal (grid) drawing* of a graph is a drawing such that the vertices are represented as points of an integer grid and the edges are represented as chains of horizontal and vertical segments on the integer grid lines. A *row* (*column*) of an integer grid is a strip of the plane between two horizontal (vertical) consecutive lines of the grid. A *row* (*column*) of an orthogonal drawing Γ is a row (column) of the integer grid that intersects Γ. An *orthogonal representation* of an embedded (planar) graph G is an equivalence class of planar orthogonal drawings such that the following holds: (i) For each edge (u, v) of G, all the drawings of the class have the same sequence of left and right turns (*bends*) along (u, v), while moving from u to v. (ii) For each vertex v of G, and for each pair $\{e_1, e_2\}$ of clockwise consecutive edges incident on v, all the drawings of the class determine the same angle between e_1 and e_2. Roughly speaking, an orthogonal representation defines a class of planar orthogonal drawings that may differ only for the length of the segments of the edges.

One of the most popular techniques for computing an orthogonal drawing of a graph G is the so called *topology-shape-metrics* approach [18,19,4]. It consists of three consecutive steps: (i) `Planarization:` An embedding of G is computed, possibly adding dummy vertices to replace crossings. (ii) `Orthogonalization:` An orthogonal representation H of G is computed within the previously computed embedding. (iii) `Compaction:` A final geometry for H is determined. Namely, coordinates are assigned to vertices and bends of H. The distinct phases of the topology-shape-metrics approach have been extensively studied in the literature [4].

Since each vertex of a planar orthogonal drawing is a grid point and since planarity does not allow distinct edges to overlap, each vertex can have at most degree equal to four. Of course, this is a severe limitation for most applications. In order to orthogonally draw graphs of arbitrary vertex degree, different drawing conventions have been introduced in the literature. Here we refer to as the *podevsnef* (planar orthogonal drawing with equal vertex size and not empty faces) drawing convention, defined by Fößmeier and Kaufmann [8]. A *podevsnef* drawing (see Figures 2 (a) and 2 (b)) is an orthogonal drawing such that: (i) Segments representing edges cannot cross, with the exception that two segments that are incident on the same vertex may partially overlap. Observe that the angle between such segments has zero degree. Roughly speaking, a podevsnef drawing is "almost" planar: it is planar everywhere but in the possible overlap of segments incident on the same vertex. Observe in Figure 2 (b) the overlap of segments incident on vertices 1, 2, and 3. (ii) All the polygons representing the faces have area strictly greater than zero.

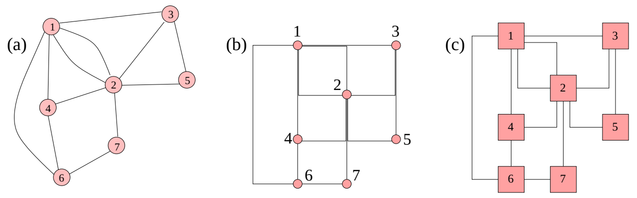

Fig. 2. (a) A planar graph and (b) one of its podevsnef drawings; (c) A more effective visualization of the podevsnef drawing in (b).

Podevsnef drawings are usually visualized representing vertices as boxes with equal size and representing two overlapping segments as two very near segments. See Figure 2 (c). 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, each zero degrees angle is in correspondence with exactly one bend [8]. An orthogonal representation corresponding to the above definition is a *podevsnef orthogonal representation*. In a podevsnef orthogonal representation we allow label assigment only to segments that do not overlap (note that each edge has a non-overlapping portion of segment). From now on we shall use the term orthogonal drawing and orthogonal representation for denoting podevsnef orthogonal drawings and podevsnef orthogonal representations, respectively. Also, when we talk about a segment we always refer to a segment (or portion of a segment) of an orthogonal representation that does not overlap with any other segment.

We conclude this section with some geometric definitions that will be used from here on to describe the different label insertion strategies. Let Γ be an orthogonal drawing, and let s be a horizontal (vertical) segment of Γ with endpoints a and b. Assume that a is to the left of b if s is horizontal and that a is below b if s is vertical. Let R be an axis aligned rectangle such that one of the sides of R is a subset of s. We call *position* of R with respect to s the distance between R and a. If such a distance is an integer number the position of R with respect to s is an *integer position*. If R lies on the left-hand side of s while moving from a, we say that R has *the left direction* with respect to s; otherwise R has *the right direction* with respect to s. R is an *empty rectangle* if the interior of R does not contain any point of Γ.

