Automatic Graph Drawing and Readability of Diagrams

ROBERTO TAMASSIA, GIUSEPPE DI BATTISTA, AND CARLO BATINI

Abstract—**Diagrams are widely used in several areas of computer science, and their effectiveness is thoroughly recognized. One of the main qualities requested for them is readability; this is especially, but not exclusively, true in the area of information systems, where diagrams are used to model data and functions of the application. Up to now, diagrams have been produced manually or with the aid of a graphic editor; in both cases placement of symbols and routing of connections are under responsibility of the designer. The goal of the work is to investigate how readability of diagrams can be achieved by means of automatic tools. Existing results in the literature are compared, and a comprehensive algorithmic approach to the problem is proposed. The algorithm presented draws graphs on a grid and is suitable for both undirected graphs and mixed graphs that contain as subgraphs hierarchic structures. Finally, several applications of a graphic tool that embodies the aforementioned facility are shown.**

I. Introduction

Models to represent reality are used in many fields of computer science. To increase their expressiveness, they are often provided with diagrammatic (or iconic) representations. When diagrams are used to represent a piece of reality, the main quality desired for them is readability, where we say that a diagram is *readable* if its meaning is easily captured by the way it is drawn. The aim of this paper is to investigate how readability of diagrams can be achieved by means of automatic tools; namely, we compare existing results in the literature and propose a comprehensive approach to the problem.

Automatic drawing of readable diagrams has interesting applications in several fields. To highlight the advantages, we will focus on the design and production phases of the information systems life cycle; however, the following discussion can be applied to many other environments, e.g., CAD/CAM and project management.

In the design phase of an information system, diagrams are an effective documentation means and represent for both the designer and the user a common language to express the requirements of the application in a formal way. In the production phase, diagrams provide the user with a friendly interface to express queries and manipulate objects of the database. Table I shows examples of diagrams used in the information systems life cycle.

Manuscript received February 17, 1987; revised July 1987. This work was supported in part by the Instituto Centrale di Statistica (ISTAT) and the Progetto Finalizzato Trasporti, CNR. This paper was presented in part at the International Workshop on Software Engineering Environment, Beijing, China, August 18–20, 1986.

R. Tamassia is with the Coordinated Science Laboratory, University of Illinois at Urbana-Champaign, 1101 W. Springfield, Urbana, IL 61801.

G. Di Battista and C. Batini are with the Dipartimento di Informatica e Sistemistica, Universitá degli Studi di Roma "La Sapienza," Via Buonarroti 12, 00185 Rome, Italy.

IEEE Log Number 8718010.

TABLE I
Examples of Information System Diagrams

Life Cycle Phase	Diagrammatic Representation
Feasibility study	PERT
Functional analysis	data-flow diagrams, SADT
Conceptual data design	entity-relationship diagrams
Functional design	Jackson diagrams
Logical data design	DBMS models diagrams

Several design and documentation tools have been developed in the last years [13], [32], [46]; they assist the designer in describing a rich set of characteristics of the information system, discovering inconsistencies in the documentation, and producing various types of reports. Some of these tools are provided with a graphic editor; see, for instance, [2], [9], [24], [45].

Typically, the aforementioned editors present several limitations. First, they have graphic primitives that allow only simple operations, like creating and deleting symbols and connections. Instead, the use of diagrams for design purposes needs more complex manipulation capabilities. Examples of useful primitives are:

1) expanding symbols corresponding to concepts into more refined structures;
2) integrating intermediate products of the design activity; and
3) restructuring draft versions to improve their quality (e.g., completeness, correctness, and minimality).

A second limitation arises when the information system is described by means of both text-based and graphic-based tools. In this case, when some part of the textual documentation is changed, no automatic system is usually provided to keep the graphic documentation consistent (see Fig. 1).

Clearly, in the previous situations the layout of the new drawings (placement of symbols and routing of connections) is under responsibility of the designer, and achieving readability is also left to the designer. As a consequence, producing "good" diagrams is often a time-consuming and neglected activity.

The limitations of graphic editors can be overcome only by providing an automatic layout capability. The use of such capability can dramatically decrease the cost of the graphic documentation and provide an effective integra-

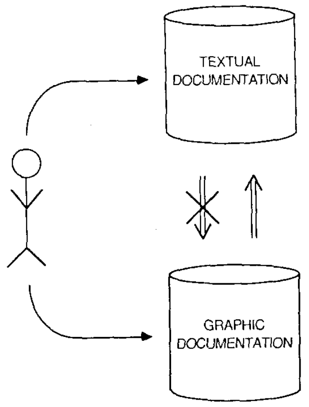


Fig. 1. Flows between textual and graphical documentation in existing tools.

tion of the phases of conception and production of diagrams.

The problem of obtaining drawings where the main quality is readability has recently gained increasing interest [6], [7], [37], [40]. The mathematical model common to all approaches consists of finding an optimal embedding in the plane of a labeled graph, where optimality criteria regard the readability of the diagram. In the following, we refer to this general problem as *automatic graph drawing*. Besides readability, layout algorithms have been investigated in several other fields, with the goal of minimizing some given resource. For example, in integrated circuit design the resource is the circuit area [29], [39]. Since algorithms for circuit layout produce inaesthetic crowding of connections, they are not well suited for information systems diagrams. Also, structural characteristics and constraints are quite different in the two cases.

This paper is organized as follows. In Section II we address the problem of automatic graph drawing and we survey state-of-the-art algorithms presented in the literature. Each algorithm deals with a specific *class of graphs* and adopts a given *graphic standard*, i.e., graphic rules to draw symbols and connections. Furthermore, each algorithm takes into account a specific set of *aesthetics* that express the readability criteria adopted for the drawing. In addition, we consider *computational complexity* and *implementation issues*. These concepts are introduced and discussed to provide a general framework in which the problem of automatic graph drawing can be better understood and to establish a basis of comparison for the algorithms surveyed.

In Section III we introduce a comprehensive approach to automatic graph drawing and propose an algorithm general enough to cover a wide range of applications. This approach is based on a well-known strategy in computer science, namely, to proceed by abstraction levels. In the algorithm the layout process is performed by choosing first the topology of the graph (considered as the more abstract property), then its shape, and finally its metrics. The algorithm, extending previous results presented in [3], [6], has the following innovative features.

1) It has the capability of taking into account *constraints* on the layout provided by the user as additional input. For example, the user can specify a set of vertices to be placed on the external boundary of the drawing, or a fixed shape for a piece of the drawing.

2) It also has several levels of performance/runtime trade-off. For example, very large graphs can be drawn with a fast heuristics, while for small graphs exact optimization procedures can be used.

The algorithm has been implemented and is currently part of a graphic tool for the design and documentation of information systems. Several applications of this tool are discussed in Section IV. Specifically, we show its use both as a self-standing documentation tool and in an interactive design environment. Finally, in Section V we examine future research directions.

II. The State of The Art In Automatic Graph Drawing

A. Basic Concepts

In this section we survey state-of-the-art algorithms for producing aesthetically pleasing drawings of graphs. The terminology we use about graphs can be found in many textbooks; see, for example, [19]. The number of vertices and edges of the graph are denoted by n and m, respectively. We categorize algorithms with respect to the following parameters:

1) class of graphs;
2) graphic standard;
3) aesthetics;
4) constraints; and
5) computational complexity.

Class of Graphs: Each algorithm is usually targeted to a specific class of graphs, e.g., trees, planar graphs, directed, or undirected graphs. In the next subsections the algorithms surveyed will be grouped according to this parameter.

Graphic Standard: The simplest way to draw a graph in the plane consists of placing first the vertices, and then drawing the edges as straight-line segments. This is called the *straight-line standard* (see Fig. 2). Another widely used graphic standard, called *grid standard*, consists of embedding the graph in a rectangular grid so that the vertices are placed at grid nodes, and the edges follow the horizontal and vertical tracks of the grid (see Fig. 3). Treelike diagrams are usually drawn according to the straight-line standard. When the diagrams are of a more general type, as in [21], [36], the grid standard is usually preferred because of its regularity and modularity. Other standards are generally variations of the two aforementioned ones. For example, several algorithms adopt a *mixed standard*, where vertices are placed at grid nodes, and edges are drawn as polygonal chains that bend only at grid nodes.

Aesthetics: We use the term *aesthetics* to denote the criteria that concern graphic aspects of readability. A well-admitted aesthetic, valid independently from the

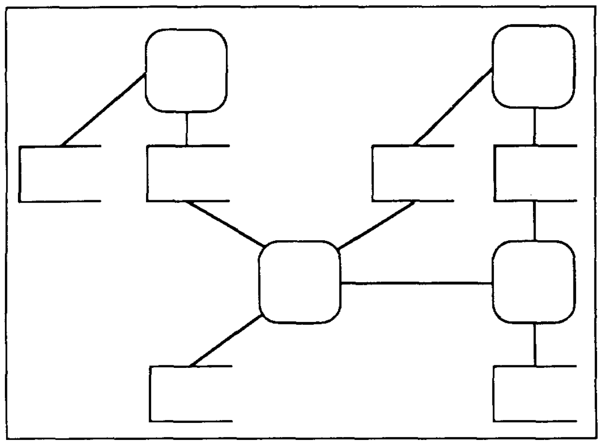


Fig. 2. Straight-line standard.

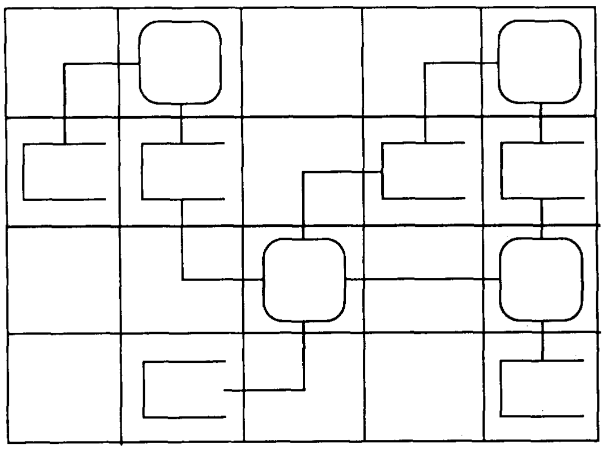


Fig. 3. Grid standard.

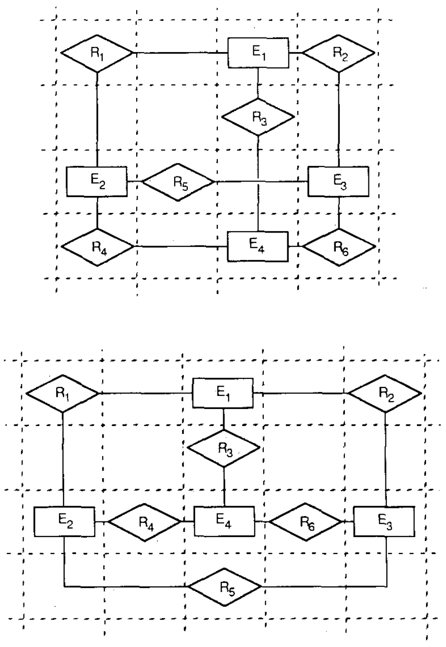


Fig. 4. Conflicts among aesthetics.

graphic standard, is the minimization of *crossings* between edges. Also, to avoid unnecessary waste of space, it is usual to keep the *area* occupied by the drawing reasonably small. When the grid standard is adopted, it is meaningful to minimize the number of *bends* (turns) along the edges, as well as their total *length*.

Usually, more than one aesthetic is considered in real life applications. Drawings that are optimal with respect to a specific aesthetic are generally not optimal with respect to another one. In Fig. 4 two equivalent diagrams are shown that minimize the number of bends and the number of crossings, respectively. An ideal algorithm should be able to take into account variable weights for the different aesthetics.

Constraints: Aesthetics characterize a tidy drawing from the graphical point of view. However, they cannot deal with features that require knowledge about the meaning of the drawing. For example, if there is a concept of utmost importance, the drawing is usually arranged so that the corresponding symbol is placed in the center of the diagram. Semantic features can be expressed by means of *constraints* on the drawing, which must be explicitly provided to the algorithm as additional input. Examples of constraints are positioning a group of vertices close one to the other and placing specific vertices on the external boundary of the drawing.

