A Layout Algorithm for Data Flow Diagrams

CARLO BATINI, ENRICO NARDELLI, AND ROBERTO TAMASSIA

Abstract—**A layout algorithm is presented that allows the automatic drawing of data flow diagrams, a diagrammatic representation widely used in the functional analysis of information systems. A grid standard is defined for such diagrams, and aesthetics for a good readability are identified. The layout algorithm receives as input an abstract graph, specifying connectivity relations between the elements of the diagram, and produces as output a corresponding diagram according to the aesthetics. The basic strategy is to build incrementally the layout; first, a good topology is constructed with few crossings between edges; subsequently, the shape of the diagram is determined in terms of angles appearing along edges; and finally, dimensions are given to the graph, obtaining a grid skeleton for the diagram.**

Index Terms—**Database design, design tools, functional analysis, layout algorithms.**

I. Introduction

STRUCTURED methodologies for information system design suggest wide use of diagrams as a documentation tool. Such diagrams are usually produced manually, or with a graphic editor; in both cases, the layout of the diagram is under the responsibility of the designer. We believe that a tool for automatic layout of information systems diagrams (ISD) would be very useful for designers in terms of the following:

- reduction of costs involved in the production and maintenance of diagrams;
- increase in the expressive power of diagrammatic representations, that stems from the fact that several aesthetics may be automatically satisfied;
- integration of the phases of conception and production of diagrams;
- automatic unified management of graphic and textual documentation; and
- increase of the communication bandwidth between the user and the designer.

Several layout algorithms have been developed for integrated circuits [12]: their most important goal is to minimize the circuit area. Since this gives rise to unaesthetic crowding of connections, they are not well suited for ISD's. Also, structural characteristics and constraints are quite different in the two cases.

Existing results in the specific field of aesthetic layout fall into the following cathegories.

1) Tidy Layout of Trees: In [11], [13], [16], and [17] a set of aesthetics has been defined for the tidy layout of binary trees, such as centering of fathers over their sons, minimum global width of the tree, and equal layout (up to translation) of isomorphic subtrees. The problem has been shown NP-hard in [13], while in [11], [16], and [17] several heuristics are proposed.

2) Automatic Display of Hierarchized Graphs: In [5] and [3], algorithms for layout of hierarchized graphs are proposed. Aesthetics considered are minimization of crossings between connections and uniform density of symbols.

3) Automatic Layout of Entity Relationship Diagrams: The problem of automatic layout of entity relationship diagrams [4], widely used in data analysis, has been studied in [2]: an algorithm is shown that embeds entity relationship diagrams into a grid according to several aesthetics.

In this paper, we present an algorithm for automatic layout of *data flow diagrams* [6], one of the most popular and effective representations for functional analysis. Problems dealt with in the paper, and solutions proposed, are sufficiently general to cover a wide class of applications. We show in Fig. 1 an example of a data flow diagram (DFD), taken from [6], that represents the information system of a book seller. The meanings of symbols are the following:

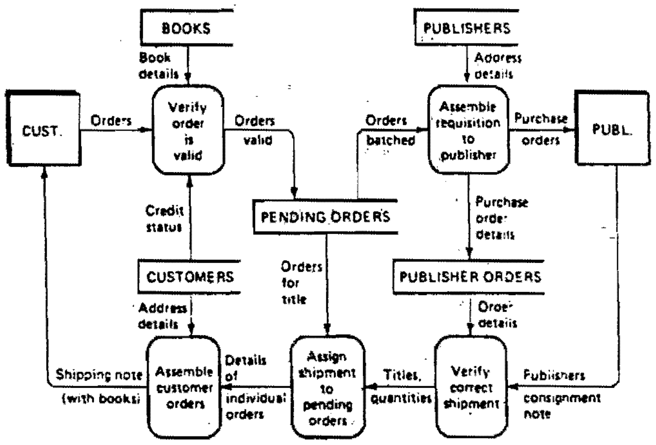


Fig. 1. Example of data flow diagram.

- *double squares* stand for *interfaces*, i.e., sources or destinations of data, external to the system (e.g., customers);
- *rounded rectangles* stand for *processes* which transform flows of data;
- *open-ended rectangles* represent *stores* of data; and
- *arrows* represent *flows* of data.

