Drawing Relational Schemas *

Giuseppe Di Battista, Walter Didimo,
Maurizio Patrignani, and Maurizio Pizzonia
{gdb,didimo,patrigna,pizzonia}@dia.uniroma3.it

Dipartimento di Informatica e Automazione, Università di Roma Tre,
Via della Vasca Navale 79, 00146 Roma, Italy.

Abstract. A wide number of practical applications would benefit from automatically generated graphical representations of relational schemas, in which tables are represented by boxes, and table attributes correspond to distinct stripes inside each table. Links, connecting two attributes of two different tables, represent relational constraits or join paths, and may attach arbitrarily to the left or to the right side of the stripes representing the attributes. To our knowledge no drawing technique is available to automatically produce diagrams in such strongly constrained drawing convention. In this paper we provide a polynomial time algorithm solving this problem and test its efficiency and effectiveness against a large test suite.

1 Introduction

The tasks of designing, maintaining, updating, and querying databases require users and administrators to cope with the complexity of the relational schemas describing the structure of the data. A graphical representation of such schemas greatly improves the friendliness of database applications and it is essential for producing high-quality understandable documentation. For this reason many commercial tools have some diagramming facilities (see Figure 1 for an example) that rely on the user to nicely place tables and their relationships on the screen. However, drawing diagrams manually is time consuming and the aesthetic results are often unsatisfactory.

Unfortunately, to our knowledge, no drawing technique is available to automatically produce high quality diagrams of this kind. In fact, such diagrams are strongly constrained: each table of the relational schema is usually represented by a box composed by a vertically ordered sequence of attributes, topped by the table name. Edges represent constraints or join paths between tables. An edge linking an attribute of one table to an attribute of a different table may attach arbitrarily to the left side or to the right side of the boxes, and should incide the box at the level of the attribute name.

Actually, even if the link between the database research area and the graph drawing one is strong, the interest has been so far mainly focused on the visualization of Entity-Relationship diagrams and Data-Flow diagrams, that are relatively simpler to draw automatically than the relational schema diagrams (see, e.g. [5, 3, 11]).

The results presented in this paper can be summarized as follows: (i) We formulate the problem of automatically generating relational schema diagrams as a constrained

* Work partially supported by: “Progetto Algoritmi per Grandi Insiemi di Dati: Scienza e Ingegneria”, MURST Programmi di Ricerca di Rilevante Interesse Nazionale.

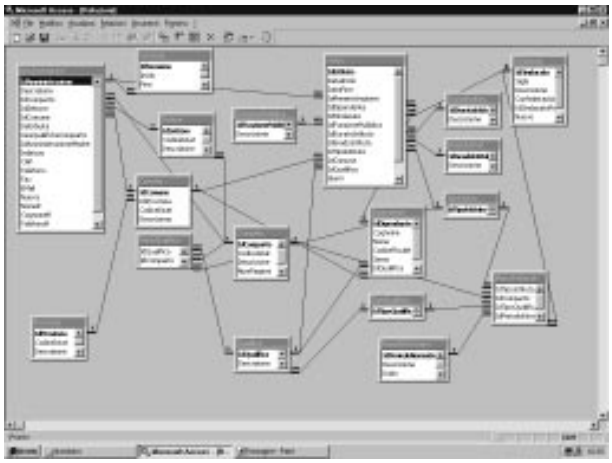


Fig. 1. A screen snapshot of Microsoft Access©. The example is taken from a real life application. Lines represent referential integrity constraints (see Section 2).

orthogonal graph drawing problem, and we address it within the "classical" topology-shape-metric approach [17, 18, 11], showing how this approach can be tailored to take into account the complex constraints originated by this type of diagrams (Section 3). (ii) We give a polynomial time algorithm for constructing relational schema diagrams. The algorithm relies on several variations of existing graph drawing techniques, giving new highlights on their practical applicability (Section 3). (iii) We present an implementation of the algorithm and show its efficiency and effectiveness performing an experimental test with a random generated test suite (Section 4). Basic definitions and background are given in Section 2 and open problems are outlined in Section 5.

2 Background

Since our target is to pictorially show only few, well determined, features of database schemas, we give a simplified model of them. We call *table* an ordered set of named *attributes*. We call *relational schema* a set of named tables and a set of pairs of attributes, called *links*. In our model tables and attributes are in one-to-one correspondence with the homonymous concepts of the database research field while the concept of link is new.