The following taxonomy can be applied to both aesthetics and constraints and is useful to understand their interaction. An aesthetic or constraint may be:

1) *local/global:* local when it refers only to a part of the drawing, global otherwise.
2) *hierarchic/flat:* hierarchic when it concerns the relative position of a set of symbols, flat otherwise.

Identifying a simple set of aesthetics that capture the intuitive notion of a readable drawing is a significant problem, as pointed out in [41]. In this framework, an analysis of the literature has been performed in [4] to determine the aesthetics most commonly adopted in several information systems diagrams. The results of this analysis are summarized in Table II, where the aesthetics are classified according to the local/global and hierarchic/flat taxonomy. In Table III we give a similar classification for several types of constraints. In the rest of this paper, specific aesthetics and constraints will be referred to by means of the acronyms given in Tables II and III.

In many documentation applications a sequence of drawings is produced by means of successive updates. An example can be found in top-down methodologies for software development, where new diagrams are created by expanding symbols into more complex structures. We would expect two successively generated diagrams to differ only locally, in those places where the expansion has been performed. In this case, dynamic aesthetics and constraints can be considered that minimize the sum of the "distances"

TABLE II
A TAXONOMY OF AESTHETICS

Acronym	Aesthetic	Category
AREA	minimization of the area occupied by the drawing	G/F
BALAN	balance of the diagram with respect to the vertical axis or horizontal axis	G/H
BENDS	minimization of the number of bends along the edges	G/F
CONVEX	maximization of the number of faces drawn as convex polygons	G/F
CROSS	minimization of crossings between edges	G/F
DEGREE	vertices with high degree in the center of the drawing	L/F
DIM	minimization of differences among vertices dimensions	G/F
LENGTH	minimization of the global length of edges	G/F
MAXCON	minimization of the length of the longest edge	G/F
SYMM	symmetry of sons in hierarchies	L/H
UNIDEN	uniform density of vertices in the drawing	G/F
VERT	verticality of hierarchic structures	L/H

TABLE III
A TAXONOMY OF CONSTRAINTS

Acronym	Constraint	Category
CENTER	place a set of given vertices in the center of the drawing	L/F
DIMENS	assign the dimension of the symbols representing specified vertices	L/F
EXTERN	place specified vertices on the external boundary of the drawing	L/F
NEIGH	place close together a group of vertices	L/H
SHAPE	draw a subgraph with a prespecified shape	L/H
STREAM	place a sequence of vertices along a straight line	L/H

between all consecutive drawings of the sequence, where the distance between two drawings is suitably defined.

Computational Complexity: With regard to complexity issues, the running time of the drawing algorithm is critical in interactive applications. However, it should be pointed out that many natural aesthetics correspond to *NP*-hard optimization problems. This is the case, for example, for aesthetic CROSS [25] and AREA [39]. This explains why most algorithms are heuristic. All the complexity measures mentioned in this paper refer to the standard RAM model [1].

B. Trees

A first group of algorithms considers binary trees [34], [41], [50], [53]. Following the usual way of representation, these algorithms adopt the straight-line standard and the aesthetics CROSS, VERT, SYMM, and AREA. These aesthetics are specialized as follows:

CROSS:	edges do not cross;
VERT:	nodes at the same level in the tree are placed along a horizontal line, with a minimum distance δ between any two consecutive nodes;
SYMM:	fathers are horizontally centered above their sons;
AREA:	the width of the drawing is minimized.

Before describing the algorithms, we notice that all of them have time complexity $O(n)$, and produce drawings of width $O(n)$, which is asymptotically optimal in the worst case.

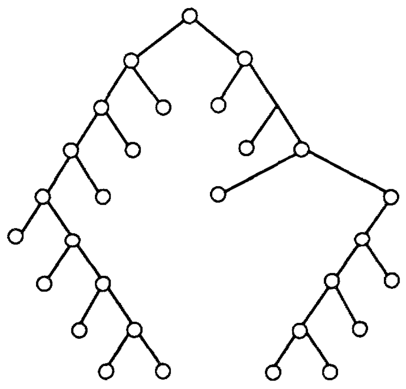


Fig. 5. Tree drawn with algorithm of Wetherell and Shannon (from [53]).

Two algorithms with similar structure have been presented by Wetherell and Shannon [53]. The position of each node is assigned by means of two traversals of the tree. Nodes are pushed to the left as much as possible, provided they are correctly placed with respect to their father and sons. The first version of the algorithm is heuristic with respect to the aesthetic AREA. The second one considers a weaker variant of aesthetic SYMM, i.e., a node is placed between its sons and finds a drawing of minimum width.

Vaucher [50] independently developed a tree drawing algorithm similar to the one of Wetherell and Shannon. His algorithm seems to produce a minimum-width drawing, although this is not clearly stated in the paper.

Reingold and Tilford [34] observed a drawback common to all of the previous algorithms: the drawing of a subtree is influenced by the positioning of nodes outside that subtree, so that symmetric trees may be drawn asymmetrically (see Fig. 5). To guarantee that a symmetric tree be drawn symmetrically, they introduced aesthetic ISO, formulated as follows:

ISO: isomorphic subtrees must have the same drawings, and symmetric subtrees must have mirror image drawings.

The tree drawing algorithm of Reingold and Tilford is simple and elegant:

```
if   the tree is empty or consists of a single node
then trivially construct its drawing
else recursively draw the left and right subtrees;
     place the two subdrawings so obtained close to one
     another so that the minimum horizontal distance
     between them is $\delta$;
     position the root halfway between the roots of the
     subtrees (if one of the subtrees is empty, the root is
     placed at distance $\delta/2$ from the root of the other
     subtree).
endif □
```

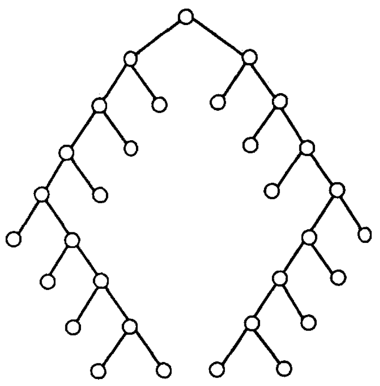


Fig. 6. Tree of Fig. 5 drawn with algorithm of Reingold and Tilford (from [34]).

Fig. 6 shows the tree of Fig. 5 as drawn by the previous algorithm. The algorithm produces a drawing that is not guaranteed to be of minimum width and, in fact, can be $O(n)$ times wider than necessary [41]; however, its performance is very good from the aesthetic point of view. The extension to the case of general ordered trees is straightforward: once the subtrees have been recursively drawn and placed close to one another, the root can be positioned halfway between the leftmost and rightmost son.

Supowit and Reingold [41] studied the computational complexity of the binary tree drawing problem. Their results can be summarized as follows: 1) the problem of finding a minimum-width drawing of a binary tree subject to aesthetics CROSS, VERT, SYMM, AREA, and ISO is solvable in polynomial time since it is reducible to linear programming; and 2) the discretized version of the earlier problem, where the x coordinates of the nodes are constrained to be integer multiples of $\delta/2$, is instead NP-hard. While the latter result justifies the heuristic approach of Reingold and Tilford, the former leaves open the question of finding an efficient algorithm for the continuous case.

C. Planar Graphs

Planar graphs can be drawn in the plane without crossing edges. Based upon this property, specific algorithms for drawing planar graphs without crossings have been devised [10], [31], [35], [39], [42], [43], [44], [49], [54]. Since in every planar graph $m \leqslant 3n - 6$, some of these algorithms achieve $O(n)$ time complexity.

Notice that an algorithm for drawing planar graphs can be used also for a nonplanar graph provided crossings are replaced by ficticious vertices.

Wagner [51] and Fary [20] proved that every planar graph admits a planar straight-line drawing. Tutte [48] extended this result by showing that every three-connected planar graph admits a *convex drawing*, i.e., a planar straight-line drawing such that all the face boundaries are convex polygons (see Fig. 7). He also gave a method for finding a convex drawing such that the external face is any prescribed convex polygon, and the position (x_v, y_v) of

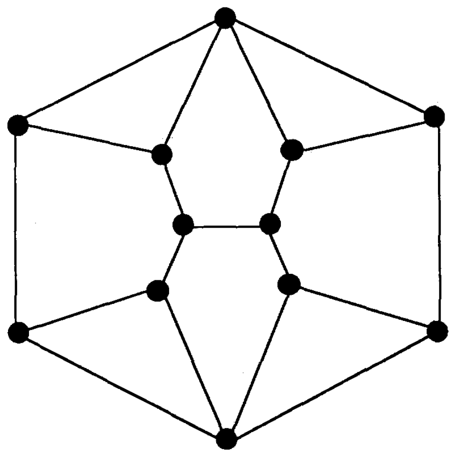


Fig. 7. Planar graph drawn with convex faces.

each vertex v is given by

$$x_v = \frac{1}{\deg(v)} \sum_{w \in \operatorname{adj}(v)} x_w \qquad y_v = \frac{1}{\deg(v)} \sum_{w \in \operatorname{adj}(v)} y_w$$

where $\operatorname{adj}(v)$ is the set of vertices adjacent to v. Using the results in [26], the algorithm of Tutte can be implemented in time $O(n^{1.5})$.

An $O(n)$ algorithm for constructing convex drawings has been presented by Chiba *et al.* [10]. It is based on the characterization given by Thomassen [47] of the class of planar graphs that admit a convex drawing. The algorithm can be extended to the case of general planar graphs and produces straight-line drawings such that the faces of each three-connected component are drawn as convex polygons. Experiments have shown that this algorithm sometimes distributes vertices unevenly, thus requiring high-resolution display devices.

When the grid standard is adopted, aesthetic BENDS is very important for a pleasing drawing. Storer [39] analyzed first the problem of embedding in the grid a planar graph with a given planar representation so that the number of bends is minimum. He conjectured this problem to be NP-hard and gave three heuristics that produce drawings with $O(n)$ bends. Although these heuristics appear to have polynomial time complexity, a running time analysis is not given. Tamassia [44] disproved Storer's conjecture by presenting an algorithm that finds an embedding with the exact minimum number of bends in $O(n^2 \log n)$ time. His algorithm is based on a transformation into a minimum cost flow problem and handles constraints on the number of bends allowed on each edge. Furthermore, the area of the embedding is $O(n^2)$, which is optimal in the worst case.

A *visibility representation* for a planar graph consists of representing vertices with horizontal segments and edges with vertical segments so that each edge segment has its endpoints on the segments associated with its incident vertices and does not cross any other vertex segment (see Fig. 8). Otten and van Wijk [31] introduced this represen-

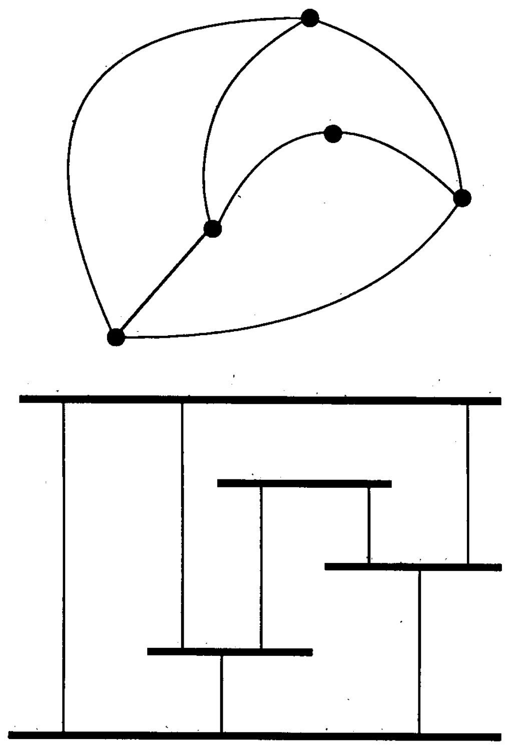


Fig. 8. Visibility representation.