3 A Unified Framework for Greedy Labelers

In this section we describe a general greedy strategy for the edge labeling problem for orthogonal drawings.

3.1 The Greedy Labeler

Let H be an orthogonal representation and let L be a set of labels for the edges of H. We call our general strategy `Algorithm Greedy Labeler`. The algorithm is based on a greedy approach. It first computes a drawing of H with no labels, and then it performs $|L|$ steps. At each step a new label is selected and inserted in the current drawing. The insertion is performed by possibly stretching some of the edges in order to avoid intersections between any two labels or between a label and an edge. Let $\lambda \in L$ be the label of and edge e of H such that λ has not been inserted in the drawing yet. We associate λ with a pair $< place(\lambda), cost(\lambda) >$ such that: (i) $place(\lambda)$ is the *drawing placement* of λ. It is a triplet $< s, d, p >$, where s is the segment of e on which λ will be drawn, d specifies the direction (left or right) of λ with respect to s, and p defines the integer position of λ

with respect to s. (ii) $cost(\lambda)$ is the *drawing cost* of λ. It measures the "price" that must be paid when inserting λ in the current drawing with the constraints defined by its drawing placement. Different definitions of this price give rise to different greedy heuristics. For example, $cost(\lambda)$ can measure how much the area of the current drawing is increased in order to accommodate λ, or it can also look ahead and estimate whether inserting λ can lower the cost of some next insertions.

The insertion of λ in the current drawing is performed by invoking a suitable `Insert Label Procedure` that receives as input λ and the current labeled drawing Γ and computes a new labeled drawing containing all labels in Γ plus λ. The basic idea is to enlarge Γ by inserting a minimal number of columns and/or rows that are needed to draw λ without overlaps. Since the insertion of a new label can change the geometry of the current drawing, it may be needed to update the values of the drawing placement and cost for some of the remaining labels that have not been inserted yet. Namely, there are two main implications of the insertion of λ in the drawing: (i) Inserting rows or columns in the drawing causes additional free area in some faces. This can imply the reduction of the cost of some labels to be drawn yet. (ii) Drawing λ in a face f reduces the free area in f and this can lead to an increase of the drawing cost of some other labels, for example those that are going to be drawn inside f. In both cases the drawing cost of some of the remaining labels must be updated after the insertion of λ. Let λ' be one of those labels whose drawing cost and placement is affected by the insertion of λ; `Algorithm Greedy Labeler` invokes a `Cost Assignment Procedure` that receives as input λ' and the current drawing and computes as output the values $< place(\lambda'), cost(\lambda') >$. A detailed description of `Algorithm Greedy Labeler` is given below.

```
Algorithm Greedy Labeler
input: An orthogonal representation $H$ and a set $L$ of labels for the edges of $H$.
output: A labeled drawing $\Gamma$ of $H$.

Step 1: Preprocessing Phase.
    – Compute a grid drawing $\Gamma_0$ of $H$ with no labels
    – Copy all labels in a set $Q$
    – for each label $\lambda \in Q$ execute Cost Assignment Procedure($\lambda$,$\Gamma_0$)
Step 2: Labeling Phase.
    – for $i := 1$ to $|L|$ perform the following three steps
      Step 2.1: Label Selection.
          • let $\lambda \in Q$ be the label such that $cost(\lambda)$ is minimum
          • Remove $\lambda$ from $Q$
      Step 2.2: Label Insertion. Compute a new drawing $\Gamma_i$ by executing
          Insert Label Procedure ($\lambda$,$\Gamma_{i-1}$)
      Step 2.3: Cost Update.
          • for each label $\lambda' \in Q$ that may be drawn in some of the faces
            whose free area is changed after the insertion of $\lambda$, execute Cost
            Assignment Procedure($\lambda$,$\Gamma_i$)
    – set $\Gamma = \Gamma_{|L|}$
```

`Step 1` (the `Preprocessing Phase`) can be accomplished by means of a standard technique for compacting (podevsnef) orthogonal representations [8, 1]. `Step 2` strongly relies on the `Cost Assignment Procedure(`λ`,`Γ_i`)` and on the `Insert Label Procedure (`λ`,`Γ_{i-1}`)` which are the subject of the next two subsections.

