A Software System for Computing Labeled Orthogonal Drawings of Graphs

(Extended Abstract)

Carla Binucci [1]

Dipartimento di Ingegneria Elettronica e dell'Informazione
Universitá di Perugia
Perugia, Italy

Walter Didimo [2]

Dipartimento di Ingegneria Elettronica e dell'Informazione
Universitá di Perugia
Perugia, Italy

Abstract

The paper presents a software system for computing orthogonal drawings of graphs with labels on vertices and edges, while minimizing the area or the total edge length of the drawing. The system is thought mainly to support CASE tools in the automatic visualization of diagrams like UML diagrams and ER-diagrams.

1 Introduction

The increasing complexity of software systems and information systems has caused a significant demand of technologies for the automatic visualization of diagrams, like for example UML diagrams and ER-diagrams (see Figure 1). These diagrams can be modeled by graph structures, where vertices represent objects and edges represent relationships between pairs of objects. In addition, textual or graphic labels are usually associated with vertices and edges of the diagrams.

Several CASE tools offer sophisticated graphic editors to create diagrams used in the design of a software or in the design of a database, but it is still

[1] Email: binucci@diei.unipg.it
[2] Email: didimo@diei.unipg.it

Preprint submitted to Elsevier Preprint

poor the supply of functionalities to automatically draw diagrams in such a way that they are "readable" for the user. Readability of diagrams is usually reached by taking into account the optimization of several drawing aesthetics, like for example the area occupied by the drawing, the length of the edges, the number of crossings between edges, vertices, and labels, the number of bends along the edges, the angular resolution of the edges.

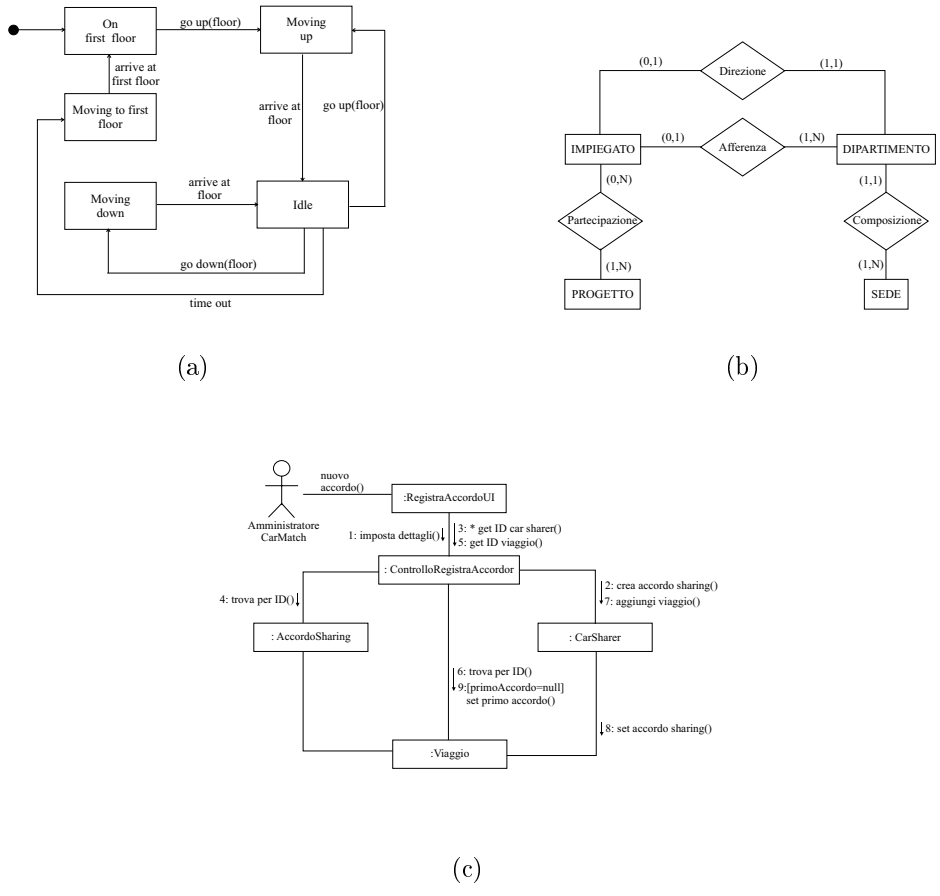

Fig. 1. Diagrams from real-world applications. (a) A state-transition diagram, (b) an ER-diagrams, (c) A collaboration diagrams.