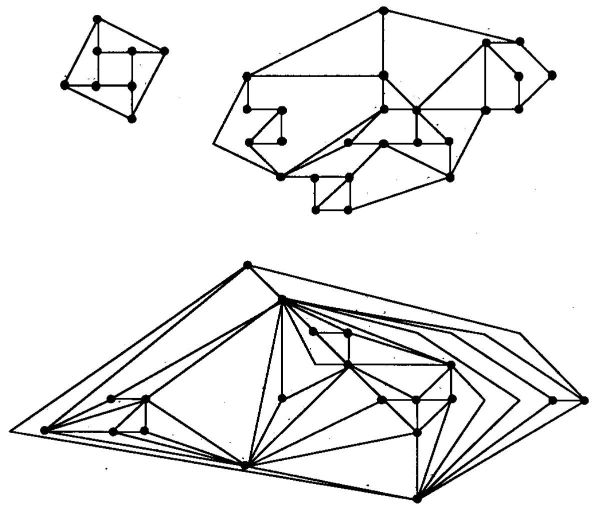


Fig. 9. Examples of drawings produced by algorithm of Woods (from [54]).

tation with application to circuit schematics and showed that every planar graph admits one. Tamassia and Tollis [42] presented an algorithm that constructs a visibility representation in $O(n)$ time. They also studied other types of visibility representations in a unified framework. As shown in [43], their algorithm can be modified to construct a grid embedding with $O(n)$ bends and $O(n^2)$ area in $O(n)$ time. A more detailed analysis shows that the performance guarantee is the same as the one of Storer's heuristics. Another variation of the Otten and van Wijk algorithm appears in [35].

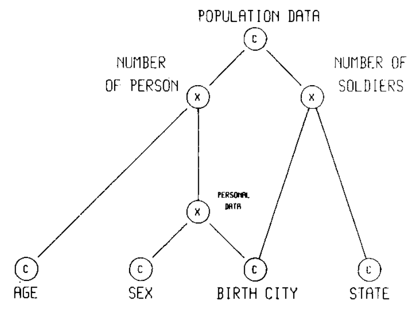


Fig. 10. Hierarchic graph used for statistical databases.

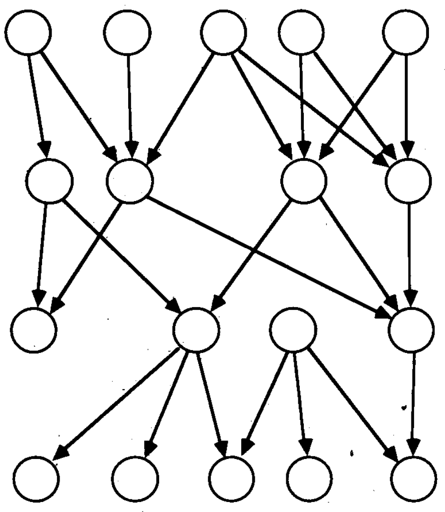


Fig. 11. Proper k-layer graph.

Woods [54] presented a drawing algorithm that adopts the mixed standard and takes into account aesthetics CROSS, BENDS, and AREA. This algorithm has $O(n^2)$ time complexity and produces drawings with $O(n^2)$ bends and $O(n^2)$ area. See examples in Fig. 9.

D. Hierarchic Graphs

Graphs representing hierarchic structures are widely used. Examples include PERT networks, subroutine-call graphs, family trees, and organization charts. These graphs have the common property to be directed and acyclic. In the following, they will be denoted as *hierarchic graphs*. It is usual to represent hierarchic graphs with the straight-line standard by arranging vertices into *levels* (parallel lines) so that the edges can be all drawn in the same direction, e.g., from top to bottom (aesthetic VERT). See an example in Fig. 10. Besides VERT, aesthetics CROSS and AREA are widely used for hierarchic graphs. A *k-layer graph* is a hierarchic graph where the assignment of vertices to the k levels is fixed. A k-layer graph is *proper* if edges connect pairs of vertices belonging to consecutive levels (see Fig. 11).

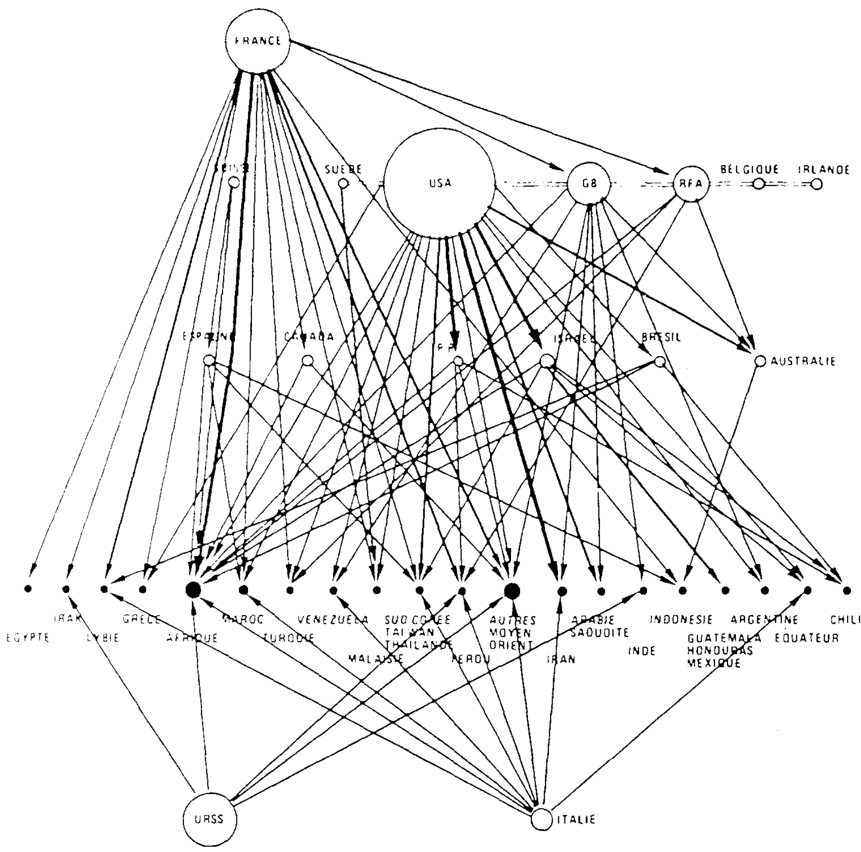


Fig. 12. Hierarchic graph drawn with Carpano algorithm (from [7]).

Theoretical results on drawing hierarchic graphs have been presented by Di Battista and Nardelli [14], who considered proper k-layer graphs with exactly one source vertex (vertex without incoming edges). They characterized the proper k-layer graphs that admit a drawing without crossings and presented a linear time algorithm for testing the "k-layer planarity." This algorithm produces also a cross-free drawing when the graph is found to be k-layer planar.

Eades *et al.* [16] showed that the problem of minimizing crossings in a proper k-layer graph is NP-hard, even for two-layer graphs where the positions of the vertices on the second layer are held fixed. Eades and Kelly [17] gave four heuristics for reducing the number of crossings in a k-layer graph, with running times between $O(m + n \log n)$ and $O(mn^2)$. The experimental results showed that the performance increases with the density and is very good for densities above 0.5, where the density of a two-layer graph with n_1 vertices on the first level and n_2 vertices on the second level is defined as m/n_1n_2. Eades and Wormald [18] presented another heuristics for reducing crossings in a two-layer graph that guarantees to produce a drawing with at most three times the minimum number of crossings. Warfield [52] developed a "crossing theory" for proper k-layer graphs and gave heuristic methods for reducing the number of crossings.

Algorithms for drawing hierarchic graphs appear in [7], [37], [40]. The algorithms of Carpano [7] and Sugiyama *et al.* [40], although independently discovered, present several similarities.

Carpano proposed an iterative method for the reduction of crossings in a two-layer graph, called *relative degree algorithm*: given a placement of the vertices on level 1, the abscissa of each vertex on level 2 is computed as the average of the abscissas of its neighbors on level 1 (*up-barycenter*). Then, a symmetric step is performed by computing the *down-barycenter* for the vertices on level 1, maintaining fixed the position of vertices on level 2. Iterating this procedure, the position of the vertices on both levels is guaranteed to converge under mild assumptions. The foregoing algorithm is also extended to the case of k-layer graphs. To deal with general hierarchic graphs, vertices are assigned to levels according to aesthetics VERT and LENGTH. Namely, v is positioned at level i iff the length of a longest path originating at v is i.

No theoretical results on the performance of the algorithm are given. However, an experimental study conducted on real-life examples showed a significant reduction in the number of crossings (30–50 percent less). An example of a drawing produced is shown in Fig. 12. The algorithm is part of the GT1VX tool for computer-aided decision analysis, which allows for interactive editing and

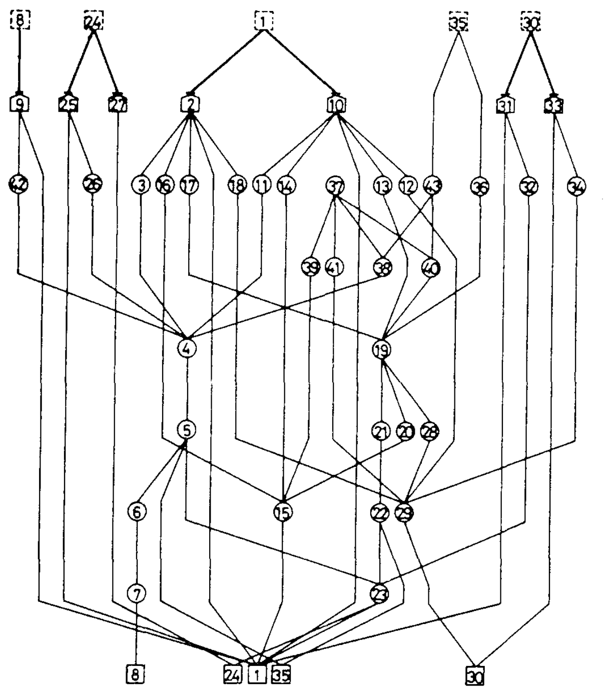


Fig. 13. Hierarchic graph drawn with algorithm of Sugiyama *et al.* (from [40]).

display of hierarchic graphs. Also, undirected graphs and directed graphs with cycles can be represented. In the former case a different graphic standard is used, called *centrifugal representation*, where the vertices are placed along concentric circles. In the latter case the adjacency graph of the strongly connected components (which is always acyclic) is displayed, and each strongly connected component is drawn separately as an undirected graph.

The algorithm of Sugiyama *et al.* [40] for drawing general hierarchic graphs appears to be more sophisticated. The mixed standard is adopted, which allows "long" edges to bend at each level, thus giving more flexibility in the placement of the vertices and permitting a more compact layout. The algorithm consists of three phases:

1) – he first phase assigns vertices to levels. Dummy vertices are added to break edges that span over more than two levels, so that the graph at the end of the first phase is a proper k-layer graph.

2) The second phase sorts the vertices on each level, with the goal of minimizing crossings. It is an iterative method that computes at each iteration a tentative placement. The first iteration scans the graph from level 1 to level k, where at level i $(i=1,\cdots,k)$ the vertices are sorted according to the up-barycenter. The second iteration is symmetric; it works from level k to level 1 and uses the down-barycenter. Successive iterations alternate between top-down and bottom-up scannings of the levels.