3.2 The Cost Assignment Procedure

The `Cost Assignment Procedure` receives as input the label λ of an edge e and a drawing Γ where e is not labeled yet. For each segment s of e, the procedure moves λ along s and computes a pair $< place_s(\lambda), cost_s(\lambda) >$. $place_s(\lambda)$ is a triplet $< s, d, p >$ where p and d are the integer position and direction of λ with respect to s that have minimum cost; this minimum cost is stored in $cost_s(\lambda)$. Pair $< place_s(\lambda), cost_s(\lambda) >$ is computed by means of a suitable *drawing cost function* that we denote as $CF(\lambda, s, \Gamma)$ and that is the kernel of the `Cost Assignment Procedure`. Finally, the output of the `Cost Assignment Procedure` is the pair $< place(\lambda), cost(\lambda) >$ corresponding to the minimum value of $cost_s(\lambda)$ over all segments s of e.

```
Cost Assignment Procedure($\lambda$,$\Gamma$)
  – let $e$ be the edge of $\Gamma$ where $\lambda$ must be drawn
  – set $place(\lambda) =< nil, nil, nil >$
  – set $cost(\lambda) = +\infty$
  – for all segments $s$ of $e$
      • set $< place_s(\lambda), cost_s(\lambda) >= CF(\lambda, s, \Gamma)$
      • if $(cost_s(\lambda) < cost(\lambda))$ then
        set $< place(\lambda), cost(\lambda) >=< place_s(\lambda), cost_s(\lambda) >$
  – return $< place(\lambda), cost(\lambda) >$
```

We are now ready to provide more details about $CF(\lambda, s, \Gamma)$. For concreteness, we assume that s is vertical (the case where s is horizontal is handled analogously). Let h_λ denote the height of λ and let h_s denote the height of s. We distinguish between two cases.
Case 1: $h_\lambda + 2 \leq h_s$ (this implies that λ can be inserted in Γ on s without stretching s). In this case $CF(\lambda, s, \Gamma)$ analyzes each integer position of λ with respect to s and for each such position it considers the two possible directions (left and right) of λ with respect to s. For a pair $< p, d >$, where p is a given integer position and d is a given direction of λ with respect to s, $CF(\lambda, s, \Gamma)$ executes the following two tasks:

`Task 1:` It evaluates a minimal number of rows and columns that must be added in order to properly draw λ in Γ. Let r and c be the computed number of rows and columns, respectively.

`Task 2:` It computes a cost C that depends on r, c, Γ and the dimensions of λ. If C is lower than $cost_s(\lambda)$, we set $cost_s(\lambda) = C$ and $place_s(\lambda) =< s, d, p + 1 >$. Different heuristics for the labeling problem can be designed within our greedy framework and by changing the definition of C (see also Section 4).

`Task 1` applies the following rule (refer to Figure 3). Let h_λ and w_λ be the height and the width of λ, respectively. We compute the largest empty rectangle R such that the position of R with respect to s is p, the direction of R with respect to s is d, and the width of R is equal to $w_\lambda + 1$. Let h_R be the height of R. There are two cases to consider:

- $h_R \geq h_\lambda + 2$. In this case λ can be drawn inside R, in position $p + 1$ with respect to s, and without inserting any rows or columns to the drawing. Hence, r and c are both set to be 0. See for example in Figure 3 (a), where d is the right direction.
- $h_R < h_\lambda + 2$. In this case we insert only rows or only columns in the drawing in order to place λ in position $p + 1$ with respect to s (see for example Figure 3 (b)). The number c of columns is defined as follows. Assume that d is the right direction (the reasoning is symmetric for the case that d is the left direction) and let f be the face of Γ on the right-hand side of s. Let s_d be the closest segment (not touching s) on the boundary of f on the right-hand side of s that has one point in common with the boundary of R; let δ be the distance between s and s_d. See for example Figure 3 (b). Then c is set to $c = w_\lambda + 1 - \delta$. As for the value of r, we distinguish between the case that R has 0 height and the case that R has positive height. In the first case, r is set to be $r = +\infty$. In the second case, r is set to be $r = h_\lambda + 2 - h_R$. If $min(c, r) = r$ then we set $c = 0$, else we set $r = 0$.