Manuscript received October 31, 1984. This work was supported in part by Progetto Finalizzato Trasporti of the National Research Council (CNR). The work of E. Nardelli was supported in part by ENIDATA S.p.A.

The authors are with the Dipartimento di Informatica e Sistemistica, Universitá di Roma "La Sapienza," Via Buonarroti 12, 00185 Rome, Italy.

IEEE Log Number 8607786.

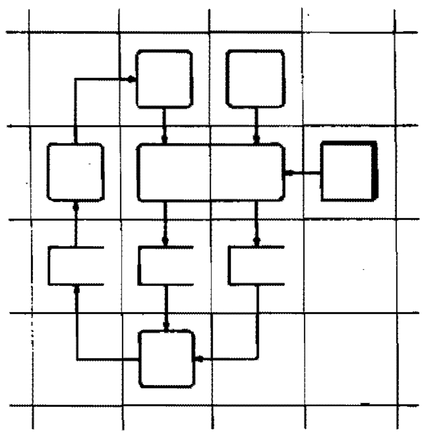


Fig. 2. Example of DFD drawn according to the grid standard.

The paper is organized as follows. In Section II, a graphic standard is defined and a set of aesthetics usually adopted by designers is identified. In Section III, a mathematical model for the layout of DFD's is presented. In Section IV, the layout algorithm for DFD's is described.

II. Aesthetics for Data Flow Diagrams

Like other diagrams used in information systems design, DFD's are usually drawn according to one of the following graphic standards:

- the *straight line standard*, where all connections are straight lines joining two symbols. When using this standard, processes are often represented with circles. In such a case, DFD's are also called "bubble charts," and
- the *grid standard*, where all connections run along the lines of a rectangular grid in which the diagram is embedded.

We refer, in the following, to the grid standard that gives rise to diagrams with high regularity and modularity. The simplest way of embedding a DFD into the rectangular grid is to make connections run horizontally and vertically in the middle of grid cells (so that, except in the case of crossings, each cell contains at most one connection) and place symbols into arrays of grid cells whose perimeter grows with the number of connections (see Fig. 2). Such solution may give rise to waste of space due to connections (see Fig. 3).

A way of solving the above problems is to use a thinner grid where each symbol occupies at least an array of $K \times K$ cells, where the parameter K is chosen according to a statistics on the average degree of symbols. This allows one to 1) have uniform symbol dimensions, 2) draw connections more closely, 3) compact the placement of symbols, and, as a consequence, 4) approach the way in which human beings tend to draw diagrams (see Fig. 4, where the same diagram of Fig. 3 is drawn using arrays 3×3 for symbols).

The above standards are general guidelines, valid for a large class of information systems diagrams. For specific types of diagrams we need additional rules. For instance, connections in DFD's enter store symbols only from the North and South sides. We have found that aesthetics typ-

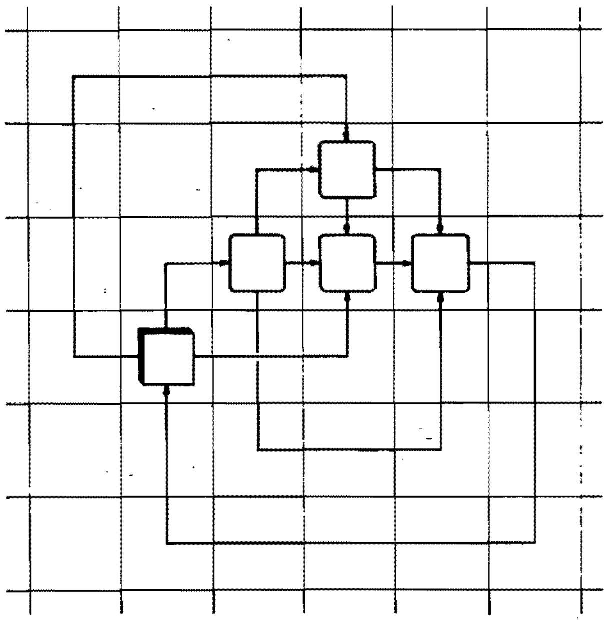


Fig. 3. Example of waste of space due to connections.