3) The third phase is a fine tuning of the layout that determines the absolute positions of the vertices in the drawing, with the goal of distributing uniformly the vertices

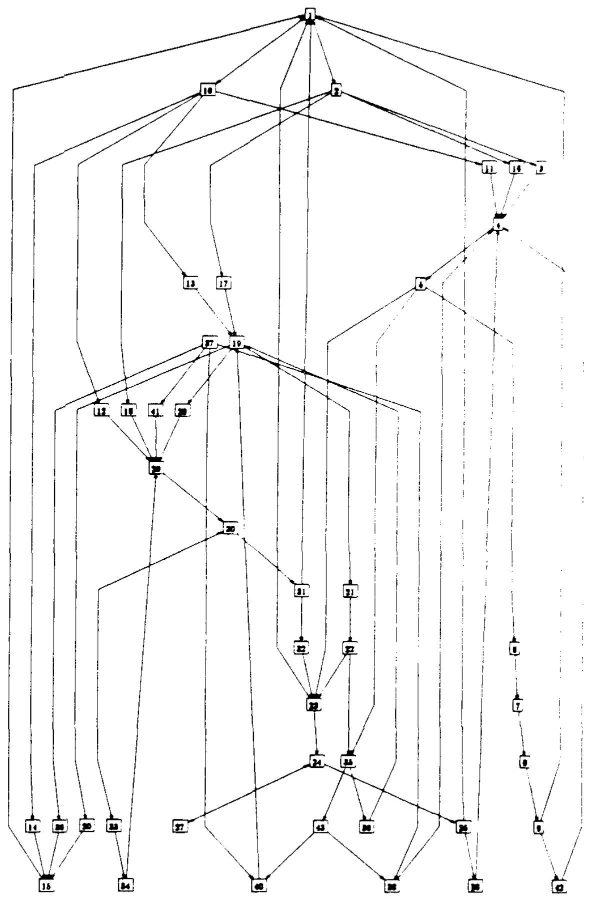


Fig. 14. Drawing of Fig. 13 drawn by algorithm of Rowe *et al.* (from [37]).

and straightening as much as possible the "long" edges (aesthetics UNIDEN and BENDS).

The case of directed graphs with cycles is handled by collapsing cycles into supervertices. Experiments performed with a FACOM M190 computer on real-life examples having up to 500 vertices showed a significant reduction of the number of crossings. An example of a drawing produced is shown in Fig. 13.

The algorithm of Sugiyama *et al.* has been modified and extended by Rowe *et al.* [11], [27], [37]. The main improvement concerns phase 2. In the original algorithm of Sugiyama *et al.*, when the order given by up- and down-barycenters is very different, the position of some vertices never converges. To overcome this problem, phase 2 has been modified so that the third and successive iterations sort by the average of up- and down-barycenters. Other improvements have been made to phase 3. The most important extension is that directed graphs with cycles are also handled by the algorithm. The basic idea is to temporarily reverse the directions of some edges to eliminate all cycles and then apply the hierarchic layout algorithm. This algorithm is part of the GRAB system [37], a general purpose browser for directed graphs that allows operations of edit, display, and automatic layout on directed graphs. The drawing produced by GRAB for the graph of Fig. 13 is shown in Fig. 14. A prototype version of GRAB has been implemented on a Sun workstation.

E. General Undirected Graphs

Because of the difficulty of the general problem, only a few layout algorithms have been presented for the case of arbitrary undirected graphs [15], [33]. The *spring embedder* algorithm by Eades [15] is a drawing heuristics based on a physical model of the layout process. The straight-line standard is adopted, and the aesthetics considered are UNIDEN and SYMM. The graph is represented by a mechanical system, where the vertices are replaced by rings, and the edges are replaced by springs. The force $F_1(u,v)$ exerted by a spring on the two endpoint rings u, v, is of logarithmic type

$$F_1(u,v) = f_0 \cdot \log \frac{d(u,v)}{d_0}$$

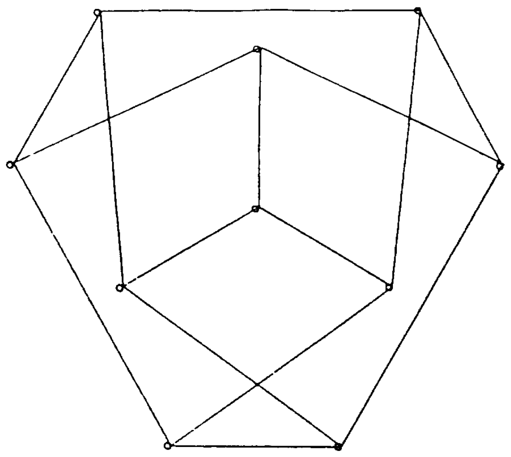
where f_0 and d_0 are constants and $d(u,v)$ is the length of the spring. Therefore, for $d(u,v) > d_0$, the spring attracts the rings, while for $d(u,v) < d_0$, it repels them. Also, there is a force $F_2(u,v)$ between nonadjacent rings with inverse square law

$$F_2(u,v) = \frac{d_1}{d(u,v)^2}.$$

The model is devised so that the adjacent vertices are put together, but not too close, and nonadjacent vertices remain well apart.

The drawing algorithm simulates the evolution of the foregoing system from a given initial state until a stable configuration (local minimum energy) is reached. Experiments conducted have shown that the algorithm produces good drawings for many two-connected sparse graphs (see Fig. 15(a)). However, poor drawings are obtained for graphs with dense subgraphs and graphs with a small number of bridges (see Fig. 15(b)). Using a discretized time approximation, the computational load is $O(n^2)$ for each simulation step, where about 100 steps were found to be enough for most graphs. The spring embedder is one of the layout tools available in TYGES (typed graph editing system) [23], an extensible system for editing and displaying graphs that has been implemented on an ICL Perq workstation.

A drawing algorithm specifically targeted to entity-relationship diagrams is presented in [33]. The grid standard is adopted, and a *divide and conquer* technique is used, where the graph is partitioned into clusters. Fast running time was the main concern in the design of this heuristics, whose goal is to produce quickly a reasonable layout. The previous layout algorithm is intended to be used within DDEW, a graphic tool for database design currently under implementation [32]. The algorithms surveyed in this section are briefly summarized in Table IV of Section III, after the description of our drawing algorithm.


(a)

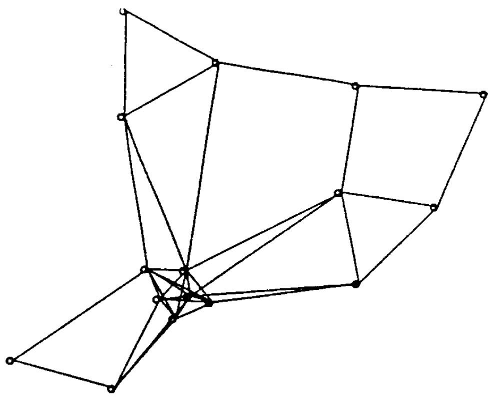

(b)

Fig. 15. Drawings produced with algorithm of Eades (from [15]).

III. A Graphtheoretic Approach To Automatic Graph Drawing

In this section we describe our approach to automatic graph drawing. We represent the layout problem by means of a general graphtheoretic model that allows homogeneous treatment of a wide range of aesthetics and constraints, and can be thus applied to several classes of diagrams. Graphs are drawn using the grid standard, and vertices are represented by rectangles or by symbols embedded into a rectangle.

The basic idea of the approach is that within the grid standard a drawing is characterized by three fundamental properties, defined in terms of the equivalence classes they establish among drawings of the same graph:

1) *topology:* two drawings have the same topology if they can be obtained one from the other by means of a continuous deformation that does not alter the sequences of edges contouring the faces of the drawing;
2) *shape:* two drawings have the same shape if they have the same topology and can be obtained one

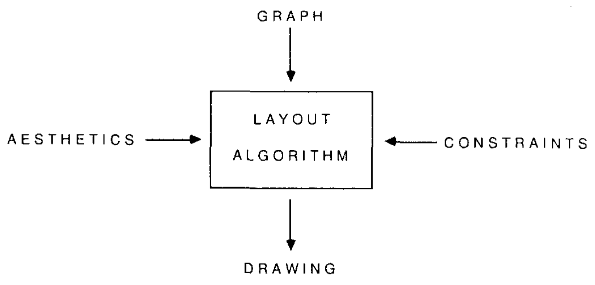


Fig. 16. Input/output description of algorithm.

from the other by modifying only the lengths of the edges, without changing the angles formed by them;

3) *metrics:* two drawings have the same metrics if they are identical up to translations.

Each of these properties provides a description of the layout that is a refinement of the previous one. Namely, two graphs with the same metrics have also the same shape, and two graphs with the same shape have also the same topology. At the same time, aesthetics and constraints typically address only one of the above properties; e.g., aesthetics CROSS, BENDS, and AREA deal with topology, shape, and metrics, respectively. Therefore, the hierarchy among topology, shape, and metrics suggests a stepwise generation of the drawing, where at each step one of the properties is taken into account and an intermediate representation is produced.

The algorithm we present has the following features:

1) accepts as input both general undirected graphs and mixed graphs that contain as subgraphs hierarchic structures;
2) adopts the grid standard;
3) deals with a rich variety of aesthetics;
4) allows interactive specification and treatment of several constraints; and
5) provides various levels of performance/runtime trade-off, depending on the application needs.

The aesthetics considered by the algorithm are CROSS, BENDS, LENGTH, and AREA. If the graph contains hierarchic subgraphs, the VERT and SYMM aesthetics are also applied. Constraints of type CENTER, EXTERN, NEIGH, SHAPE, and STREAM are accepted, as well as the following constraints (not mentioned in Table II) that are concerned with bounds on the properties of individual edges:

CMAX:	upper bound on the number of crossings along a given edge;
BMAX:	upper bound on the number of bends along a given edge;
LMAX:	upper bound on the length of a given edge.

Fig. 16 shows an input/output description of the algorithm.

The algorithm consists of three phases: *planarization*, *orthogonalization*, and *compaction* (see Fig. 17). The planarization phase determines the topology of the drawing, which is described by the *planar representation*. Formally, the planar representation of a planar drawing of a graph G is the set of ordered adjacency lists of the vertices of G, where the adjacency list of each vertex v is sorted according to the circular order in which the edges incident upon v appear clockwise around v in the drawing. For a nonplanar drawing the planar representation is defined similarly by replacing crossings with fictitious vertices. The planarization phase deals with aesthetic CROSS and takes into account constraints CENTER, CMAX, EXTERN, NEIGH, and SHAPE.

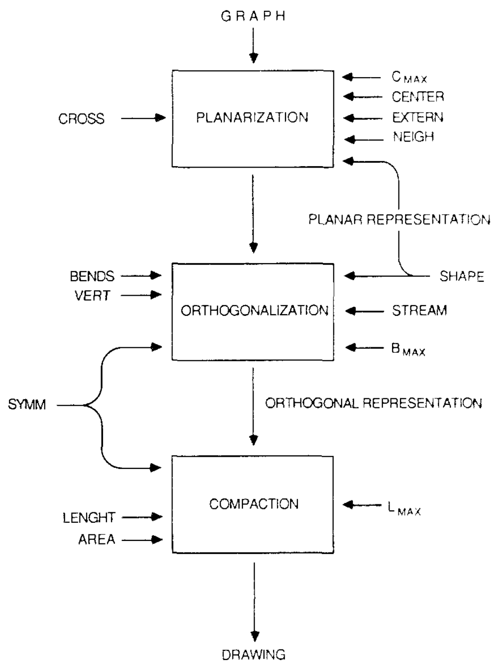


Fig. 17. Outline of layout algorithm.

The orthogonalization phase receives as input a planar representation and produces as output an *orthogonal representation* that describes the shape of the drawing. Formally, the orthogonal representation is defined as an enriched planar representation, where the sequence of angles along each edge is specified. Aesthetics BENDS, VERT, and SYMM, and constraints BMAX, SHAPE, and STREAM, are considered in this phase.

Finally, the third phase, compaction, determines the metrics of the drawing according to aesthetics AREA and LENGTH, and to constraint LMAX. Each phase is described in more detail in the next subsections.