From the above motivations, the problem of computing drawings of graphs with labels on vertices and edges has been widely investigated in the graph drawing and computational geometry communities. Most of the results in the literature assume that the vertices and the edges of the graph have been already drawn, and concentrate only on the placement of the labels. See for example [7,8,9,12,13,15]. These assumptions however can represent a severe limitation to the readability of the final drawing in many applications. For example, in the drawing of UML diagrams and ER-diagrams, there are no motivations to consider the geometry of vertices and edges as predefined. In-

stead, the geometry of vertices and edges can be computed in such a way to free up space for the insertion of the labels.

In this paper we present a software system that computes labeled drawings of graphs in the orthogonal drawing convention. An *orthogonal drawing* is such that vertices are drawn as points or boxes of the plane and edges are drawn as sequences of vertical and horizontal segments between their end-vertices. Several kinds of diagrams used in the software engineering and information system areas can be represented by labeled orthogonal drawings. Namely, we are interested in those diagrams where vertices are represented as boxes that can host their associated labels (see for example Figure 1). Our system allows the computation of a labeled orthogonal drawing while minimizing the area or the total edge length of the drawing. Note that, the problems of computing labeled or unlabeled orthogonal drawings of minimum area or minimum total edge length have been shown to be NP-complete [14]. The drawing algorithms of our system are based on recent results presented in [2], in which, for the first time, strategies are provided for the computation of labeled and unlabeled orthogonal drawings with minimum area. Previous results concentrate on the minimization of total edge length and only deal with vertex labels [11,10]: also, vertices are always represented as points and their labels are placed around them. This model is not suitable for the diagrams we are interested in.

The paper is organized as follows. Section 2 recalls basic definitions on graph drawing and introduces the labeled orthogonal drawing standard adopted by our system. Section 3 describes the architecture and the functionalities of our system. Section 4 sketches some algorithmic ideas that is behind our system. Conclusions and open problems are given in Section 5.

2 Labeled Orthogonal Drawings

Let G be a graph. A *drawing* Γ of G maps each vertex v of G to a distinct point p_v of the plane and each edge $e = (u, v)$ of G to a simple Jordan curve of the plane between p_u and p_v: vertices p_u and p_v are called *end-points* of e. Drawing Γ is *planar* if no two edges intersect, except at common end-points. A graph is *planar* if it admits a planar drawing.

A *planar orthogonal drawing* G is a special planar drawing of G on the plane, where edges are drawn as sequences of horizontal and vertical segments. More formally, a planar orthogonal drawing of G is such that:

- Each vertex v of G is mapped to a distinct point p_v of an integer coordinates grid of the plane;
- Each edge (u, v) of G is mapped to a sequence of horizontal and vertical grid segments between p_u and p_v. We call these segments *edge-segments*.
- No two distinct edges intersect except at common end-points.

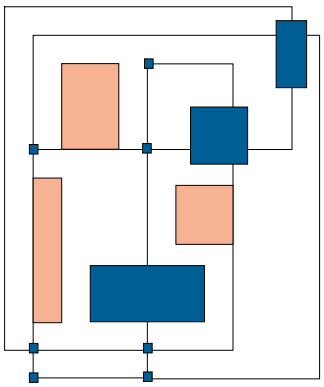

Fig. 2. A labeled planar orthogonal drawing. Red (light) boxes represent edge-labels. Blue (dark) boxes represent vertices.

A planar graph G admits a planar orthogonal drawing if and only if every vertex of G has degree at most four, where the degree of a vertex is the number of its adjacent vertices. Several extensions of the original definition of orthogonal drawing have been provided in the literature in order to deal with vertices of degree higher than four. These extensions use to represent vertices as boxes instead of points. Some applications allow boxes representing different vertices to have different sizes [16,1]; other applications require that all the boxes have the same size [3].

A class H of planar orthogonal drawings of G that differ only for the length of some edges is called a *planar orthogonal representation* of G. In other words, an orthogonal representation of G specifies:

- For each edge e of G, the sequence of left and right turns (*bends*) encountered along e moving from a predefined vertex of e;
- For each vertex v, the value of the angle formed by every pair of edges incident on v that are consecutive in the clockwise order around v.

A classical graph drawing problem is to determine a planar orthogonal drawing Γ of G within a given orthogonal representation of G, in such a way that either the area or the total edge length of Γ is minimized. Both the versions of this problem have been shown to be NP-complete [14].

We consider the problem of computing planar orthogonal drawings of $G = (V, E)$ with labels on edges and vertices, while minimizing the area or the total edge length of the drawing within a given orthogonal representation H of G. Each label is represented as a box with predefined height and width. We call *edge-label* and *vertex-label* a label of an edge and a label of a vertex, respectively. We suppose that each edge and each vertex can have only one associated label.