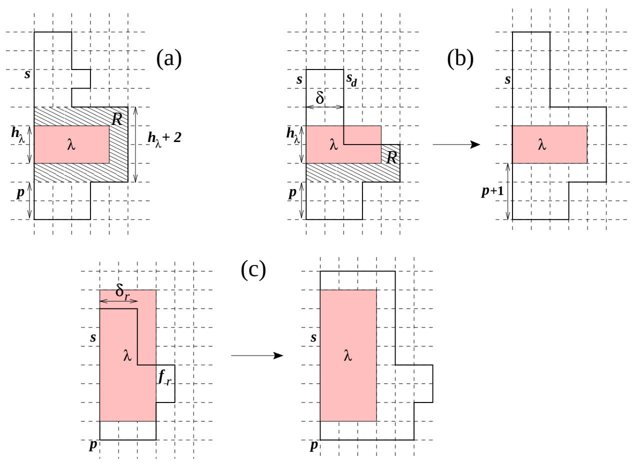

Fig. 3. (a) Label λ can be placed in position $p + 1$ without inserting rows or columns (b) Label λ can be placed in position $p + 1$ adding two rows or three columns. It is inserted by adding two rows. (c) The height of the label plus two is larger than the length of the segment. To place the label we insert two rows and two columns.

Case 2: $h_\lambda + 2 > h_s$ (this implies that s must be stretched in order to support λ). $CF(\lambda, s, \Gamma)$ executes the following operations (see Figure 3 (c)):

– It sets $p = 0$.
– It sets $r = h_\lambda + 2 - h_s$. Also, let f_l be the face of Γ on the left-hand side of s, and let f_r be the face of Γ on the right-hand side of s. Let $c_l = max\{0, w_\lambda + 1 - \delta_l\}$, where δ_l is the minimum distance between s and a segment (not touching s) on the boundary of f_l that is on the left-hand side of s. Let $c_r = max\{0, w_\lambda + 1 - \delta_r\}$, where δ_r is the minimum distance between s and a segment (do not touching s) on the boundary of f_r that is on the right-hand side of s. Let $c = min(c_l, c_r)$. If $c = c_l$ it is set $d = d_l$, else it is set $d = d_r$.
– It computes a cost C that depends on r, c, Γ and the dimensions of λ. It sets $cost_s(\lambda) = C$ and $place_s(\lambda) =< s, d, p + 1 >$.

3.3 The Insert Label Procedure

The `Insert Label Procedure(`λ`,`Γ`)` draws label λ in drawing Γ. Label λ is placed in Γ according to $place(\lambda) =< s, d, p >$. The number of rows and columns to be added to Γ so to avoid overlaps is computed as described for $CF(\lambda, s, \Gamma)$. Actually, in order to save computation time one can store the number of rows and columns computed by $CF(\lambda, s, \Gamma)$ into two variables $r(\lambda)$ and $c(\lambda)$ and pass these information to the `Insert Label Procedure(`λ`,`Γ`)`.

We provide some details about the insertion strategy assuming that s is vertical and that d is the right direction (see Figure 3 (b) and Figure 3 (c)). The other cases are handled similarly. The value of position p with respect to s defines a unique point along segment s; let l_h and l_v be the horizontal line and the vertical line through this point, respectively. Let H_u be the closed half-space above l_h and let H_r be the open half-space to the right of l_v. The following tasks are executed: (i) $r(\lambda)$ rows are inserted in Γ by translating of $r(\lambda)$ units to the North direction all points of Γ that lie in H_u (ii) $c(\lambda)$ columns are inserted in Γ by translating of $c(\lambda)$ units to the right direction all points of Γ that lie in H_r. (iii) λ is drawn in position p and direction d with respect to s. Once inserted, λ will be treated by the greedy algorithm as a new face whose segments cannot be stretched.

The following theorem summarizes the complexity of `Algorithm Greedy Labeler` in the worst case. Such a complexity is obtained by considering the case in which all the costs of the remaining labels have to be updated at each new greedy step and by considering the complexity of function $CF()$. We omit the proof due to space limitations, but we observe that our experiments show better time performance in practice than the one theoretically estimated. This can be justified by a more careful analysis of the practical situations. In Section 4 we give some intuitions that are behind such a behavior.

Theorem 1. *Let H be an orthogonal representation of a planar graph with n vertices, and let L be a set of labels for the edges. There exists a general greedy algorithm that computes a labeled drawing Γ of H in $O(|L|nT_L)$ time, where T_L is the total length of the labeled edges of Γ.*

4 Experimental Comparison of Different Labeling Heuristics

We designed three basic heuristics for edge labeling within the framework of `Algorithm Greedy Labeler`. Such heuristics differ for the definition of cost adopted by the `Assignment Procedure` where selecting the label with highest insertion priority. We use the notation of the previous sections and denote with h_λ and w_λ the height and the width of λ, respectively. Also, r and c denote the number of rows and columns computed by $CF(\lambda, s, \Gamma)$. We denote with Δ_A the area increase (measured in terms of grid points in the bounding box of the drawing) of Γ implied by inserting r rows and c columns in Γ with the `Insert Label Procedure`. The three heuristics are as follows.