A. Planarization

An expanded view of the planarization phase is shown in Fig. 18. The first step preprocesses the graph in case constraints EXTERN, NEIGH, and SHAPE are present. First, a new dummy vertex x is added to the graph and is connected to all vertices constrained to appear on the external face (constraint EXTERN). A bound CMAX $= 0$ is imposed on the edges incident upon x. Then the subgraphs of fixed

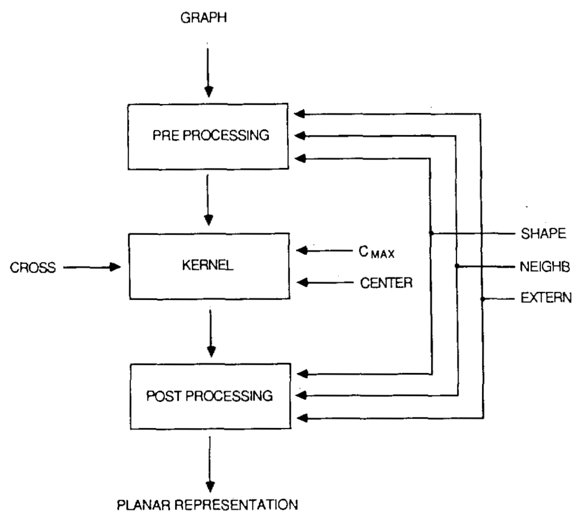


Fig. 18. Outline of planarization phase.

shape (constraint SHAPE) and the groups of vertices that have to appear together in the drawing (constraint NEIGH) are contracted into supervertices.

The second step is the kernel of the planarization phase. A modified planarity testing is performed [28] that outputs a maximal planar subgraph G' of the given graph (so if G is planar, $G' = G$). A planar embedding for G' is determined whose topology is described by the planar representation P'. The "nonplanar" edges are successively added to P', observing whenever possible the crossing bounds associated with the edges (CMAX) and minimizing at each step the number of crossings introduced. Finally, a planar representation P'' for the resulting planarized graph G'' is computed so that the external face contains vertex x (EXTERN) and is the most possible distant from the vertices to be placed in the middle of the drawing (CENTER). This step is repeated for each subgraph G_i condensed into a supervertex because of constraint NEIGH to construct its planar representation P_i. Notice that the planar representations of the other condensed subgraphs is already provided by constraint SHAPE.

In the third step the dummy vertex x is removed and the supervertices are expanded into the corresponding structures. This produces the final planar representation P. The overall complexity of the planarization phase is $O((n+c)\min(n, m'+1))$, where m' and c are the number of nonplanar edges and crossings, respectively. Notice that if the graph is planar, the complexity is $O(n)$. The heuristic approach used for this phase is justified by the fact that minimizing the number of crossings is an NP-hard problem [25].

B. Orthogonalization

A refined description of the second phase is shown in Fig. 19. This phase can be performed in two different ways, depending on the user's choice. In the first case, *fast orthogonalization*, a linear time algorithm is executed to

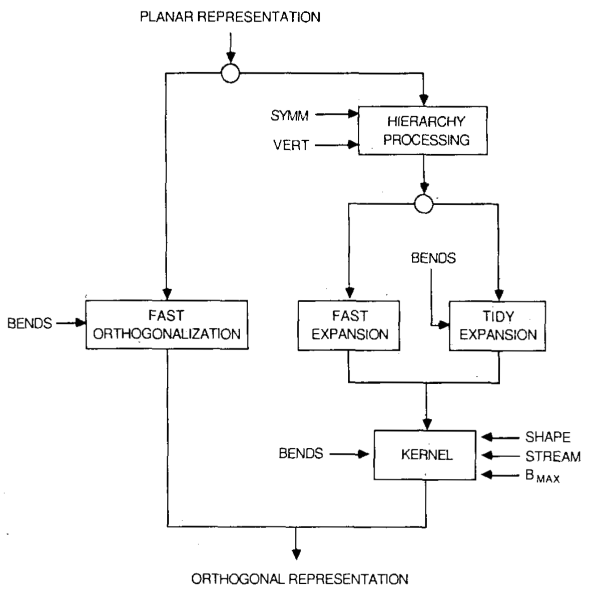


Fig. 19. Outline of orthogonalization phase.

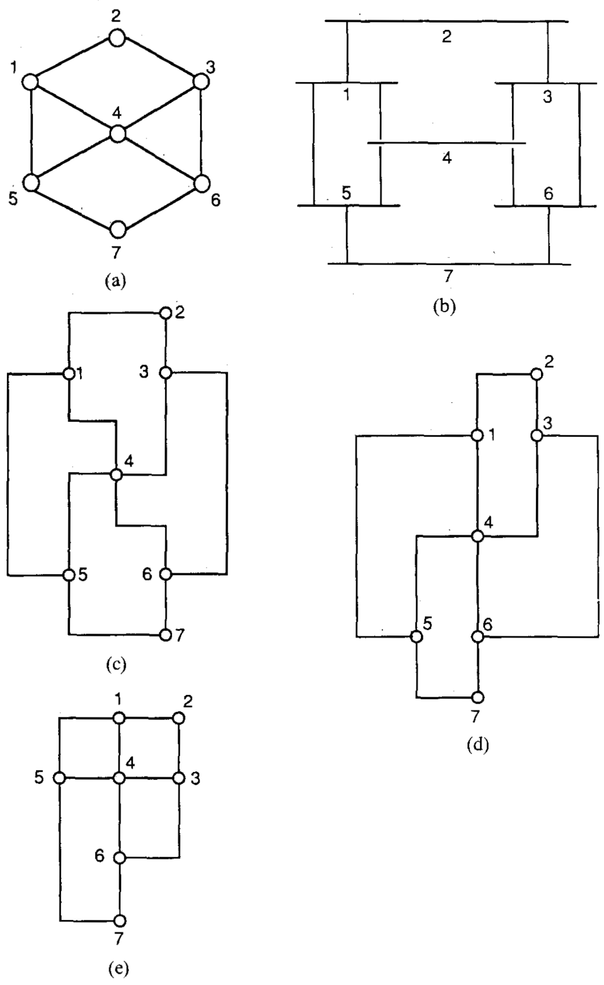


Fig. 20. Example of fast orthogonalization.

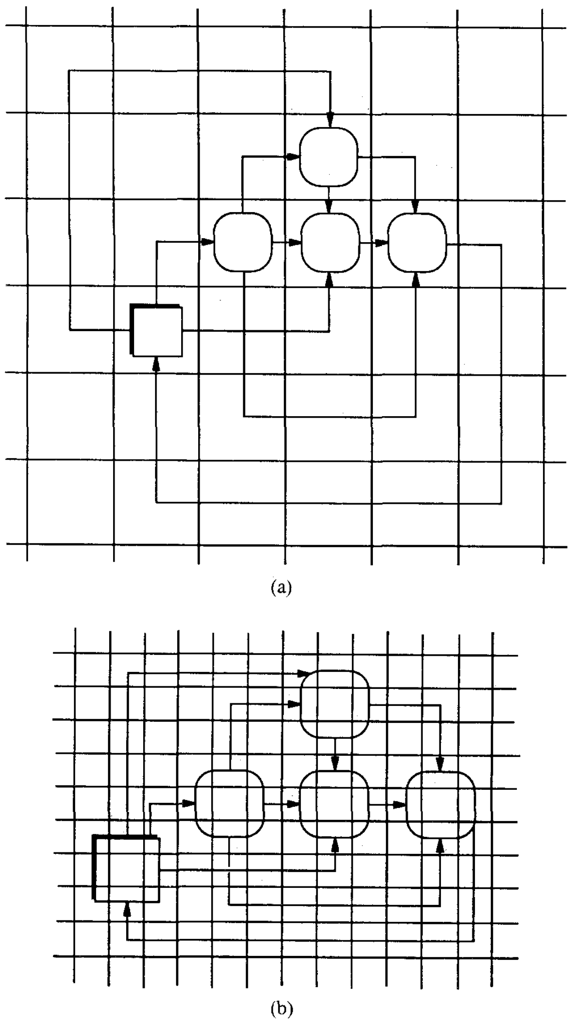


Fig. 21. Comparison among drawings produced with fast expansion (a) and tidy expansion (b).

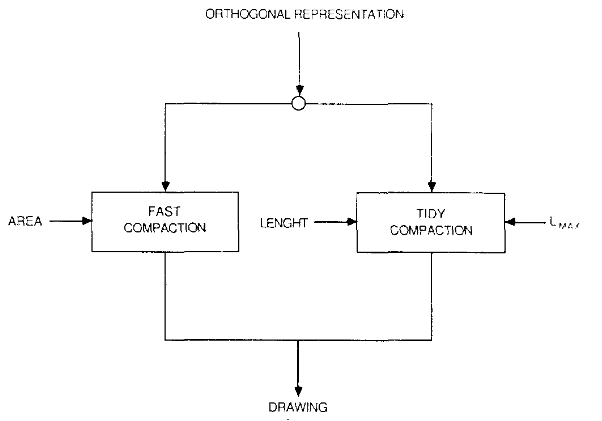


Fig. 22. Outline of compaction phase.

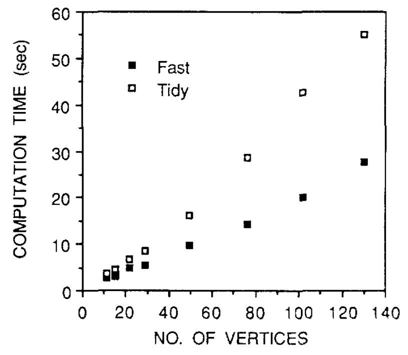


Fig. 23. Computation times of layout algorithm.

construct the orthogonal representation of the graph. This algorithm is heuristic and takes into account only aesthetic BENDS. However, even if the exact minimum number of bends is not guaranteed, good results are produced in practice, and at most $2.5(n+c)+4$ bends are introduced [43]. In the second case, *tidy orthogonalization*, all the aesthetics and constraints are taken into account, and a quadratic time algorithm that finds the exact minimum number of bends is used.

The fast orthogonalization procedure is illustrated in Fig. 20. From the planar representation P shown in Fig. 20(a), a visibility representation is constructed in linear time using an algorithm described in [42]. The visibility representation shown in Fig. 20(b), is then transformed into a grid embedding by means of local expansions (see Fig. 20(c)). Bend-stretching transformations are applied to reduce the number of bends, as shown in Fig. 20(d) and (e). For example, in Fig. 20(d) the pairs of concave and convex bends on edges (1, 4) and (4, 6) are removed. The output of these transformations is an orthogonal representation with at most $2.5(n+c)+4$ bends. Further details can be found in [43].

In the rest of this subsection we describe the tidy orthogonalization. The first step deals with the hierarchic subgraph G_H and takes into account aesthetics SYMM and VERT. The algorithm for this step constructs a shape for G_H that verifies the previous aesthetics. In the subsequent steps, this shape is maintained fixed by adding a constraint of type SHAPE.

The second step expands each vertex v of degree greater than four into a rectangular skeleton. At this point, two different strategies can be adopted to assign the edges incident upon v to the sides of the rectangle. The simplest strategy consists of placing $\deg(v)/4$ terminals along each side of the rectangle. The edges are associated with the terminals according to their circular order around v, which is given by the planar representation P; this can be done in linear time. A more sophisticated strategy consists of finding an "optimal" assignment by means of an algorithm based on the concept of *k-gonal representation* [44]. In this case the time complexity is quadratic, but the layout is improved as shown in Fig. 21. We conclude the discussion of the expansion step observing that the shape of the rectangular skeletons is kept fixed in the next step by introducing additional constraints of type SHAPE.