More formally, let L_E (L_V) be a set of edge-labels (vertex-labels) of G, each of them with associated height and width. A *labeled planar orthogonal drawing*

of (G, L_E, L_V) within an orthogonal representation H of G is a drawing Γ such that:

(i) G is drawn as a planar orthogonal drawing within H. The vertices of G that have no labels are mapped to grid points. Each vertex v that has a label λ_v is drawn as a box with the height and the width associated with λ_v: v does not intersect other vertices and edges, except its incident edges.

(ii) Each edge-label λ_e of e is drawn as a box that has the height and the width of λ_e and that has one side properly contained in a segment of e. Label λ_e does not intersect other edge-segments, vertices, and edge-labels.

Figure 2 shows an example of a labeled planar orthogonal drawing.

3 High Level Description of the System

The system we present computes a labeled orthogonal drawing of a graph G while minimizing the area or the total edge length of the drawing within a given orthogonal representation H of G.

The whole system integrates different software technologies. It uses the GDToolkit graph drawing library [5], the Microsoft Visual Basic programming language, the AMPL [4] language for mathematical programming, and the CPLEX solver [6]. The general architecture of the system is depicted in Figure 3.

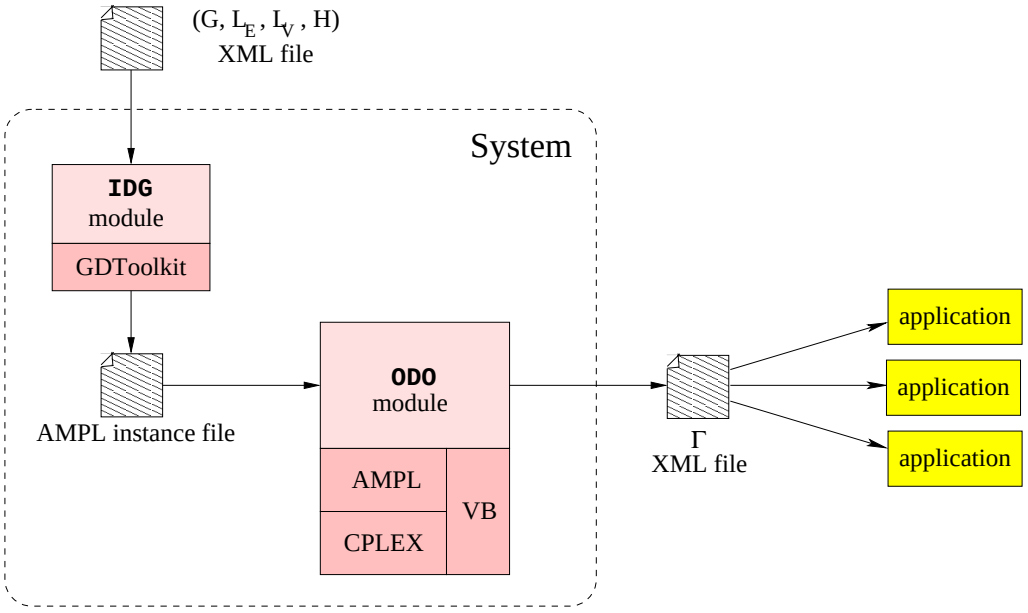

Fig. 3. The general architecture of the system.

The system computes labeled orthogonal drawings of minimum total edge

length based on an ILP (Integer Linear Programming) formulation. Some algorithmic ideas are given in Section 4. A more detailed description of the ILP formulation can be found in [2].

To deal with the ILP problem the system uses the AMPL language, which allows the programmer to keep distinct the description of the model and the description of the instances of the ILP problem. The model of the ILP problem is an abstract specification of the objective function, variables, and constraints of the ILP problem, and it is stored in a file with a specific file format. Each instance of the ILP problem describes concrete data according to a specific input labeled graph, and it is stored in a file with a format that is different from the ILP model file format. Different instances of the same ILP problem are stored in distinct files, but are described with respect to the same model. To run a computation of the ILP problem the system uses the well known CPLEX solver. AMPL provides the interface to access the CPLEX functionalities.