`Delta-area:` The label with highest insertion priority is the one that causes the minimum increase of area. Therefore, $C = \Delta_A$.

`Max-size-delta-area:` The label with highest insertion priority is the one with maximum area. If two labels have the same area, the label that implies the minimum increase of the area is chosen. Therefore $C = \Delta_A - K(h_\lambda+1)(w_\lambda+1)$ where K is a constant such that $K >> \Delta_A$.

`Max-ratio-delta-area:` The label with highest insertion priority is the one with maximum aspect ratio. If two labels have the same aspect ratio, it is chosen the one that causes the minimum increase of the area. Therefore $C = \Delta_A - Kmax\{(h_\lambda+1)/(w_\lambda+1), (w_\lambda+1)/(h_\lambda+1)\}$, where K is a constant such that $K >> \Delta_A$.

We implemented and experimentally compared the performances of the above three heuristics. The implementation uses a PC-Pentium III (800 MHz and 256 MB RAM), Linux RedHat 6.2 O.S., gnu g++ compiler, and the GDToolkit library [9]. The experimental analysis measured the following quantities:CPU time, area, total edge length, and screen ratio of the labeled drawings.

The test suite for the experiments is a variant of the real-world graphs extensively used in the graph drawing community for experimental analysis (the so-called "Rome-graphs") [5]. Namely, for each graph of this set we computed an orthogonal representation H with a variation of the technique in [8] and randomly generated labels to be assigned to the edges. The assignment of a label to an edge is done by a coin flip (i.e. one edge gets a label with probability 0.5). For each label λ, h_λ and w_λ are in the range of integer values $0-5$ and are defined at random with uniform probability distribution (a label whose height and width are both 0 is a grid point). The resulting test suite consists of about eleven thousand labeled graphs that are grouped into families according to their number of vertices. The number of vertices ranges from 10 to 100.

Concerning the screen ratio the three heuristics have very similar behaviour. The charts of the total edge length have the same behaviour as those of the area, although the differences between the three heuristics are less remarkable for total edge length. For reasons of space, we omit the charts on screen ratio and total edge length and show only a subset of those about area and CPU time. All our experimental results have the property that the curves relative to different heuristics almost never overlap and are always monotonically increasing. Since differences of $5 - 10\%$ may appear not very readable in small figures, we shall often display only charts relative to a subset of the test suite while talking about the performance of the heuristics over the whole test set.

Figure 4 (c) shows the CPU time performance (average values) of the three heuristics. We observe that they look fast in practice since they never require more than two seconds even for the largest graph instances of the test suite. In the implementation of `Algorithm Greedy Labeler` we used the Fibonacci heap data structure to implement Q, that supports selections and updates in constant and logarithmic amortized time. The complexity of updating the cost of a label λ depends on the computation of function $CF(\lambda, s, \Gamma)$. Although in the worst case this computation can require evaluating a number of positions for λ that is equal to the length of s, in many cases it evaluates only a small number of positions. In fact, if λ is so "big" that s must be stretched to accommodate it, just one integer position for λ is taken into account (see **Case 2** of Section 3.2). Conversely, if λ is "small" $CF(\lambda, s, \Gamma)$ usually finds in a few steps a position for λ that do not require to insert rows or columns and then stops its computation. Further, consider that portions of segments that overlap are discarded.