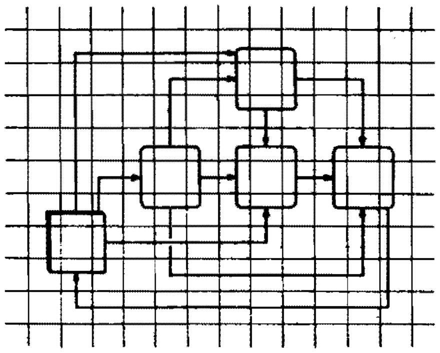


Fig. 4. Improved grid standard.

ically adopted in DFD's used in real-life applications are as follows:

*A*1: minimization of crossings between connections;

*A*2: minimization of the global number of bends in connection lines;

*A*3: minimization of the global length of connections;

*A*4: minimization of the area of the smallest rectangle covering the diagram; and

*A*5: placement on the external boundary of symbols representing interfaces.

The above aesthetics are generally not compatible; however, none of them captures in itself the idea of a nice DFD. A way to solve this conflict is to establish a priority order between aesthetics. We do this in the next section, where we show that layout features of the grid standard induce a natural hierarchy among aesthetics.

III. A Mathematical Model for DFD's Layout

The basic definitions we adopt about graphs are essentially those in [8]. We allow graphs to have multiple edges and self-loops.

Every DFD describes a *data flow schema* which, how-

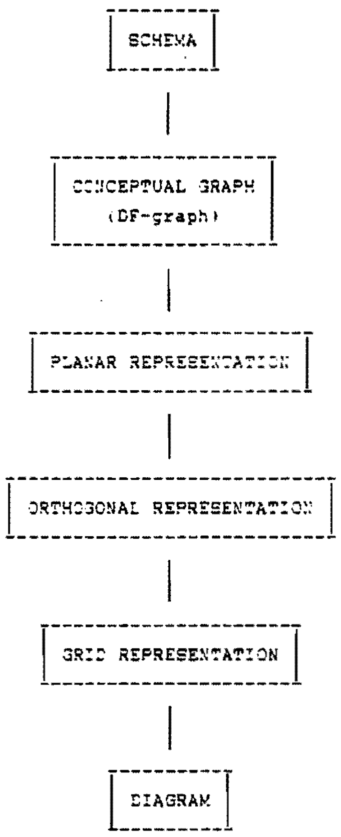


Fig. 5. Hierarchic layout representation for DFD's.

ever, may be represented by an infinite number of DFD's, differing only in their graphical appearance. A data flow schema may be formally characterized as an abstract graph called *DF-graph*. Since the aesthetics we are considering are not concerned with arrow directions, we assume that the DF-graph is undirected. Different DFD's corresponding to the same DF-graph G can be viewed as different embeddings of G in the plane. In this framework, the grid standard allows only "rectilinear" embeddings into points and paths of a "gridded paper."

The aesthetics we have identified for DFD's refer to heterogeneous properties of rectilinear embeddings: $A1$ and $A5$ refer to topology, $A2$ to shape, $A3$ and $A4$ to metric. This fact suggests a hierarchic layout representation (see also [1]), where the above properties are successively considered (see Fig. 5).

A *plane graph* is a graph such that: 1) vertices are distinct points of the plane, 2) edges are simple curves connecting their endpoint vertices, and 3) edges do not cross each other, but possibly at their common endpoints. A graph is *planar* if it is isomorphic to some plane graph. If a graph is not planar, it can anyhow be related with a plane graph by drawing it in the plane and adding fictitious vertices in order to represent crossings.

A plane graph divides the set of points of the plane that do not belong to edges into topologically connected regions, called *faces*. The unbounded region is referred to as the external face. Let G' and G'' be two plane graphs isomorphic together with their geometric duals. If dual

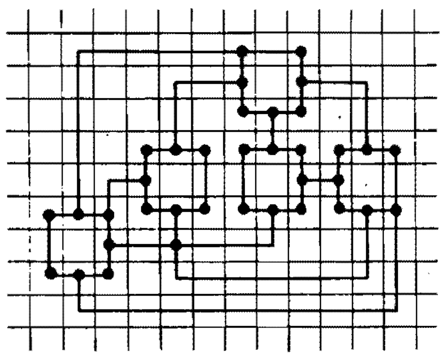


Fig. 6. Grid graph corresponding to the DFD of Fig. 4.

vertices associated to external faces are joined by the isomorphism, we say that G' and G'' are *plane equivalent*. We call *planar representations* the equivalence classes established by the plane-equivalence relation between plane graphs.