TABLE IV
A TAXONOMY OF GRAPH-DRAWING ALGORITHMS

Reference	Authors	Standard	Aesthetics/Performance	Constraint	Complexity
			Trees		
[53]	Wetherell, Shannon	*S*	CROSS/*E* VERT/*E* SYMM/*G* AREA/*E**	—	n
[50]	Vaucher	*S*	CROSS/*E* VERT/*E* SYMM/*E* AREA/*E*(?)	—	n
[34]	Reingold, Tilford	*S*	CROSS/*E* VERT/*E* SYMM/*E* AREA/*G* ISO/*E*	—	n
			Planar Graphs		
[49]	Tutte	*S*	CROSS/*E* CONVEX/*E*	—	$n^{1.5}$
[10]	Chiba *et al.*	*S*	CROSS/*E* CONVEX/*E*	—	n
[39]	Storer	*G*	CROSS/*E* BENDS/*G*	—	?
[44]	Tamassia	*G*	CROSS/*E* BENDS/*E* AREA/*G*	BMAX/*E* SHAPE/*E*	$n^2 \log n$
[43]	Tamassia, Tollis	*G*	CROSS/*E* BENDS/*G* AREA/*G*	—	n
[54]	Woods	*M*	CROSS/*E* BENDS/*H* AREA/*G*	—	n^2
			Hierarchic Graphs		
[7]	Carpano	*S*	CROSS/*H* VERT/*E* SYMM/*H* LENGTH/*H*	—	?
[40]	Sugiyama *et al.*	*M*	CROSS/*H* VERT/*E* SYMM/*H* LENGTH/*H* UNIDEN/*H* BENDS/*H*	—	?
[37]	Rowe *et al.*	*M*	CROSS/*H* VERT/*E* SYMM/*H* LENGTH/*H* UNIDEN/*H* BENDS/*H*	—	?
			General Undirected Graphs		
[15]	Eades	*S*	UNIDEN/*H* SYMM/*H*	—	sn^2**
[33]	Reiner *et al.*	*G*	CROSS/*H* BENDS/*H* AREA/*H*	CENTER/*G* NEIGH/*G*	?
[6]	Batini *et al.*	*G*	CROSS/*H* BENDS/*E* AREA/*G*	EXTERN/*E*	$(n+c)^2 \log n$
[†]	Tamassia, Di Battista, Batini	*G*	CROSS/*H* BENDS/*E* AREA/*G* LENGTH/*G* SYMM/*G* VERT/*G*	BMAX/*E* CENTER/*G* CMAX/*H* EXTERN/*E* LMAX/*G* NEIGH/*G* SHAPE/*E* STREAM/*E*	from n to $k^2(n+c)^2 \log kn$‡

*Another version of the algorithm has the following performance: CROSS/*E* VERT/*E* SYMM/*E*, AREA/*G*.
**s is the number of simulation steps used.
†This paper.
‡The complexity is $O(n)$ for planar graphs and fast orthogonalization and compaction, and $O(k^2(n+c)^2 \log kn)$ for nonplanar graphs and tidy orthogonalization and compaction. c is the number of crossings, and k is a parameter related to the tidy expansion step.

Once the expansion step has been performed, the kernel of the orthogonalization phase is executed to find a shape with the minimum number of bends subject to the constraints of type BMAX, SHAPE, and STREAM. The algorithm for this step has quadratic time complexity and is described in the Appendix. The final output of phase 2 is an orthogonal representation H for the graph.

C. Compaction

The third phase, compaction, is illustrated in Fig. 22. As in the previous phase, two algorithms with different performance/complexity trade-offs can be adopted.

The *fast compaction* algorithm considers only aesthetic AREA and runs in linear time. Namely, if the orthogonal representation H has n vertices, c crossings and b bends, the complexity is $O(n+c+b)$. The algorithm is a variation of the one described in [22] and can be summarized as follows. First, each face of the orthogonal representation is decomposed into rectangles. Then, two directed graphs, G_v and G_h, are constructed that represent the constraints imposed by the orthogonal representation H onto the vertical and horizontal metrics, respectively. These graphs are constructed in such a way that the lengths of the longest paths in G_v and G_h are equal to the minimum height and width of the drawing, respectively. Finally, the lengths of the edges are computed applying the *critical path method* [19] to G_v and G_h.

The *tidy compaction* algorithm has quadratic complexity and takes into account aesthetic LENGTH and constraint LMAX. After the faces of the orthogonal representation have been decomposed into rectangles, a network flow technique is used where each unit of flow represents a unit of length. Two flow networks are used, one for the vertical metrics and one for the horizontal metrics. The nodes of the networks are the face rectangles, and the conservation of the flow at each node means that opposite sides of a rectangle must have equal lengths. Finally, the minimization of the cost of the flow corresponds to the minimum total length of the edges.

Experiments conducted on real-life examples have shown that the algorithm achieves satisfactory runtime performance. We show in Fig. 23 the computation time on a Sun-2 workstation for several graphs with average vertex degree between 3 and 4.

To conclude this section, we compare in Table IV the algorithms presented in Section II and our algorithm. The algorithms are grouped according to the class of graphs considered. For each algorithm, we give the graphic standard (S-straight-line, G-grid, or M-mixed), the aesthetics and constraints taken into account, and the time complexity. Finally, for each aesthetic, we indicate if it is satisfied exactly (***E***), heuristically with some performance guarantee (***G***); or heuristically without any performance guarantee (***H***).

IV. Applications

In this section we show several applications of the layout algorithm described in Section III. This algorithm has been implemented in GIOTTO, a tool for computer-aided production and manipulation of diagrams used in functional analysis and conceptual data design. The automatic layout facility provided by this tool can be used for a variety of purposes:

1) *documentation*, either as a self-standing system or as a "back-end" for an existing documentation tool;
2) *support to a graphic editor*, to optimize diagrams produced with the editor;
3) *support to a design tool*, to perform the layout activities associated with complex design commands.

Now, we examine several applications in these areas.

A. Documentation

The simplest use of GIOTTO is as a self-standing documentation tool. In this case, the user declares the structure of a data (or functional) schema by filling in a form. Figs. 24 and 25 show two forms and the corresponding diagrams for an entity-relationship schema and a data-flow schema, respectively. Notice that aesthetic VERT is satisfied for the hierarchic substructure of Fig. 24, consisting of the generalization abstraction between entities PERSON, WOMAN, and MAN. The tool produces a diagram suitably normalized to a particular format, depending on the output device (video, plotter, printer). Diagrams can be displayed at several detail levels. For example, in entity-relationship schemas, attributes of entities can be declared with a separate form and displayed in a more detailed diagram (see Fig. 26). The attributes are handled by the algorithm as additional symbols, whose connections are kept straight and short using constraints of type BMAX and LMAX.

GIOTTO can also be integrated in existing documentation tools to provide consistency between textual and graphic

ENTITIES:

1 PLACE   2 PERSON   3 MAN   4 WOMAN   5 SOLDIER

RELATIONSHIPS:

MET 1-3-4   BORN 1-2   SERVESIN 1-5

GENERALIZATIONS:

SEX 3,4 → 2

SUBSETS:

MAN-SOLD 5 → 3

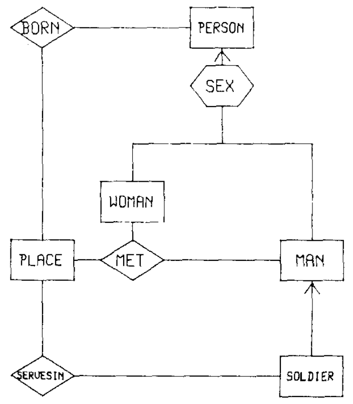


Fig. 24. Entity-relationship schema and corresponding diagram.

INTERFACES:

1 GOVERNM   2 DRIVER

STORES:

3 BUSES   4 TRAVELS   5 TIMETAB   6 FARES   7 DRDATA

PROCESSES:

8 TRAVELSM   9 BUSM   10 UPDATE   11 DRIVERSM

FLOWS:

1-8	5-8	8-6	8-4	3-9	6-9
9-7	9-4	4-10	10-2	2-11	11-7

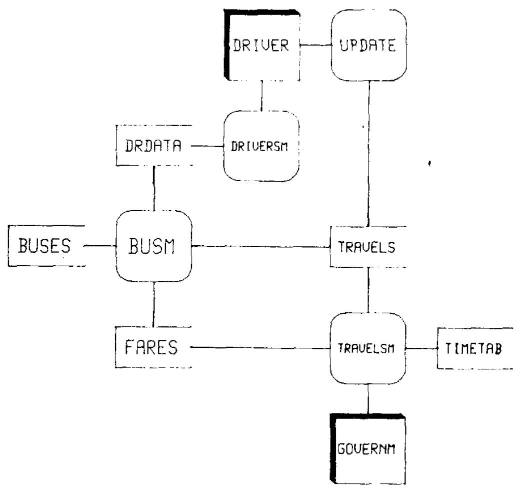


Fig. 25. Data-flow schema and corresponding diagram.

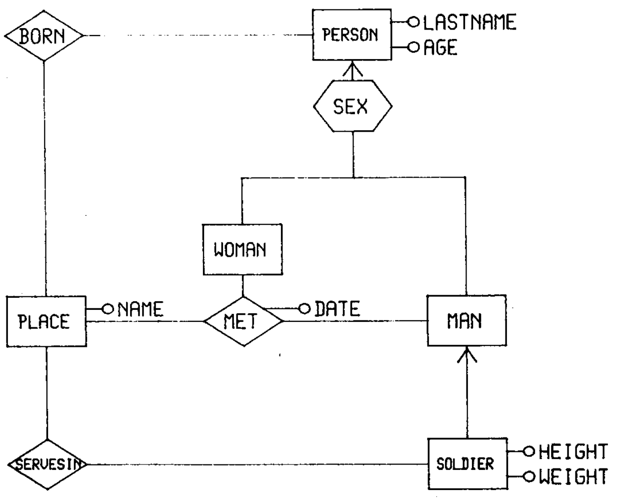


Fig. 26. Diagram of Fig. 24 including attributes.

```
DEFINE ENTITY       SEAT-TIME;
     DERIVED BY     SEAT-RES;
     EMPLOYED BY    TIME-INQ;
DEFINE ENTITY       TIME-FARE;
     DERIVED BY     TIME-INQ;
     EMPLOYED BY    FARE-INQ;
DEFINE INPUT        TRAV-SEAT;
     GENERATED BY   TRAVELER;
     RECEIVED BY    SEAT-RES;
DEFINE INTERFACE    TRAVELER;
DEFINE OUTPUT       FARE-TRAV;
     GENERATED BY   FARE-INQ;
     RECEIVED BY    TRAVELER;
DEFINE PROCESS      FARE-INQ;
     EMPLOYS        FARES;
DEFINE PROCESS      TIME-INQ;
     EMPLOYS        TIMETAB;
DEFINE PROCESS      SEAT-RES
     EMPLOYS        TRAVELS;
DEFINE SET          FARES;
DEFINE SET          TRAVELS;
DEFINE SET          TIMETAB;
```

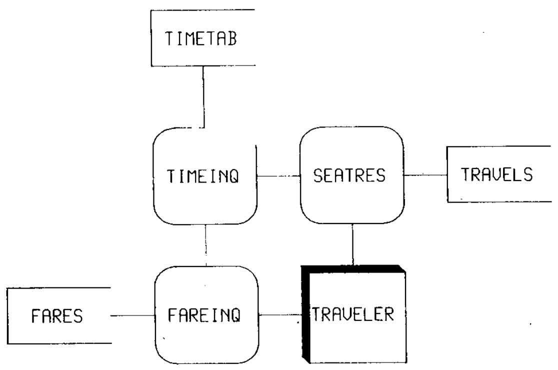


Fig. 27. Schema description in PSL and corresponding GIOTTO diagram.