Figure 4 (a) compares the (average) areas of the drawings computed by the three heuristics for graphs with number of vertices in the range $80 - 100$. Heuristics `max-ratio-delta-area` is the best performing and improves the solutions of heuristic `delta-area` by at least an $4 - 5\%$ factor; there are also cases in which such an improvement is much greater. In our opinion this result is a consequence of the fact that `max-ratio-delta-area` gives higher priority to "skinny" labels, i.e. those that have high aspect ratio. Inserting a skinny label often implies inserting extra rows and/or columns and enlarging the size of several faces in the current drawing. As a consequence, other labels can be inserted without any further insertions of rows and columns. Also, a skinny label λ occupies only a small portion of the face f in which it is drawn; hence inserting λ does not increase too much the cost of other labels that have to be placed in f. On the other extreme, `delta-area` does not take into account in any way the impact that inserting a label in a face can have on further label insertions. The behavior of `max-ratio-delta-area` is in-between these two extremes: it draws first those labels that have large area but this is somehow not properly reflecting our rows/columns insertion strategy that, when possible, stretches the drawing only in one direction for each label insertion. In order to test the robustness of our techniques and further verify the above results we repeated the experiments on the same graphs but where label dimensions vary in the range $5 - 10$. Figure 4 (b) shows the same trend of performance as the one of Figure 4 (a) and it even emphasizes the difference between `delta-area` and `max-ratio-delta-area` for some graph instances.

Motivated by the above experiments we ran several new experiments to further investigate the effectiveness of the `max-ratio-delta-area` heuristic. Because of the NP-hardness of the problem, estimating how much the feasible solutions of the `max-ratio-delta-area` heuristic are far from optimum is not an easy task. Nevertheless, we executed four experiments that provide evidence of the good behavior of the heuristic. Let Γ be the labeled drawing computed by `max-ratio-delta-area` for a labeled graph G and let S be the solution space, i.e. the set of all possible labeled orthogonal drawings of G. Our experiments pick other feasible solutions in S and compare them with Γ.

We first compared the areas of drawings computed by `max-ratio-delta-area` with the area of the drawings computed by a trivial greedy heuristic, called `random`, that chooses randomly both the ordering in which labels will be inserted and the segment and the drawing placement of each label. Figure 4 (d) shows that `max-ratio-delta-area` outperforms `random` by about 20% for most graphs instances.

A second experiment iteratively executes `random` until a labeled drawing with area less than or equal to the area of the output of `max-ratio-delta-area` is found. Figure 4 (e) shows the CPU time (in seconds and logarithmic scale) spent by `random` to find a solution less than or equal to the one computed by `max-ratio-delta-area` for the same graph. In order to execute such an experimentation in a reasonable time, we performed it over a subset of graphs of the test suite, with maximum running time of 30 minutes for each graph.

A third experiment uses local search to explore the neighborhood of Γ in S. We visit the neighborhood of Γ searching for better solutions. Clearly, setting the neighborhood to be searched involves a tradeoff between solution quality (the larger the neighborhood the better the solution that can be found) and running time (larger neighborhoods require longer time to be searched). Our experiment is as follows: For each label λ assigned to a segment s, we tried to flip λ around s while preserving the directions of the other labels and ran again `max-ratio-delta-area` with the constraint that the labels are inserted in the same ordering as the one used for constructing Γ. If a better solution is found (i.e. for the flipping of some of the labels) we move on with a new solution and iterate the search, otherwise it is stopped. Figures 4 (f) and (g) show our experimental results about the area and CPU time.

In order to overcome the inner limit of local search strategies that get trapped in a local optimum, we ran a fourth experiment that includes our local search within a greedy randomized adaptive search procedure (GRASP)[7]. A GRASP repeatedly starts the search from different solution points in the feasible region. Such points are not selected at random as in classical multi-start methods, but are computed by perturbing the criterion of the `max-ratio-delta-area` heuristic when selecting the best current label. Our GRASP is based on choosing at random the label whose cost lies in the range $[\chi_m, \chi_m + (\chi_M - \chi_m)\alpha]$ where α is a real in the interval $[0, 1]$ and χ_m and χ_M are the minimum and the maximum costs of the labels considered in the selection step. We then apply a local search approach to each one of the different solutions and keep the best solution among the computed ones. On each graph we performed 50 GRASP iterations. Figures 4 (h) and (i) show the area and the CPU time experimental results for $\alpha = 0.1$ and $\alpha = 0.3$.

Both the local search approach and the GRASP show that solutions with smaller area than those computed by the `max-ratio-delta-area` heuristic can be obtained by paying a relatively high price in terms of computation time. The experimental data show that the obtained area improvement is for most case not larger than $7 - 11\%$.

