GRID: An Interactive Tool for Computing Orthogonal Drawings with the Minimum Number of Bends *

Walter Didimo †
didimo@inf.uniroma3.it

Antonio Leonforte ‡
leonfort@inf.uniroma3.it

Abstract

In this paper we present a new interactive tool for computing orthogonal grid drawings of planar graphs. The tool is based on GDToolkit, an object-oriented library of classes for handling graphs and computing their layout. GDToolkit is built on LEDA (an efficient library of data types and algorithms) and currently implements three orthogonal layout methods. Especially, we provide a new branch-and-bound algorithm choosing a planar embedding in order to minimize the number of bends. The enumeration schema of the branch-and-bound algorithm is based on the GDToolkit SPQR-tree class (as far as we know, the only existing SPQR-tree implementation). The tool offers an interactive graphical interface to the branch-and-bound algorithm, which allows to edit the embedding, to execute the algorithm step by step and to view partial results. It also gives quality measures on the drawing, and quantitative measures on the algorithm's perfomance.

1 Introduction

The graph drawing research has received increasing attention recently. Variuos graphic standards have been proposed to draw graphs, in order to fit specific application fields. An extensive literature on the subject can be found in [2]. Several libraries of data structures and algorithms supporting graph drawing have been developed, e.g. LEDA [11], Tom Sawyer Graph Layout Toolkit [16] and GraphBase [10]. Many graph visualization and editing systems are also available, e.g. Diagram Server [3], daVinci [5], Graphlet [8], GIOTTO3D [7].

This paper is about *orthogonal grid drawing* of 4-planar graphs (i.e. planar graphs with degree of the vertices at most 4). An orthogonal grid drawing places

*Research supported in part by the ESPRIT LTR Project no. 20244 - ALCOM-IT.

†Dipartimento di Informatica e Automazione, Università di Roma Tre, via della Vasca Navale 84, 00146 Roma, Italy

‡Dipartimento di Informatica e Automazione, Università di Roma Tre, via della Vasca Navale 84, 00146 Roma, Italy

each node and each bend on a different point of the grid, (i.e. defines integer coordinates for each of them) and maps each edge into a chain of horizontal and vertical segments.

Orthogonal grid drawings are widely used for graph visualization in many applications, including database design (entity-relationship diagrams) software engineering (data flow-diagrams) and circuit design (circuit schematics).

A very elegant algorithm [14] allows to generate, in polynomial time, orthogonal grid drawings with the minimum number of bends of a planar graph, within the given embedding. Unfortunately, the bends-minimization problem over all the possible planar embeddings is NP-complete [6].

We propose an interactive tool, GRID, implementing a new algorithm for computing orthogonal drawings with the minimum number of bends. We call this algorithm *Slow Orthogonal*, due to exponential time needed by the computation. It is based on a branch-and-bound technique with an efficient enumeration schema of the planar embeddings and several new methods for comuputing lower bounds of the number of bends. Slow selects the *best* embeddings taking advantage of an SPQR-tree [4], a data-strucure representing the decomposition of the given graph into its triconnected components. As far as we know, GDToolkit (available at www.inf.uniroma3.it/people/gdb/wp12/) is the only library currently providing an SPQR-tree class. A complete description of the Slow Orthogonal algorithm, with experimental results, can be found in [1].

GRID also provides two classical algorithms [14, 15] which compute an orthogonal drawing preserving the given embedding.

The user can select the algorithm that is better suitable for his needs in terms of a compromise between effectiveness and efficiency or/and to compare the results produced by the Slow Orthogonal against those produced by classical algorithms. A complete description of the GRID tool is given in the next section.

2 GRID: Graph Interactive Drawer

In this section we describe the main features of GRID, along with some extensions we plan to implement in the near future.

The tool provides a user-friendly graph editor (Fig. 1). You can create/edit a multi-graph drawing, apply one of the available layout algorithms on the underlying graph, and edit again the resulting drawing. In particular, a new layout algorithm, *Slow Orthogonal*, is available in the tool, which allows to compute an orthogonal drawing with the minimum number of bends; all the tool's features with respect to this algorithm are described in the next subsection. In addition, GRID provides two classical layout algorithms: *Quick Orthogonal* computes an orthogonal layout of the graph, using a heuristic method to reduce the number of bends within the given embedding [15]. *Optimal Orthogonal* computes an orthogonal layout of the graph in polynomial time $O(n^2 \log n)$, minimizing the number of bends within the given embedding. We implemented a minimum cost flow algorithm searching minimum cost paths with the Dijkstra algorithm, as described in the original paper of Tamassia [14].