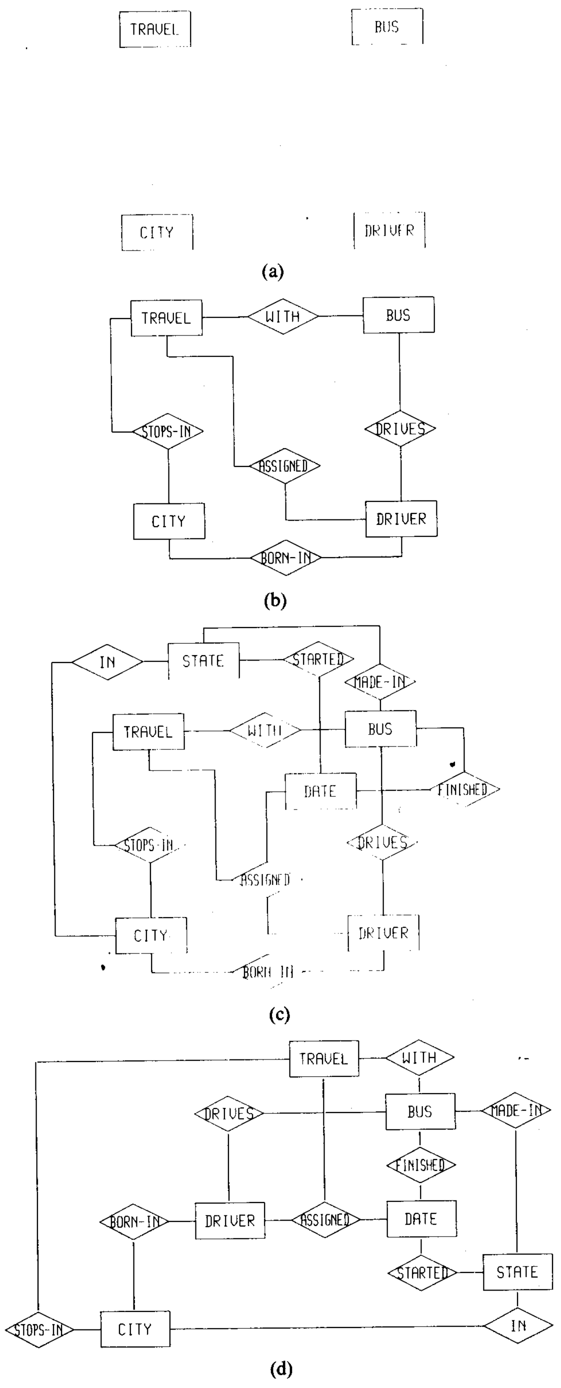


Fig. 28. Using optimize command. (a) Creation of group of entities. (b) Insertion of relationships. (c) Addition of new concepts. (d) Restructured diagram.

documentation, as noticed in the introduction. See, for example, in Fig. 27 a fragment of a schema description in PSL/PSA [46], and the corresponding diagram produced by GIOTTO.

B. Support to a Graphic Editor

GIOTTO has been used in connection with GINCOD [5], a graphic editor that provides traditional CREATE and DELETE commands for adding and removing symbols and connections to (from) the diagram. These commands can be performed either in user-driven mode (*U*-mode) or in system-driven mode (*S*-mode). For example, in the CREATE command the user declares the symbol to be added and its connections with the rest of the diagram. In *U*-mode, the placement of the symbol and the routing of connections are performed by the user, while in *S*-mode the routing is

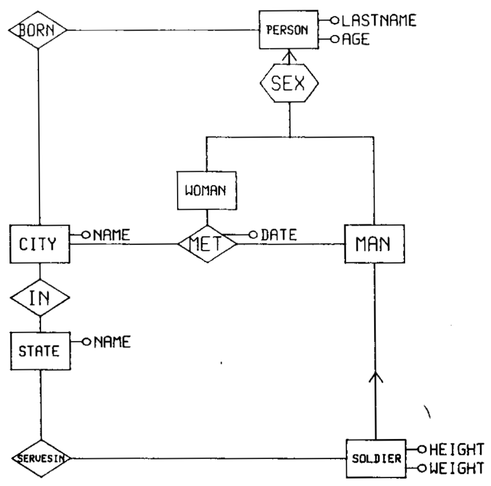


Fig. 29. Schema resulting from expansion.

performed by the system that adopts a simple heuristics. When a new connection is added, the system first attempts to find a free grid path to route it; if no such path exists, it is created by adding new grid tracks.

Design sessions consisting of long series of CREATE commands in U-mode may result in diagrams with inaesthetic crowding of connections. This problem can arise also in S-mode, since it provides only a local optimization. Now, the layout facility may be conveniently invoked using the OPTIMIZE command that produces a new diagram with an improved appearance. We show this facility by means of an example that refers to the information system of a travel company (see Fig. 28). First of all, entities TRAVEL, BUS, DRIVER, and CITY are created; then, a group of relationships is inserted. Further enrichments are made by adding new entities and relationships. At this point GIOTTO is called to restructure the diagram.

C. Support to a Design Tool

Methodologies currently used in data and functional analysis make use of a wide set of design primitives such as

1) EXPAND: replaces a symbol with a more refined structure;
2) INTEGRATE: merges two (or more) diagrams into a new global diagram;
3) CHANGE: restructures a diagram according to predefined "patterns" of transformation.

When applying these primitives with a graphic editor, it is not meaningful to leave to the designer the responsibility of moving symbols. For example, in the EXPAND command it is often necessary to make space for the new structure. In this case, GIOTTO may be invoked (using constraint SHAPE) to restructure the diagram only in the neighborhood of the modified part. The shape of the unmodified part changes softly so that the user can easily perceive the transformation. See in Fig. 29 the schema obtained ex-

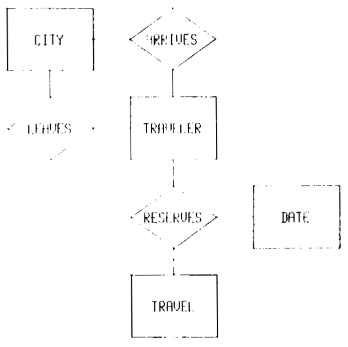


Fig. 30. New conceptual schema for travel company.

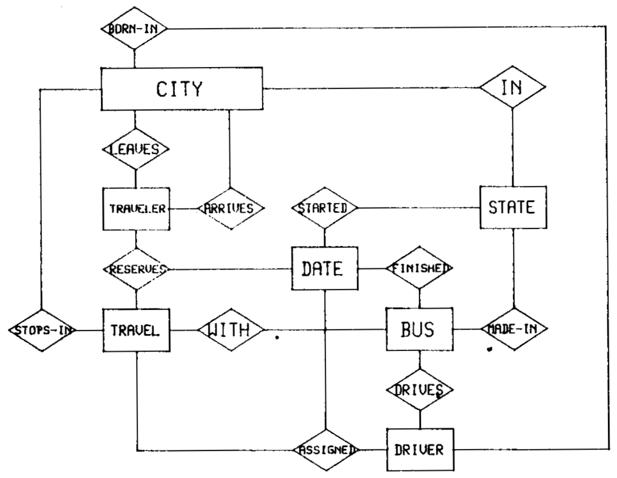


Fig. 31. Entity-relationship integrated schema.

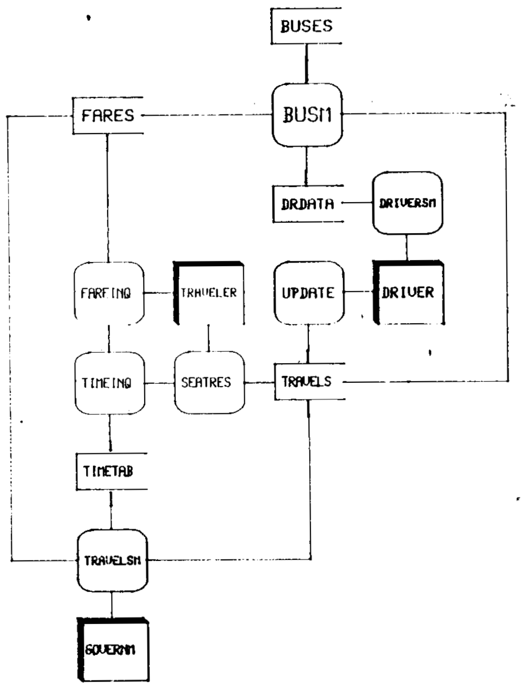


Fig. 32. Data-flow integrated schema.

panding the entity PLACE of Fig. 24 into one relationship and two entities.

In the INTEGRATE command, local restructurings are not meaningful, and GIOTTO is invoked for the whole layout. Concepts resulting from different schemas are drawn with different colors to keep track of the old schemas during the integration process. Fig. 30 shows an entity-relationship schema that may be integrated with the schema of Fig. 28; the integrated schema appears in Fig. 31. Finally, Fig. 32 shows the integration of the data-flow schemas of Figs. 25 and 27.

V. Future Developments

We have presented a comprehensive approach to aesthetic layout of diagrams and have shown applications of a layout algorithm that takes into account several aesthetics and user-defined constraints. The tool presented in Section IV is currently operating on an IBM PC with color display and on a Sun workstation. Both implementations are in Pascal.

The algorithm presented in this paper considers mainly undirected graphs and adopts the grid standard. We are investigating alternative ways to combine together the building blocks of the algorithm. Examples of problems currently under analysis are:

1) giving more priority to the aesthetic BENDS than to the aesthetic CROSS. To this extent, we are studying a new orthogonalization strategy that can modify the planar representation to further reduce the number of bends.

2) introducing a preliminary phase that detects global symmetries of the graph, and sets appropriate constraints in order to display them.

Other directions of future research include:

1) extending the graphtheoretic approach described in this paper to hierarchic graphs within the straight-line standard;
2) devising a layout strategy that allows the use of different graphic standards for different parts of the diagram, for example, hierarchic structures could be drawn with the straight-line standard, and the rest of the diagram with the grid standard;
3) developing a parametric algorithm that can be interactively tailored to specific classes of diagrams, graphic standards, aesthetics, and constraints. A first step in this direction is to express the structural properties of diagrams in a formal way (see a preliminary work in [4]). In this framework, we are presently investigating a unifying definition of several information system diagrams in terms of graph grammars.

Appendix

We describe the kernel of the tidy orthogonalization phase of the algorithm presented in Section III. This step receives as input a planar representation P, together with constraints of type BMAX, SHAPE, and STREAM. Namely, we are given:

1) a planar representation P, where the sets of vertices, edges, and faces are denoted with V, E, and F, respectively;
2) a bound maxbends (e) on the number of bends for each edge e;
3) a fixed orthogonal representation H' for a part P' of P.

The aesthetic considered is BENDS. The output is an orthogonal representation H for P with the minimum number of bends such that the bending bounds are satisfied, and the restriction of H to P' is equal to H'.

First, we consider the case in which only the bending bounds are present. The orthogonalization technique is based on a transformation into a network flow problem, where each unit of flow represents an angle of 90°. To avoid confusion, the terms *node* and *arc* will be used for networks instead of vertex and edge, respectively.

A flow network $N=(U,A)$ is constructed from P, as follows. The nodes of N are the vertices and faces of P: $U=V\cup F$. Each node u supplies flow $\text{SUPPLY}(u)$ given by:

$$\text{SUPPLY}(u)=\begin{cases}4, & \text{for each } u\in V\\ 4-2|u|-8\,\text{ext}(u), & \text{for each } u\in F\end{cases}$$

where $|f|$ is the length of the circuit contouring face f, and

$$\text{ext}(f)=\begin{cases}1, & \text{if } f \text{ is the external face}\\ 0, & \text{otherwise.}\end{cases}$$

Arcs of N are of two types, $A=A_F\cup A_V$, where the following hold:

1) A_F contains a pair of symmetric arcs, (f',f'') and (f'',f'), for every pair $\{f'',f'\}$ of distinct faces of P that share an edge in their contour. The flow $x(f,g)$ of arc $(f,g)\in A_F$ denotes the number of 90° angles inside face F that appear along edges separating f from g. The flow of each arc (f,g) has unit cost, zero lower bound, and upper bound (capacity) given by

$$\sum_{e\in E(f,g)}\text{maxbends}(e)$$

where $E(f,g)$ is the set of edges that separates face f from face g.

2) A_V contains arcs of the type (v,f), such that v is a vertex on the contour of face f. Flow $x(v,f)$ denotes the sum of angles with vertex v inside face f, measured in multiples of 90° (e.g., $x(v,f)=3$ for an angle of 270°). The flow of each arc (v,f) of A_V has zero cost, a lower bound equal to the number of angles with vertex v inside face f, and an upper bound $5-\deg(v)$.

The conservation of the flow at vertex nodes means that the sum of the angles around each vertex is equal to 360°. The conservation of flow at face nodes means that every face of P is transformed into a rectilinear polygon. Network N has the following properties [44]:

Proposition 1: For each integer circulation x in N, an orthogonal representation H computable from x exists that satisfies the bending bounds and has a number of bends equal to the cost of x.

Proposition 2: For each orthogonal representation H observing the bending bounds, a feasible circulation in N exists whose cost is equal to the number of bends in H.

Proposition 3: There are minimum cost circulations in N taking only integer values.