A link is an abstract placeholder. Its purpose it to represent *join paths* and/or *referential integrity constraints*. A join path is a frequently used join operation between two tables, based on the equality of the two attributes (represented in our model as the extremes of the link). A referential integrity constraint states that the legal values for a given attribute a are the values that appear in the key k of specified table (represented in our model as a link between a and k). Further details may be found in [2]. From the point of view of our algorithm it is not relevant to distinguish the two cases.

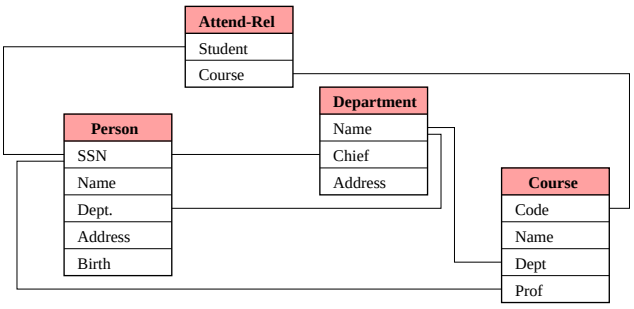


Fig. 2. An example of relational schema with 4 tables and 6 links. Links represent referential integrity.

We consider pictorial representations of a relational schema (see Figure 2) with the following properties: (a) Each table is represented as a box and its attributes are sequentially listed in the box. Each attribute corresponds to a horizontal stripe of the box. The top stripe of each table is reserved for its name. All stripes have the same height. (b) Each link (a, b) is represented as a polygonal line between the boxes of the two tables containing attributes a and b, respectively. The polygonal line incides the extremal boxes at the heights of the stripes associated with a and b, and all its segments are either horizontal or vertical (*orthogonal standard*).

We call `RO-drawing` (*Relational-schema Orthogonal drawing*) a layout of a relational schema that respects the above properties.

We now recall basic graph drawing definitions. We assume some familiarity with graph theory and connectivity [13].

A *plane drawing* Γ of a graph G maps each vertex of G into a point of the plane, and each edge of G into a Jordan curve between the two points associated with the end-vertices of the edge. A drawing Γ of G is *planar* if it does not contain crossings between edges. A graph is *planar* if it admits a planar drawing. A planar drawing Γ of G induces for each vertex v of G a circular clockwise ordering of the edges incident on v. Two planar drawings of G are said to be *equivalent* if for each vertex v of G they induce the same ordering of the edges around v. An *embedding* ϕ of G is a class of equivalent planar drawings of G. In other words, we can regard an embedding of G as the choice of a clockwise ordering of the edges around every vertex. An *embedded graph* G_ϕ is a planar graph G with a given embedding ϕ.

An *orthogonal drawing* of G is a plane drawing of G such that all edges are represented as chains of horizontal and vertical segments. An *orthogonal representation* (or *shape*) of G is an equivalence class of planar orthogonal drawings such that all the drawings of the class: (i) have the same sequence of left and right turns (*bends*) along the edges, and (ii) two edges incident at a common vertex determine the same angle. Roughly speaking, an orthogonal representation defines a class of orthogonal drawings that may differ only for the length of the segments of the edges.

One of the most popular technique for computing orthogonal drawings of a graph G is the so called *topology-shape-metrics* approach [4, 17, 11]. It consists of three consecutive steps:

Topology: In this step a topology for G is computed. Namely, if G is planar an embedding ϕ of G is determined in linear time, by applying a well-known planarity testing algorithm [14, 10]. If G is not planar, an embedding can be computed for it by adding a minimal number of dummy vertices to replace crossings. Such operation is usually called *planarization*. The number of crossings depends on the planarization technique, and it may be $\Omega(n^4)$. However, in practice this number is usually much smaller. For a survey on planarization techniques see [11].

Shape: During this step, an orthogonal representation H of G_ϕ is computed within the embedding ϕ. A famous algorithm for constructing an orthogonal representation of an embedded graph with vertices having at most four incident edges is presented in a work by Tamassia [17]. Such algorithm computes an orthogonal drawing that has the minimum number of bends within the given embedding. Extensions of Tamassia's algorithm to general embedded graphs are provided in [18, 12].