2.1 Features of the Slow Orthogonal algorithm

The Slow Orthogonal algorithm computes an orthogonal layout with the minimum number of bends by applying the Optimal Orthogonal algorithm while exploring all the relevant embeddings with a branch-and-bound approach. This algorithm takes advantage of new lower-bounds recently found [1] for the number of bends of an orthogonal drawing.

The tool offers several features you can use freely and interactively, according to the graph you want to draw/inspect, and to the goals you want to achieve. Neverthless, on a general basis, you are likely to cycle on a sequence of operations we describe hereunder.

Create a 4-planar biconnected graph (or load and edit one).

Decompose the graph into its triconnected components. Clicking the DECOMPOSE button, the tools prompts for a reference edge e, then performs a decomposition of the current graph into triconnecetd components. As a result of such a decomposition, an SPQR-tree is built and displayed on a dedicated window. Series, parallel, rigid and Q nodes are respectively drawn as triangles, ellipses, rectangles and circles. Q-nodes display can be disabled when their number makes the SPQR-tree drawing too wide.

Evaluate the graph structure by playing with the corresponding SPQR-tree. You can change the root of the SPQR-tree in, simply clicking at the button EVERT. You can display an orthogonal drawing of any S/P/Q/R component by clicking on the corresponding SPQR-tree node: a dedicated window pops up displaying the skeleton subgraph and highlighting with different colors the separation-pair (orange nodes), virtual (green) and non-virtual (black) edges (Fig. 3. a). You can cycle across all the possible embeddings of a given skeleton, with respect to its reference edge, and thus, changing the currently planar embedding of the graph. Please note that the skeleton of a rigid component has only two possible embeddings (flipping around the separation-pair), while the skeleton of a parallel component has a number of embeddings equal to the number of the possible permutations of its non-reference edges.

Optionally adjust the parameters affecting the Branch-and-Bound embedding search. You can adjust several parameters according to the graph structure, in order to reduce the search time. You can decide to compute the upper bounds in a non-repetitive way or in a random way, and you can set how to calculate the lower bounds: efficient lower bounds need more computation time, but they very often reduce the search domain more quickly. Please note that all these options can be changed even once the embedding search is started, by pausing it temporarily.

Start the Branch-and-Bound embedding search. Clicking the START button, the Branch-and-Bound embedding search starts. During the embedding search, the SPQR-tree drawing is updated to reflect each search step (animation and printouts, however, can be disabled to minimize the running time). A complete redraw is performed each time a new reference-edge is choosen. Node colors change to reflect the algorithm's embedding choices: each node is yellow, red or green when the embedding of the corresponding skeleton is respectively unkown,

fixed or under construction (Fig. 3. b). Given an edge e and its expansion graph G_e, the label of e reflects the preprocessing lower-bound available for the number of bends of any orthogonal drawing of G_e.

Optionally pause or stop the Branch-and-Bound embedding search. You can optionally pause the search in order to adjust some algorithm parameters. You can also execute the search algorithm step by step, or stop it definitively when you think the current orthogonal drawing is suitable.

Qualitative and quantitative measures. Once the Branch-and-Bound embedding search is completed, you can display the final orthogonal drawing, and evaluate the quality and performance measures provided by the tool. The quality measures available are: ***bends***: total number of bends; ***maxedgebends***: max number of bends on an edge; ***unifbends***: standard deviation of the number of edge bends; ***area***: area of the smallest rectangle with horizontal and vertical sides covering the drawing; ***totaledgelen***: total edge length; ***maxedgelen***: maximum length of an edge; ***uniflen***: standard deviation of the edge length; ***screenratio***: deviation of the ascpect ratio of the drawing (width/height or height/width) with respect to typical screen ratio (4/3). The performance measures are: cpu time, total number of search tree nodes, number of visited search tree nodes.

Experimenting GRID on a large number of reference graphs, we observed that the layouts computed by Slow Orthogonal tipically have a smaller area (and a smaller number of bends) with respect to the ones computed by Quick and Optimal. (Fig. 2 and Fig. 4).