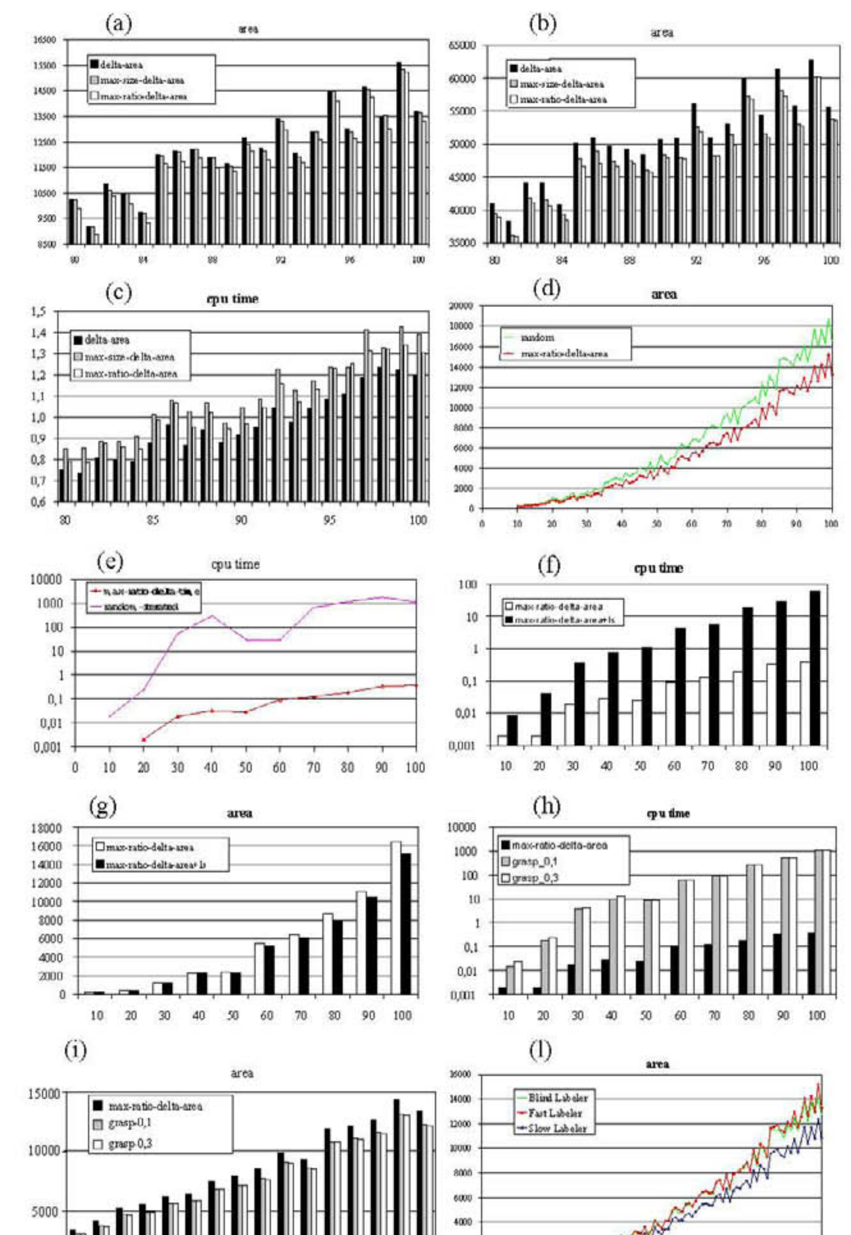

Fig. 4. Charts of the experiments. The x-axis represents the number of vertices. The y-axis reports average values.

We integrated the `max-ratio-delta-area` heuristic in the topology-shape-metrics approach that computes an orthogonal drawing of a graph. The main reason that led us to design an integrated algorithm is the following experiment. We first designed an algorithm that we call `Blind Labeler` based on the podevsnef model. `Blind Labeler` receives as input a labeled graph G, it discards the labels form G and computes an orthogonal representation H of G by using minimum cost flow techniques. Each edge $e \in H$ with a label λ is split into two edges, by inserting a dummy vertex representing λ. The segment of e where the dummy vertex is inserted is chosen at random. Finally, `Blind Labeler` runs a compaction step that computes a drawing of H where the dummy vertices are constrained to have fixed size [3]. We then designed a second algorithm that we call `Fast Labeler` that computes an orthogonal drawing of G by using the `max-ratio-delta-area` heuristic. Namely, `Fast Labeler` differs from `Blind Labeler` since it uses the `max-ratio-delta-area` heuristic to insert the labels in H and to compute the drawing of G. We observed that the areas are comparable (see Figure 4 (l)) while the CPU time of `Fast Labeler` is much less than that of `Blind Labeler`. The difference in CPU time performance is due to the fact that the compaction step of `Blind Labeler` relies on several iterations of minimum cost flow computations. We also note that `Blind-Labeler` chooses the drawing placement of a label at random and the result of Figure 4 (d) shows that a random choice for the labels placement in general leads to solutions that are much worse than those achievable with the `max-ratio-delta-area` heuristic. Hence, the reason why the two algorithms have comparable performances in terms of area strongly depends on the effectiveness of the compaction step used by `Blind Labeler`. To foster our intuition we designed a third integrated algorithm, that we call `Slow Labeler`. The `Slow Labeler` first computes H in the same way as `Fast Labeler` does, then it uses the strategy of the `max-ratio-delta-area` heuristic to define the segment and the direction of each label, and finally it applies the compaction step of `Blind Labeler` to H taking into account the constraints for the segments and the directions of the labels. Figure 4 (l) also shows the performances of `Slow Labeler`.