Metrics: In this step a final geometry for H is determined. Namely, a *compaction* algorithm assigns coordinates to vertices and bends of H with the purpose of reducing as much as possible the area (or the total edge length) of the final drawing, while preserving its planarity [11].

The Topology-Shape-Metrics approach allows us to deal with topology, shape, and geometry of the drawing separately, so simplifying the whole drawing problem. However, decisions taken in early steps cannot be changed, thus overall optimization is not achieved in general. For instance, introducing cross vertices forces crossings to appear on specific edge pairs, thus the total number of bends may be not optimal.

3 RO-Algorithm

In this section we describe a polynomial time algorithm for computing drawings of relational schemas within the `RO-drawing` convention described in Section 2. The algorithm is based on the topology-shape-metrics approach and exploits and modifies several existing graph drawing techniques. In particular, it makes careful use of a constrained planarization technique and of a variation of the algorithm in [6] for computing an orthogonal drawing of the relational schema.

Let S be a relational schema. The *underlying* graph of S is the graph G_S whose vertices are the tables of S and whose edges are the links of S. We say that S is *connected* when G_S is connected. We assume that S is always connected. If S is not connected we can apply the algorithm we describe hereunder to every connected component, and then arrange all obtained drawings on the plane by using any packing heuristics [9, 15].

The RO-Algorithm consists of three main steps:

Constrained Planarization A planarization is performed on the underlying graph G_S of S. The purpose of this step is to obtain a planar embedding of G_S such that the order of the edges around each vertex v_T, representing a table T, is compatible with the drawing standard described in Section 2 and the specific sequence of attributes of T. The output of this step is an embedded graph G'_S where dummy vertices of degree four

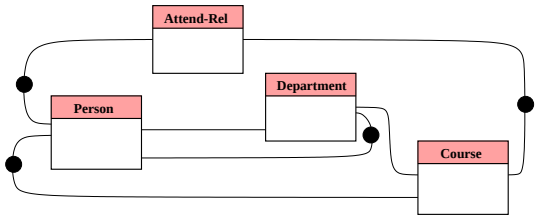


Fig. 3. Four u-turns in a partially computed RO-drawing, the round vertices are u-vertices.

are introduced to replace crossings (*cross-vertices*). Each link of S is represented in G'_S as an alternating chain of edges and cross-vertices.

U-Turns Assignment This step deals with the left-to-right development of the drawing. From this perspective the edges can be classified into two types: Edges that monotonically follow the left-to-right direction and edges that have to perform one or more "u-turn". In this step a (possibly empty) sequence of u-turns is associated with each edge trying to minimize their total number. U-turns are represented in G'_S with a particular kind of dummy vertices (*u-vertices*) of degree two that split the edges (see Figure 3).

Orthogonalization For each vertex of G'_S a pattern, among the ones depicted in Figure 4, is applied according to the type of the vertex. Once all vertices of G'_S have been considered, an appropriate sequence of 90 degrees bends (left or right) is associated with each edge, so describing an orthogonal representation H. To obtain the final RO-drawing from H the length of the edges and the size of the vertices are computed, heuristically "minimizing" the total edge length, and avoiding overlaps. Finally, cross-vertices and u-vertices are removed so that each link is again represented by exactly one edge.

3.1 Constrained Planarization

In the Constrained Planarization step, a planarization is performed on the underlying graph G_S of S. Let v_T be a vertex representing table T in G_S and let $a_1, \ldots, a_k$ be the attributes of T. Our algorithm partitions the edges incident on v_T into $2k$ possibly empty sets $l_1, \ldots, l_k, r_1, \ldots, r_k$, where the edges of $l_i \cup r_i$ represent the links incident

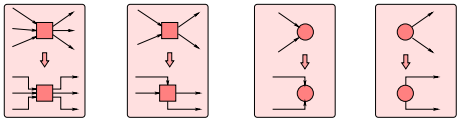


Fig. 4. Patterns for translating the description of a graph with cross-vertices and u-vertices into an orthogonal representation. (a) Vertex representing a table; (b) Cross-vertex; (c,d) u-vertices.

on attribute a_i. Edges of l_i (r_i) will enter v_T from the left (right) in the final drawing. An edge of l_i (r_i) is a *left edge* (*right edge*) for v_T. We aim at computing a planarization of G_S with the following constraints for each v_T: (i) The edges of the same set should appear contiguously in the circular order around v_T; (ii) Sets $l_1, \ldots, l_k, r_k, \ldots, r_1$ should appear in this counter-clockwise order around v_T.