An *orthogonal graph* is a plane graph whose edges are sequences of alternated horizontal and vertical segments. Let H' and H'' be two plane-equivalent orthogonal graphs. If, for every pair of corresponding paths p' in H' and p'' in H'', p' has the same number of segments of p'', and consecutive segments form equal angles in p' and p'', then we say that H' and H'' are *shape equivalent*. We call *orthogonal representations* the equivalence classes established by the shape-equivalence relation between orthogonal graphs. The orthogonal representation deals only with the shape of the embedding, without taking into account dimensions.

We define *rectilinear grid* the infinite orthogonal graph where: 1) vertices are points with integer coordinates; and 2) edges are segments joining vertices at unit distance. A *grid graph* is an orthogonal graph whose vertices are grid points and whose edges run along vertex disjoint grid paths. The layout of a DFD drawn according to the grid standard can be described by means of a grid graph where each symbol is associated to a rectangular skeleton. See in Fig. 6 the grid graph corresponding to the DFD of Fig. 4.

We say that two grid graphs are *grid equivalent* if they are shape equivalent and corresponding segments have equal lengths. We call *grid representations* the equivalence classes established by the grid-equivalence relation between grid graphs.

If two grid graphs have the same grid representation, they have also the same orthogonal representation. If two orthogonal graphs have the same orthogonal representation, they have also the same planar representation. As a consequence, the three representations are hierarchically related, and each representation level is a refinement of the previous one. If we establish the same hierarchy between the corresponding aesthetics, we obtain the following strategy for DFD's layout.

1) First of all, the topology of the embedding is specified finding a planar representation for the conceptual graph that respects aesthetics $A1$ and $A5$.

2) Then an orthogonal shape is given to the planar representation finding an orthogonal representation (aesthetics $A2$).

3) Finally, the grid embedding is completed assigning integer lengths to segments, according to aesthetics $A3$ and $A4$.

IV. The Layout Algorithm for Data Flow Diagrams

The input to the layout algorithm is a DF-graph $G = (V, E)$ where

- $V = P + I + S$ is the set of vertices, union of processes (P), interfaces (I), and stores (S); and
- E is the set of edges, representing flows of data.

The algorithm is composed of three basic steps, each of them concerned with one of the representation levels.

A. Planarization

Step 1 receives as input a DF-graph $G = (V, E)$ and produces a planar representation P taking into account aesthetics $A1$ and $A5$. The planarization problem has been shown in [7] to be NP-hard. Hence, we adopt a heuristics.

Notice that aesthetics $A1$ and $A5$ may be incompatible; e.g., if nodes 1 and 4 in Fig. 7 represent interfaces, there is no way to place them at the same time on the external face without introducing crossings.

In our approach, the designer has to choose in advance which aesthetics he prefers. In Fig. 8, we show the procedure in case aesthetics $A1$ is chosen.

Procedure DECOMPOSE partitions graph $G1$ into its blocks by means of a well-known depth first search technique [15]. BLOCK-PLANARIZE is based on the well-known Hopcroft and Tarjan's planarity testing algorithm [9], which decomposes the graph into disjoint paths, and tests planarity by embedding one path at a time. This algorithm can be modified to accomplish the planarization, according to the following greedy strategy:

```
if a nonplanarity is detected while considering the cur-
   rent path P
   then begin
          find the edge sets involved in nonplanarity;
          choose the smallest such set;
          delete its edges from the graph;
          resume Hopcroft and Tarjan's algorithm from
             path P;
        end
```

The complexity of the global procedure Extract-Planar-Subgraph is $O(|E|)$. For a detailed analysis, see [10].

The goal of procedure FIND is to maximize the number of interface vertices on the external face and minimize the distance from the external face of remaining interface vertices, without introducing new crossings. Furthermore, nesting of blocks is avoided when possible, to allow a better achievement of aesthetics $A2$, $A3$, $A4$.