The areas of the drawings computed by `Slow Labeler` are considerably below those computed by the other two algorithms. The CPU time spent by `Slow Labeler` is similar to that spent by `Blind Labeler`.

5 Open Problems

In the near future we plan to investigate other heuristics within the proposed greedy framework and to devise effective polyhedral techniques in order to exactly evaluate the solution quality of our heuristics.

References

1. P. Bertolazzi, G. Di Battista, and W. Didimo. Computing orthogonal drawings with the minimum numbr of bends. *IEEE Transactions on Computers*, 49(8), 2000.

2. R. Castello, R. Milli, and I. Tollis. An algorithmic framework for visualizing statecharts. In *Proc. GD '00*, volume 1984 of *LNCS*, pages 139–149, 2001.
3. G. Di Battista, W. Didimo, M. Patrignani, and M. Pizzonia. Orthogonal and quasi-upward drawings with vertices of arbitrary size. In *Proc. GD '99*, volume 1731 of *LNCS*, pages 297–310, 2000.
4. G. Di Battista, P. Eades, R. Tamassia, and I. G. Tollis. *Graph Drawing.* Prentice Hall, Upper Saddle River, NJ, 1999.
5. 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.
6. U. Dogrusoz, K. G. Kakoulis, B. Madden, and I. G. Tollis. Edge labeling in the graph layout toolkit. In *Proc. GD '98*, volume 1547 of *LNCS*, pages 356–363, 1999.
7. T. A. Feo and M. G. C. Resende. Greedy randomized adaptative search procedure. *Journal of Global Optimization*, 6:109–133, 1995.
8. U. Fößmeier and M. Kaufmann. Drawing high degree graphs with low bend numbers. In *Proc. GD '95*, volume 1027 of *LNCS*, pages 254–266, 1996.
9. GDToolkit. Graph drawing toolkit. On line. http://www.dia.uniroma3.it/∼gdt.
10. K. G. Kakoulis and I. G. Tollis. On the edge label placement problem. In *Proc. GD '96*, volume 1190 of *LNCS*, pages 241–256, 1997.
11. K. G. Kakoulis and I. G. Tollis. An algorithm for labeling edges of hierarchical drawings. In *Proc. GD '97*, volume 1353 of *LNCS*, pages 169–180, 1998.
12. K. G. Kakoulis and I. G. Tollis. On the complexity of the edge label placement problem. *Comput. Geom. Theory Appl.*, 18:1–17, 2001.
13. G. W. Klau and P. Mutzel. Combining graph labeling and compaction. In *Proc. GD '99*, volume 1731 of *LNCS*, pages 27–37, 2000.
14. G. W. Klau and P. Mutzel. Optimal labelling of point features in the slider model. In *Proc. COCOON '00*, volume 1858 of *LNCS*, pages 340–350, 2000.
15. S. Nakano, T. Nishizeki, T. Tokuyama, and S. Watanabe. Labeling points with rectangles of various shapes. In *Proc. GD '00*, volume 1984 of *LNCS*, pages 91–102, 2001.
16. M. Patrignani. On the complexity of orthogonal compaction. *Comput. Geom. Theory Appl.*, 19:47–67, 2001.
17. T. Strijk and A. Wolff. The map-labeling bibliography. http://www.math-inf.uni-greifswald.de/map-labeling/bibliography/.
18. R. Tamassia. On embedding a graph in the grid with the minimum number of ben ds. *SIAM J. Comput.*, 16(3):421–444, 1987.
19. 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.