We solve the above problem carefully exploiting a constrained planarization technique that allows to specify a set of uncrossable edges. Primitives of this kind are available, for instance, within the GDToolkit library [1].

Namely, Graph G_S is mapped into a new graph P_S in which each vertex v_T, associated with a table T with k attributes, is represented by a $(k+2)$-vertex path whose vertices and edges are called τ-*vertices* and τ-*edges*, respectively. The vertices of the path are $\{v_{\text{north}}, v_1, \ldots, v_k, v_{\text{south}}\}$, where v_i is associated with attribute a_i ($i = 1, \ldots, k$). The edges of the path are $(v_{\text{north}}, v_1), (v_1, v_2), \ldots, (v_k, v_{\text{south}})$. The edges representing links incident on attribute a_i are made incident on v_i. Intuitively, τ-vertices and τ-edges represent the structure of a table, and vertices v_{north} and v_{south} represent the upper and bottom part of the table, respectively.

Now, we run a planarization on P_S with the constraint that every τ-edge in P_S is uncrossable. Intuitively, this is done to have no edges that intersect tables in the final drawing. After the planarization, a contraction operation is applied to all the τ-vertices and τ-edges associated with the same table. The result of this phase is an embedded graph G'_S whose vertices may either represent a table or a cross. It is possible to show that the constraints described at the beginning of the subsection are enforced.

3.2 U-Turns Assignment

This step associates a (possibly empty) sequence of u-vertices with each edge of G'_S. A two phases procedure is adopted.

First, we assign an orientation to the edges. Such orientation describes the left-to-right development of the drawing. Consider edge $e = (v_{T'}, v_{T''})$, where none of $v_{T'}$, $v_{T''}$ is a cross vertex. Four cases are possible. If e is a right edge for $v_{T'}$ and a left edge for $v_{T''}$, then e is oriented from $v_{T'}$ to $v_{T''}$. If e is a left edge for $v_{T'}$ and a right edge for $v_{T''}$, then e is oriented from $v_{T''}$ to $v_{T'}$. If e is a right edge for $v_{T'}$ and a right edge for $v_{T''}$, then e is split into two edges both oriented outgoing from $v_{T'}$ and $v_{T''}$. If e is a left edge for $v_{T'}$ and a left edge for $v_{T''}$, then e is split into two edges both oriented incoming in $v_{T'}$ and $v_{T''}$.

A special technique is used for orienting the edges around cross-vertices imposing two of the incident edges to be ingoing and the other two outgoing. Orientation is then propagated from the vertices representing tables through the cross-vertices (possibly inserting u-vertices to avoid conflicts) until all edges are oriented.

Second, consider the obtained orientation and the inserted u-vertices. Two cases are possible. Either the embedded directed graph can be drawn left-to-right within the given embedding and with edges that monotonically follow the left-to-right orientation or not. In the first case we just go to the Orthogonalization step. In the second case we insert into G'_S the minimum number of u-turns that are needed to do that. Such a problem has been studied in [8] and can be solved in polynomial time using the flow techniques described in that paper.

3.3 Orthogonalization

The output of the U-Turns Assignment Step is an embedded directed graph with some dummy vertices called cross-vertices and u-vertices. Further, such a directed graph is drawable monotonically in the left-to-right direction preserving the embedding. To compute an orthogonal representation of G'_S we first draw it monotonically with the technique shown in [8] and then apply the patterns depicted in Figure 4, obtaining an orthogonal representation.

Once an orthogonal representation is constructed we shrink it in a limited area by using a variation of the compaction technique shown in [6]. Such technique allows to assign to each vertex exactly the required size and to arrange the edges to incide at the right height.