2.2 Extensions

GRID layout algorithms currently run with 4-planar graphs only. We plan to make them run with nodes of any degree by using, for instance, the expansion techniques cited above. Moreover, the current implementation of the SPQR-trees only supports biconnected graphs; we plan to enhance the SPQR-tree class to make it handle any connected planar graph [4]. Slow Orthogonal, however, can be already applied on all the biconnected components of a graph; this yields a powerful heuristic for reducing the number of bends in connected graphs.

Akcnowledgements

We'd like to thank Paola Bertolazzi and Giuseppe Di Battista for their original ideas. We are also grateful to Sandra Follaro, Armando Parise and Maurizio Patrignani for their technical support.

References

[1] P. Bertolazzi, G. Di Battista and W. Didimo. Computing Orthogonal Drawing with the Minimum Number of Bends. In *Proc. Workshop Algorithms Data Struct.*, 1997 (to appear).

[2] G. Di Battista, P. Eades, R. Tamassia and I. G. Tollis. Algorithms for drawing graphs: an annotated bibliography. *Comput. Geom. Theory Appl.*, 4:235–282, 1994.

[3] G. Di Battista, G. Liotta, M. Strani and F. Vargiu. Diagram Server. *Proceedings of Advances Visual Interfaces*, 36:415-417, 1992.

[4] G. Di Battista and R. Tamassia. On-line planarity testing. *SIAM J. Comput.*, 25(5):956–997, 1996.

[5] M. Fröhlich and M.Werner. Demonstration of the Interactive Graph-Visualization System da Vinci. *Lecture Notes in Computer Science*, 894:266-269, 1994.

[6] A. Garg and R. Tamassia. On the computational complexity of upward and rectilinear planarity testing. Submitted to *SIAM Journal on Computing*, 1995.

[7] A. Garg and R. Tamassia. GIOTTO3D: A System for Visualizing Hierarchical Structures in 3D. In *Symposium Graph Drawing, GD'96, LNCS*, 1190, 1996

[8] M. Himsolt. The Graphlet System. In *Symposium Graph Drawing, GD'96, LNCS*, 1190, 1996

[9] J. Hopcroft and R. E. Tarjan. Dividing a graph into triconnected components. *SIAM J. Comput.*, 2:135–158, 1973.

[10] D.E. Knut. The Stanford GraphBase: a platform for combinatorial algorithms. Stanford University, 1993.

[11] K. Mehlhorn and S. Näher. LEDA: a platform for combinatorial and geometric computing. *Commun. ACM*, 38:96–102, 1995.

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

[13] A. Scott. A Survey of Graph Drawing Systems. *Technical Report 95-06.*, Department of Computer Science University of Newcastle, Australia, 1995.

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

[15] R. Tamassia and I. G. Tollis. Planar grid embedding in linear time. *IEEE Trans. on Circuits and Systems* CAS-36:1230–1234, 1987.

[16] Tom Sawyer Software. Tom Sawyer Graph Layout Toolkit. *Tom Sawyer Software Corporation*, 1824B Fourth STreet, Berkley, CA94710, USA.