The main module of our system is the `ODO` module (*Orthogonal Drawing Optimizer* module). It offers a graphical user interface written in the Microsoft Visual Basic programming language. This interface allows the user to load an instance of the ILP problem (given in the AMPL instance file format) and to run a computation over this instance. The user can decide if he wants to perform the minimization of the total edge length or the minimization of the area. In the latter case, the whole execution consists of several computations of ILP problems that are slight variations of the ILP problem for total edge length. During the execution the current best drawing is repeatedly updated and shown on an area that is embedded in the graphical interface, and several statistics about the drawing quality (like area and total edge length) and about the performance of the computation (like CPU time and number of ILP computations) are shown in a distinct panel. The user can benefit from these real-time information for two main purposes: (i) He can use them to understand the behavior of CPLEX algorithms on different instances, and he can tune some algorithm parameters in order to make the computations faster. (ii) He can decide to prematurely stop the execution if the current best drawing already matches his aesthetic requirements. This is useful especially for long time computations.

At the end of an execution, the user can save the labeled drawing Γ in a specific XML file format, which can be handled by different real-world applications in the software engineering field. The XML format is that defined in the GDToolkit library. The `ODO` module offers also the option of automatically loading and running a block of instances and saving, in a unique file, the statistics of every execution. In this sense, the module can be used as a platform for experiments. Figure 4 shows a snapshot of the graphical interface of the `ODO` module.

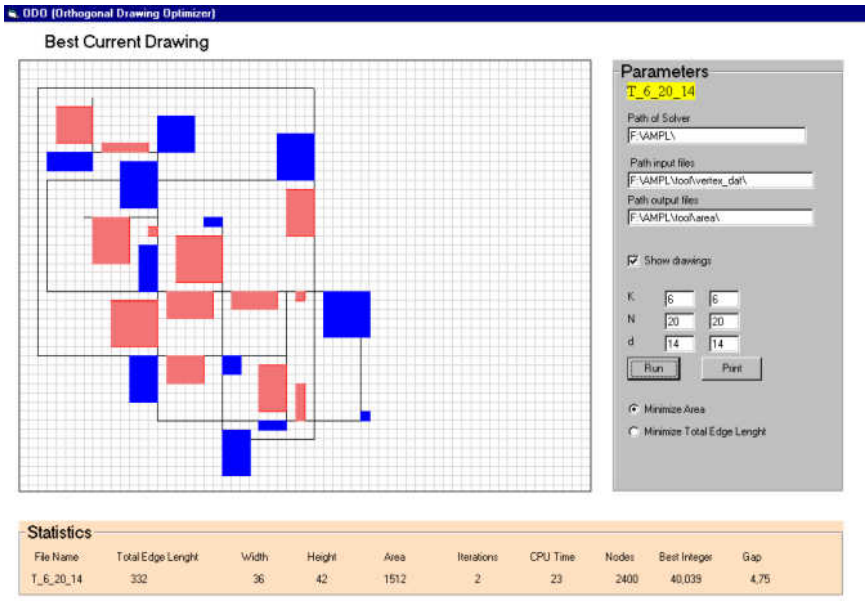

Fig. 4. A snapshot of the ODO module. In the orthogonal drawing, red (light) boxes represent edge-labels and blue (dark) boxes represent vertices.

The generation of the files that define the instances in the AMPL format is performed by the `IDG` module (*Instance Data Generation* module). This module is written in the C++ programming language and based on the GDToolkit graph drawing library. It receives as input a specific XML file describing a labeled graph (G, L_E, L_V) along with an orthogonal representation H of G, and takes advantage of the GDToolkit API to construct from the input information the description of the instance of the ILP problem in terms of variables and constraints data. The module is also able to compute an orthogonal representation of G if it is not provided one.

4 Algorithmic Aspects

Our system implements and/or integrates several algorithms both in the graph drawing area and in the optimization problems area. In this section we give some algorithmic ideas that is behind the system. A more detailed description of the algorithms and their experimental results can be found in [2].

The `IGD` module interacts with objects defined in the GDToolkit library for representing graphs and orthogonal representations of graphs. By querying these objects, the `IGD` module constructs auxiliary objects for representing segments that form edges, vertices, and labels. After that, these objects are translated into variables and constraints of the ILP problem. In particular, variables and constraints of the ILP problem represent all valid positions for the edge labels and all valid points on which edges can be incident on a vertex

represented as a box. Constraints of the ILP problem must also guarantee the planarity of the labeled drawing, in terms of edge, vertex, and label intersections.