3.4 Time Complexity

The following result summarizes the analysis of the computational complexity:

Theorem 1 *Given a relational schema with n tables, m links, and a bounded number of attributes per table, RO-Algorithm takes $O((n+c)^2 \log(n+c))$, where c is the number of crossings of the output drawing.*

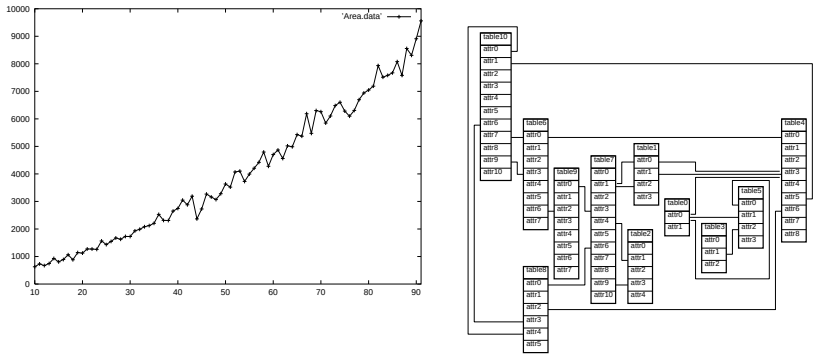
Proof. (sketch) The Constrained Planarization step takes $O(m(n+c))$ time, because it executes $O(m)$ times a breadth first search algorithm for computing shortest paths, as explained in [11]. After this step, the number of vertices of the graph is $N = n + c$. The U-Turn Assignment step takes $O(N)$ time to make G'_S oriented. The application of the flow technique described in [8] for inserting the minimum number of u-turns takes $O(N^2 \log N)$ time. The Orthogonalization step takes $O(N)$ time to produce the orthogonal representation of the planarized G'_S and $O(N^2 \log N)$ time to compact the drawing by using flow techniques, as described in [11, 6]. Hence the statement follows. □

Note that even if c may be $\Omega(n^4)$, relational schemas observed in practice are quite sparse ($m = O(n)$), and "almost planar", so c is much smaller.

4 Implementation and Experiments

We implemented the algorithm for computing `RO-drawings`, described in Section 3. The implementation is written in C++ and uses the GDToolkit graph drawing library (*http://www.dia.uniroma3.it/∼gdt*) which is based on LEDA [16].

In order to evaluate the effectiveness of the algorithm, we tested it over a set of 900 randomly generated relational schemas, with up to 90 tables. Namely, for each fixed number n of tables in the range 10–90, we considered 10 different relational schemas; each one of these schemas has been generated as follows: We denote the tables of the schema by $T_1, \ldots, T_n$. For each table T_i $(i = 1, \ldots, n)$, we randomly chose the number k_i of attributes of T_i, and sequentially enumerate them. The choice of k_i is done with a uniform probability distribution in the range $1 - 10$. We denote by $A_i = \{a_{i1}, \ldots, a_{ik_i}\}$



(a) (b)

Fig. 5. (a) Chart of the area occupied by the drawings of the schemas of the test suite with respect to the number of tables (x-asis). (b) A drawing of a relational schema of the test suite.

the set of the enumerated attributes of T_i. At the general step of the algorithm we randomly select two distinct tables T_i and T_j $(i, j \in \{1, .., n\})$, and two their attributes a_{ir}, a_{js}, where $r \in \{1, \ldots, k_i\}$ and $s \in \{1, \ldots, k_j\}$. We add the link (a_{ir}, a_{js}) to the relational schema. We add a total number of links that is randomly chosen in the range n–$2n$. When all links have been added, we check if the relational schema is connected. If it is not connected we discard the schema and restart the generation all over again, and so until a connected schema is obtained.

From the experiments, the space occupied by the drawings appears to increase quadratically with the number of tables, which is coherent with the results of other experiments in previous works [6]. In Figure 5 (a) the chart of the total area occupied by the drawings of the schemas of the test suite is shown. In Figure 5 (b) a drawing of a relational schema of the test suite is put in evidence. In Figure 6 it is shown an example of `RO-drawing` of a relational schema with 18 vertices taken from real life.

5 Conclusions and Open Problems

We have presented an algorithm for automatically drawing diagrams representing relational database schemas. We have also implemented and experimented such algorithm on a test suite of randomly generated relational schemas. Several problems remain open: (a) We are currently integrating the implementation within the Microsoft OLE platform. We would like to set up a web service that allows any user to exploit ability of our algorithm in drawing relational schemas. The user will be free to adopt several types of interchange formats. (b) Some more fine tuning could be performed on the algorithm. We plan to do it by further enriching the test suite. (c) It would be interesting to understand how to modify the algorithm if the attributes of the tables can be arbitrarily