Procedure REINTRODUCE performs for each nonplanar edge $e = (v, w)$ a shortest path computation between v and w in the graph $Ge = (F + \{v, w\}, A + B)$, where F is the set of faces of the current planar representation, (F, A) is the corresponding dual graph, and B consists of edges linking v and w to faces containing them (see Fig. 9).

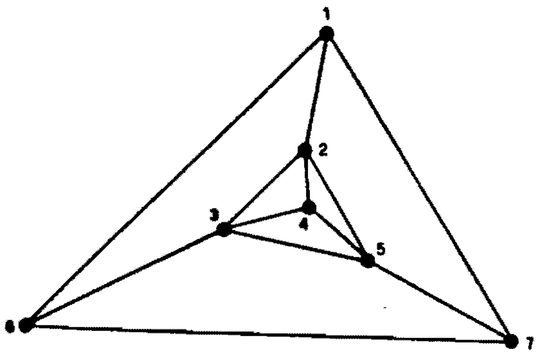


Fig. 7. Example of incompatibility between aesthetics $A1$ and $A5$.

```
Step 1: PLANARIZATION (G, P)
1. Extract-Planar-Subgraph (G, G*)
   { G* = (V, E*) is a planar subgraph of G = (V, E) }
   begin
     DECOMPOSE G in blocks (biconnected components) ;
     for each block B of G do
         BLOCK-PLANARIZE ( B, B*) ;
         { Finds a planar subgraph B* of B deleting
           nonplanar edges }
     let G* be the union of subgraphs B* ;
   end ;
2. Add-Crossings (G*, P)
   begin
     FIND a planar representation P* of G* ;
     for each nonplanar edge e do
         REINTRODUCE e minimizing crossing vertices
                            so generated ;
     let P be the resulting planar representation ;
   end ;
```

Fig. 8. The planarization step.

If aesthetics $A5$ is preferred to aesthetics $A1$, a fictitious vertex v^* is added to the DF-graph G and connected to all interface vertices [see Fig. 10(a)]. Then procedure PLANARIZATION is executed on the resulting graph: edges incident to v^* are neither deleted by BLOCK-PLANARIZE, nor intersected by REINTRODUCE [Fig. 10(b)]. At the end, v^* and all its incident edges are removed: this causes all interface vertices to appear on the same face, which is selected as the external face of the planar representation [Fig. 10(c)].

B. Orthogonalization

Step 2 receives as input a planar representation P and produces an orthogonal representation H with the minimum number of bends.