Hence an orthogonal representation with the minimum number of bends can be computed from a minimum cost circulation in N. Using standard network flow techniques for finding a minimum cost circulation, the orthogonalization algorithm can be performed in time $O(n^2 \log n)$.

To take into account constraints of type SHAPE and STREAM, we observe that imposing the given orthogonal representation H' to P' is equivalent to fixing the flow values of the arcs of a subnetwork N' of N. A simple method to achieve this is to set both the lower and the upper bound of the arcs in N' equal to the requested flow value and then simplifying the resulting network by applying iteratively the following rules:

1) If node u has an incoming arc (w, u) with both lower and upper bound equal to some value x, then increase the supply of u by x, decrease the supply of w by x, and remove arc (w, u) from the network.

2) If node u is isolated, i.e., without incident arcs, then remove it from the network.

This transformation can be performed in linear time and does not increase the asymptotic complexity of the algorithm.

ACKNOWLEDGMENT

We wish to thank Peter Eades for useful discussions and comments. We are also grateful to one of the referees for suggestions to improve the presentation of the paper.

REFERENCES

[1] A. Aho, J. Hopcroft, and J. Ullman, *The Design and Analysis of Computer Algorithms*. Reading, MA: Addison-Wesley, 1974.

[2] F. Aschim and B. M. Mostue, "IFIP WG 8.1 case solved using SYSDOC and SYSTEMATOR," in *Information Systems Design Methodologies: A Comparative Review (Proc. IFIP WG 8.1 Working Conf. Comparative Review of Information Systems Design Methodologies)* T. Olle *et al.*, Eds. New York: North-Holland, 1982, pp. 15–40.

[3] C. Batini, M. Talamo, and R. Tamassia, "Computer aided layout of entity-relationship diagrams," *J. Syst. Software*, vol. 4, pp. 163–173, 1984.

[4] C. Batini, L. Furlani, and E. Nardelli, "What is a good diagram? A pragmatic approach," in *Proc. 4th Int. Conf. Entity Relationship Approach*, Chicago, IL, 1985.

[5] C. Batini, E. Nardelli, M. Talamo, and R. Tamassia, "GINCOD: A graphical tool for conceptual design of data base applications," in *Computer Aided Data Base Design*, A. Albano, V. De Antonellis, and A. Di Leva, Eds. New York: North-Holland, 1985, pp. 33–51.

[6] C. Batini, E. Nardelli, and R. Tamassia, "A layout algorithm for data-flow diagrams," *IEEE Trans. Software Eng.*, vol. SE-12, no. 4, pp. 538–546, 1986.

[7] M. J. Carpano, "Automatic display of hierarchized graphs for computer aided decision analysis," *IEEE Trans. Syst. Man Cybern.*, vol. SMC-10, no. 11, pp. 705–715, 1980.

[8] S. Ceri, Ed., *Methodology and Tools for Data Base Design*. New York: North-Holland, 1983.

[9] E. Chan and F. H. Lochowsky, "A graphical data base design aid using the E-R model," in *Entity-Relationship Approach to Systems Analysis and Design*, P. Chen, Ed. New York: North-Holland, 1980, pp. 295–310.

[10] N. Chiba, K. Onoguchi, and T. Nishizeki, "Drawing planar graphs nicely," *Acta Informatica*, vol. 22, pp. 187–201, 1985.

[11] M. Davis, "A layout algorithm for a graph browser," M.S. Project Rep., EECS Dept., Univ. of California, Berkeley, 1984.

[12] T. De Marco, *Structured Analysis and System Specification*. Yourdon Press, 1979.

[13] Data Designer, "Product description," Ann Arbor, MI, 1981.

[14] G. Di Battista and E. Nardelli, "An algorithm for testing planarity of hierarchical graphs," in *Graph-Theoretic Concepts in Computer Science, Proc. Int. Workshop WG '86*, Bernierd, June 1986, G. Tinhofer and G. Schmidt, Eds., Lecture Notes in Computer Science, vol. 246. New York: Springer-Verlag, 1987, pp. 277–289.

[15] P. Eades, "A heuristic for graph drawing," *Congr. Numer.*, vol. 42, pp. 149–160, 1984.

[16] P. Eades, B. McKay, and N. Wormald, "An *NP*-hard crossing number problem for bipartite graphs," Dept. of Computer Science, Univ. of Queensland, Tech. Rep. 60, 1985.

[17] P. Eades and D. Kelly, "Heuristics for reducing crossings in 2-layered networks," *Ars Combinatoria*, vol. 21.A, pp. 89–98, 1986.

[18] P. Eades and N. Wormald, "The median heuristic for drawing 2-layered networks," Dept. of Computer Science, Univ. of Queensland, Tech. Rep. 69, 1986.

[19] S. Even, *Graph Algorithms*. Computer Science Press, 1979.

[20] I. Fary, "On straight lines representations of planar graphs," *Acta Sci. Math. Szeged*, vol. 11, pp. 229–233, 1948.

[21] C. Gane and T. Sarson, *Structured System Analysis*. Englewood Cliffs, NJ: Prentice-Hall, 1979.

[22] M. Y. Hsueh, "Symbolic layout and compaction of integrated circuits," Ph.D. dissertation, Univ. of California, Berkeley, 1979.

[23] J. Hynd and P. Eades, "The typed graph editing system—TYGES," in *Proc. 3rd Australasian Conf. Computer Graphics (Ausgraph 85)*, Brisbane, Australia, 1985, pp. 15–19.

[24] *Graphics Interfaces V. 2.1 for PSA or SEM—User's Manual*, ISDOS Ref. 82GI2-0400-0, ISDOS Project, Univ. of Michigan, Ann Arbor, 1982.

[25] D. S. Johnson, "The *NP*-completeness column: An ongoing guide," *J. Algorithms*, vol. 3, no. 1, pp. 89–99, 1982.

[26] R. Lipton, D. Rose, and R. Tarjan, "Generalized nested dissection," *SIAM J. Num. Anal.*, vol. 16, no. 2, pp. 346–358, 1979.

[27] C. Meyer, "A browser for directed graphs," M.S. thesis Project Rep., EECS Dept., Univ. of California, Berkeley, 1983.

[28] E. Nardelli and M. Talamo, "A fast algorithm for planarization of sparse diagrams," Rome, Tech. Rep. R.105, IASI-CNR, 1984.

[29] A. R. Newton, "Computer aided design of VLSI circuits," *Proc. IEEE*, vol. 69, no. 10, 1981.

[30] T. Olle, H. Sol, and Verrjin-Stuart, Eds., *Information Systems Design Methodologies: A Comparative Review (Proc. IFIP WG 8.1 Working Conf. Comparative Review of Information Systems Design Methodologies)*. New York: North-Holland, 1982.

[31] R. M. Otten and J. G. van Wijk, "Graph representations in interactive layout design," in *Proc. IEEE Int. Symp. Circuits and Systems*, New York, 1978, pp. 914–918.

[32] D. Reiner *et al.*, "A database design and evaluation workbench: Preliminary report," in *Proc. Int. Conf. Systems Development and Requirements Specification*, Gothenburg, Sweden, 1984.

[33] D. Reiner and G. Brown, "Heuristic layout for DDEW ER+ diagrams," Manuscript, Computer Corporation of America, 1985.

[34] E. Reingold and J. Tilford, "Tidier drawing of trees," *IEEE Trans. Software Eng.*, vol. SE-7, no. 2, pp. 223–228, 1981.

[35] P. Rosenstiehl and R. E. Tarjan, "Rectilinear planar layouts of planar graphs and bipolar orientations," *Discrete, Computational Geometry*, vol. 1, no. 4, pp. 342–351, 1986.

[36] D. Ross and K. Shoman, "Structured analysis for requirements definition," *IEEE Trans. Software Eng.*, vol. SE-3, no. 1, 1977.

[37] L. Rowe *et al.*, "A browser for directed graphs," Manuscript, E.E.C.S. Dept., Univ. of California, Berkeley, 1986.

[38] J. Soukup, "Circuit layout," *Proc. IEEE*, vol. 69, no. 10, 1981.

[39] J. A. Storer, "On minimal node-cost planar embeddings," *Networks*, vol. 14, pp. 181–212, 1984.

[40] K. Sugiyama, S. Tagawa, and M. Toda, "Methods for visual understanding of hierarchical systems," *IEEE Trans. Syst. Man Cybern.*, vol. SMC-11, no. 2, pp. 109–125, 1981.

[41] K. Supowit and E. Reingold, "The complexity of drawing trees nicely," *Acta Informatica*, vol. 18, pp. 377–392, 1983.

[42] R. Tamassia and I. G. Tollis, "A unified approach to visibility representations of planar graphs," *Discrete Computational Geometry*, vol. 1, no. 4, pp. 321–341, 1986.

[43] ____, "Efficient embedding of planar graphs in linear time," in *Proc. IEEE Int. Symp. Circuits and Systems*, Philadelphia, PA, 1987, pp. 495–498.

[44] R. Tamassia, "On embedding a graph in the grid with the minimum number of bends," *SIAM J. Comput.*, vol. 16, no. 3, pp. 421–444, 1987.

[45] *EXCELERATOR: A Guided Tour*, Index Technology Co., Cambridge, MA, 1985.

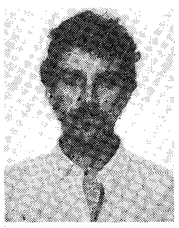
[46] D. Teichroew and E. Hershey, "PSL/PSA: A computer aided technique for structured documentation and analysis of information processing systems," *IEEE Trans. Software Eng.*, vol. SE-3, no. 1, 1977.

[47] C. Thomassen, "Planarity and duality of finite and infinite planar graphs," *J. Combin. Theory, Series B*, vol. 29, pp. 244–271, 1980.
[48] W. T. Tutte, "Convex representations of graphs," *Proc. London Math Soc.*, vol. 10, pp. 304–320, 1960.
[49] ____, "How to draw a graph," *Proc. London Math Soc.*, vol. 3, no. 13, pp. 743–768, 1963.
[50] J. Vaucher, "Pretty printing of trees," *Software Practice, Experience*, vol. 10, no. 7, pp. 553–561, 1980.
[51] K. Wagner, "Bemerkungen zum Vierfarbenproblem," *Jber. Deutsch. Math.-Verein*, vol. 46, pp. 26–32, 1936.
[52] J. Warfield, "Crossing theory and hierarchy mapping," *IEEE Trans. Syst. Man Cybern.*, vol. SMC-7, no. 7, pp. 502–523, 1977.
[53] C. Wetherell and A. Shannon, "Tidy drawing of trees," *IEEE Trans. Software Eng.*, vol. SE-5, no. 5, pp. 514–520, 1979.
[54] D. Woods, "Drawing planar graphs," Ph.D. dissertation, Computer Science Dept., Stanford Univ., Stanford, CA, Tech. Rep. STAN-CS-82-943, 1982.

Roberto Tamassia received the Dr.Ing. degree in electrical engineering from the University of Rome, "La Sapienza," Rome, Italy, in 1984. He is a doctoral candidate in the Department of Electrical and Computer Engineering of the University of Illinois at Urbana-Champaign.

He was previously a Research Associate in the Dipartimento di Informatica e Sistemistica of the University of Rome. He has been a consultant for several computer companies. His research interests include computational geometry, layout algorithms, graph theory, and computer-aided design of information systems.

Dr. Tamassia was awarded a Fulbright grant in 1985.



Giuseppe Di Battista received the Dr.Ing. degree in electrical engineering from the University of Rome "La Sapienza," Rome, Italy, in 1985.

He is now a Research Associate in the Dipartimento di Informatica e Sistemistica of the University of Rome "La Sapienza." His research interests include computer-aided design of information systems, layout algorithms, and graph theory.

Carlo Batini received the Dr.Ing. degree in electrical engineering from the University of Rome, "La Sapienza," Rome, Italy, in 1972.

He is a Professor in the Dipartimento di Informatica e Sistemistica of the University of Rome. His current research interests include methodologies for conceptual database design, graphic tools, and the topics of integration of knowledge bases.