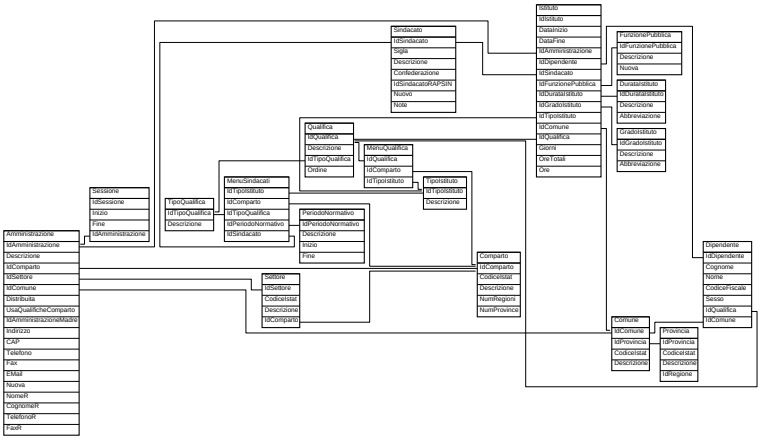


Fig. 6. A drawing of a relational schema, from real life, computed by our implementation of `RO-Algorithm`.

permuted in such a way to improve the aesthetic quality of the final drawing. (d) We plan to modify the technique described in this paper in order to obtain drawings of similar widely employed diagram standards (as, for example, UML diagrams) .

References

1. Gdtoolkit: An object-oriented library for handling and drawing graphs, 1999. Third University of Rome, http://www.dia.uniroma3.it/∼gdt.
2. P. Atzeni, S. Ceri, S. Paraboschi, and R. Torlone. *Database Systems: Concepts, Languages and Architetures*. McGraw Hill, London, United Kingdom, 1999.
3. C. Batini, E. Nardelli, M. Talamo, and R. Tamassia. GINCOD: a graphical tool for conceptual design of data base applications. In A. Albano, V. D. Antonellis, and A. D. Leva, editors, *Computer Aided Data Base Design*, pages 33–51. North-Holland, New York, NY, 1985.
4. C. Batini, E. Nardelli, and R. Tamassia. A layout algorithm for data flow diagrams. *IEEE Trans. Softw. Eng.*, SE-12(4):538–546, 1986.
5. C. Batini, M. Talamo, and R. Tamassia. Computer aided layout of entity-relationship diagrams. *Journal of Systems and Software*, 4:163–173, 1984.
6. G. D. Battista, W. Didimo, M. Patrignani, and M. Pizzonia. Orthogonal and quasi-upward drawings with vertices of arbitrary size. In J. Kratochvil, editor, *Graph Drawing (Proc. GD '99)*, Lecture Notes Comput. Sci. Springer-Verlag, 1999. to appear.
7. G. D. Battista, S. Diglio, M. Lenti, and M. Simoncelli. Queryviewer: A java system for drawing the result of a query, 1998. Third University of Rome, http://www.dia.uniroma3.it/∼lenti/QueryViewer/.
8. 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.
9. B. Chazelle. The bottom-left bin-packing heuristic: an efficient implementation. *IEEE Trans. Comput.*, C-32:697–707, 1983.

10. N. Chiba, T. Nishizeki, S. Abe, and T. Ozawa. A linear algorithm for embedding planar graphs using PQ-trees. *J. Comput. Syst. Sci.*, 30(1):54–76, 1985.
11. G. Di Battista, P. Eades, R. Tamassia, and I. G. Tollis. *Graph Drawing*. Prentice Hall, Upper Saddle River, NJ, 1999.
12. 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.
13. F. Harary. *Graph Theory*. Addison-Wesley, Reading, MA, 1972.
14. J. Hopcroft and R. E. Tarjan. Efficient planarity testing. *J. ACM*, 21(4):549–568, 1974.
15. K. Jansen. An approximation scheme for bin packing with conflicts. In *Proc. 6th Scand. Workshop Algorithm Theory*, volume 1432 of *Lecture Notes Comput. Sci.*, pages 35–46. Springer-Verlag, 1998.
16. K. Mehlhorn and S. Näher. LEDA: a platform for combinatorial and geometric computing. *Commun. ACM*, 38(1):96–102, 1995.
17. R. Tamassia. On embedding a graph in the grid with the minimum number of bends. *SIAM J. Comput.*, 16(3):421–444, 1987.
18. 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.