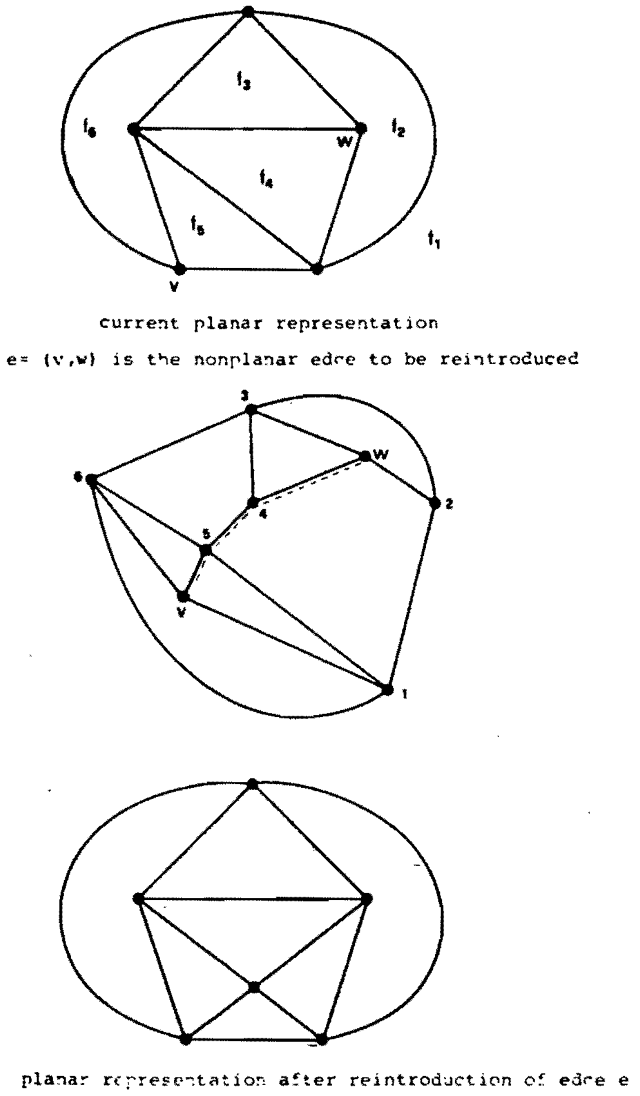

f3
f6
w
f2
f4
f5
f1
v
current planar representation
e= (v,w) is the nonplanar edge to be reintroduced
3
6
w
4
2
5
v
1
planar representation after reintroduction of edge e

Fig. 9. Reintroduction of a nonplanar edge.

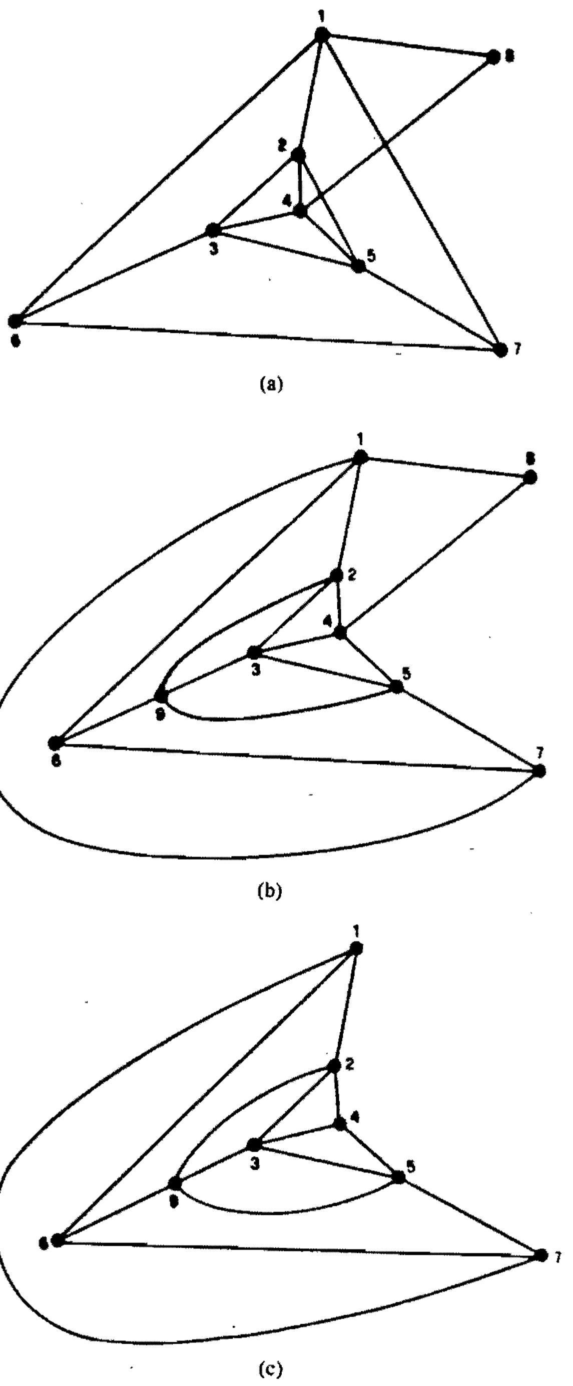


Fig. 10. Example of step 1 when aesthetics *A*5 prevails.

First of all, each vertex is expanded into a rectangular structure, called, in the following, *skeleton*. The main problem that arises while performing this task is to find a good assignment of edges entering the vertex to the four sides of the skelton. This choice may significantly affect the quality of the final layout (see Fig. 11).

The algorithm for this step makes use of the concept of k-gonal representation, defined below, which is an extension of the orthogonal representation. A *k-gonal graph* is a plane graph whose edges are sequences of segments each having slope multiple of $180/k°$ with respect to a reference axis. See Fig. 12(a) for an example of a 3-gonal graph.