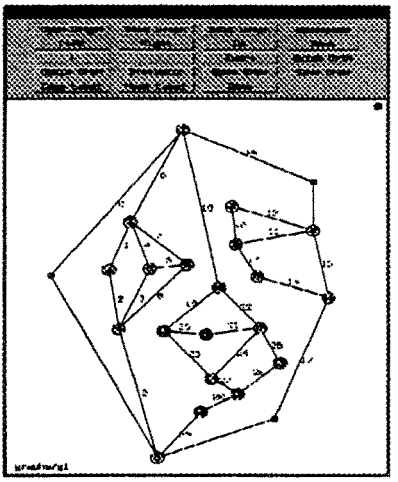
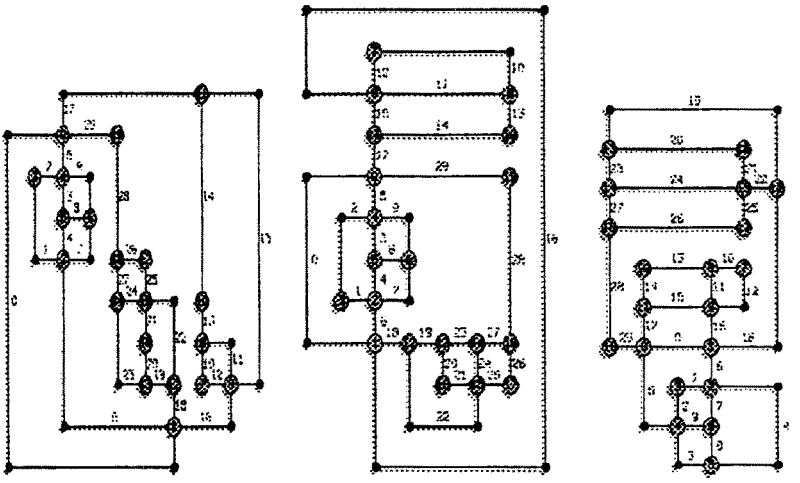
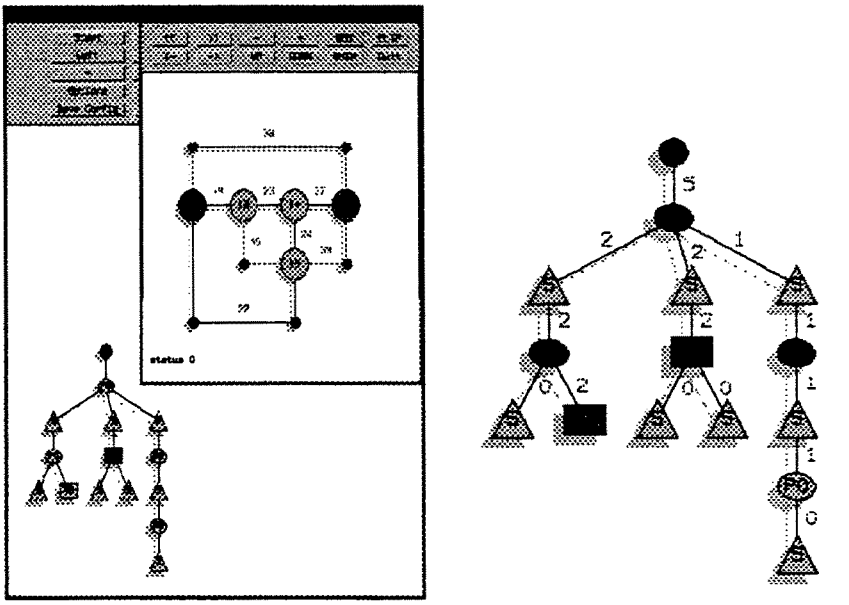


Figure 1: A 20-nodes graph created by the GRID editor



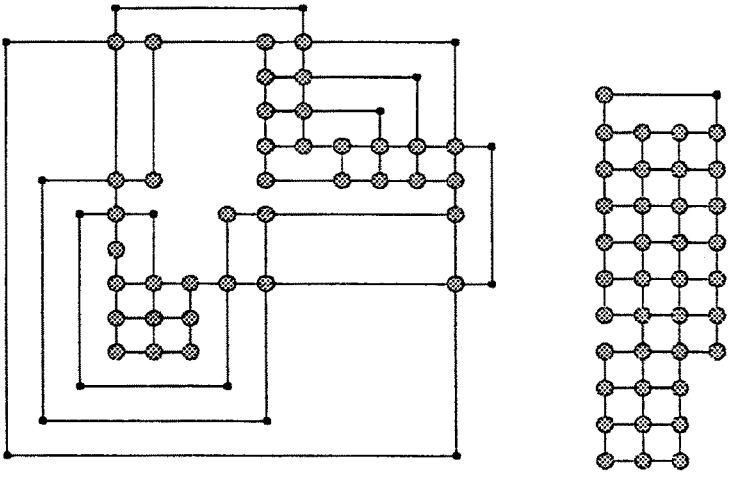
(a) 14 bends (b) 13 bends (c) 8 bends

Figure 2: Graph drawn by: (a) Quick Orthogonal, (b) Optimal Orthogonal, (c) Slow Orthogonal



(a) (b)

Figure 3: (a) An SPQR-tree with the skeleton of an R-node. (b) The same SPQR-tree during the computation



(a) 17 bends (b) 1 bend

Figure 4: Graph drawn by: (a) Optimal Orthogonal, (b) Slow Orthogonal