The objective function of the ILP problem can be either the minimum total edge length of the drawing, i.e. the sum of the length of its edges, or the minimum width or height of the drawing. The `ODO` module is also able to compute a drawing of minimum area, by applying several times a modified version of the ILP problem, which allows us to compute a drawing of minimum width, under a certain constraint on the height. Namely, suppose that M is an upper bound on the height of a minimum area drawing. The algorithm for computing a drawing with minimum area works as follows. First, it computes a drawing Γ such that: (i) Γ has the minimum width among the drawings with height less than $M + 1$; (ii) Γ has the minimum height among the drawings with the same width as Γ. After that, the algorithm iteratively proceeds by looking for a new drawing with the same properties as Γ among those of height less than the height of Γ. The algorithm stops when it fails searching such a drawing. The upper bound M can be found easily. For example, M can be set equal to the number of horizontal segments of the orthogonal representation plus the heights of all labels. The algorithm above sketched can also be used, as a special case, for computing orthogonal drawings with minimum area of graphs without labels. Our system is the first that implements an algorithm for computing minimum area orthogonal drawings, both for the labels and for the unlabeled version of the problem.

Experiments performed with the system on a large test suite of labeled graphs have shown that the algorithm for the minimization of the area requires few seconds for graphs with number of vertices up to 20, and it takes some minutes for graphs with up to 40 vertices. Long time computations can be needed for graphs with more than 50 vertices. The experiments have been performed on a PC Pentium III, 800MHz, 512MB RAM capacity, Windows NT operating system, and CPLEX 7.1. Also, a comparison with previous known heuristics for the area minimization of labeled orthogonal drawings has shown that the minimum area can significantly improve the area of a drawing computed by the heuristics. The average improvements with respect to the best known heuristic are between 10% and 20%, depending on the density (number of edges on number of vertices) of the graphs.

5 Conclusions and Open Problems

We presented a system that computes orthogonal drawings with labels on vertices and edges, while minimizing the area or the total edge length of the drawing within a given orthogonal representation. Our system is based on an integer linear programming formulation presented in a previous work. How-

ever, this formulation is currently restricted to graphs with vertices of degree at most four. Since our system is thought to support CASE tools in the automatic visualization of diagrams, it is crucial to extend the ILP formulation to graphs with any vertex degree. Also, we are working to improve the efficiency of the algorithms for those graphs that have a high number of vertices.

Acknowledgments

We thank Giuseppe Liotta and Maddalena Nonato for their useful discussions.

References

[1] G. D. Battista, W. Didimo, M. Patrignani, and M. Pizzonia. Orthogonal and quasi-upward drawings with vertices of arbitrary sizes. In *Symposium on Graph Drawing (GD'99)*, volume 1731 of *LNCS*, pages 297–310, 2000.

[2] C. Binucci, W. Didimo, G. Liotta, and M. Nonato. Computing labeled orthogonal drawings. In *Symposium on Graph Drawing (GD'02)*. to appear.

[3] 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.

[4] R. Fourer, D. M. Gay, and B. W. Kernighan. *AMPL: A modeling Language For Mathematical Programming.* Duxbury Press / Brooks/Cole Publishing Company, 1993.

[5] GDToolkit. On line: Graph drawing toolkit: http://www.dia.uniroma3.it/∼gdt.

[6] ILOG CPLEX. http://www.ilog.com/products/cplex.

[7] K. G. Kakoulis and I. G. Tollis. On the edge label placement problem. In *Symposium on Graph Drawing (GD'96)*, volume 1190 of *LNCS*, pages 241–256, 1997.

[8] K. G. Kakoulis and I. G. Tollis. An algorithm for labeling edges of hierarchical drawings. In *Symposium on Graph Drawing (GD'97)*, volume 1353 of *LNCS*, pages 169–180, 1998.

[9] K. G. Kakoulis and I. G. Tollis. On the complexity of the edge label placement problem. *Computational Geometry: Theory and Applications*, 18:1–17, 2001.

[10] G. Klau and P. Mutzel. Combining graph labeling and compaction. In *Symposium on Graph Drawing (GD'99)*, LNCS, pages 27–37, 1999.

[11] G. Klau and P. Mutzel. Optimal compaction of orthogonal grid drawings. In *Integer Programming and Combinatorial Optimization (IPCO'99)*, volume 1610 of *LNCS*, pages 304–319, 1999.

[12] G. Klau and P. Mutzel. Optimal labelling of point features in the sliding model. In *In Proc. COCOON'00*, volume 1858 of *LNCS*, pages 340–350, 2001.

[13] S. Nakano, T. Nishizeki, T. Tokuyama, and S. Watanabe. Labeling points with rectangles of various shape. In *Symposium on Graph Drawing (GD'00)*, volume 1984 of *LNCS*, pages 91–102, 2001.

[14] M. Patrignani. On the complexity of orthogonal compaction. *Computational Geometry: Theory and Applications*, 19:47–67, 2001.

[15] T. Strijk and A. Wolff. *The map labeling bibliography.* on-line: http://www.math-inf.uni-greifswald.de/map-labeling/bibliography/.

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