A *k-gonal representation* is an equivalence class established by the shape equivalence relation between k-gonal graphs. To each edge of a k-gonal graph, we associate a cost given by its angular deviation from the straight line (measured in multiples of $180/k°$. The cost of a k-gonal graph is the global cost of its edges [see Fig. 12(b)]. According to the definition, for $k = 2$, we have orthogonal graphs, whose cost is equal to the number of bends.

In order to assign connections to the sides of skeletons, a precomputation is performed to produce an optimal k-gonal representation for the planar representation P, where $k >= \frac{1}{2} \max \{\deg(v)\colon v \text{ is a vertex of } P\}$. Then, edges are attached to skeleton sides according to their slope in the k-gonal representation (see Fig. 13).

In [14] an algorithm that uses network flow techniques is given for computing an optimal k-gonal representation with complexity $O(k^2n^2)$, where n is the number of vertices of the planar representation.

The same algorithm can now be applied to find the orthogonal representations with minimum number of bends.

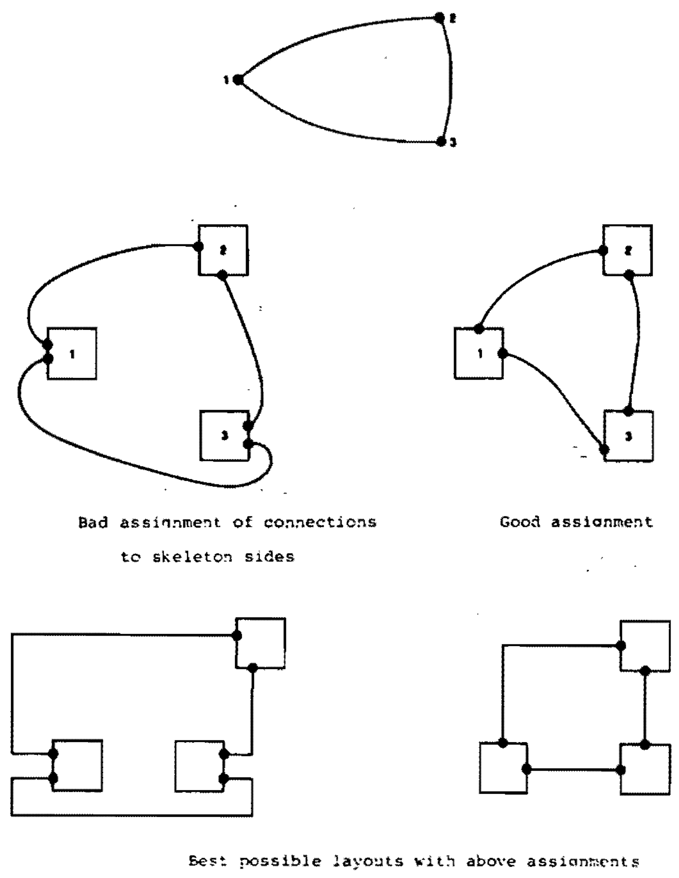
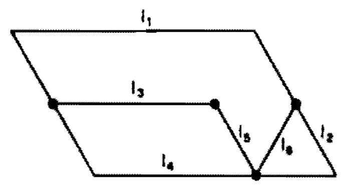

1
2
3
Bad assignment of connections to skeleton sides
Good assignment
Best possible layouts with above assignments

Fig. 11. Assignment of edges to skeleton nodes.



(a)

Edge	Cost
l_1	3
l_2	2
l_3	0
l_4	1
l_5	0
l_6	0

(b)

Fig. 12. (a) An example of 3-gonal graph; (b) costs of edges.

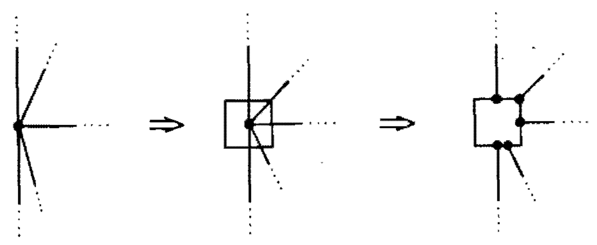
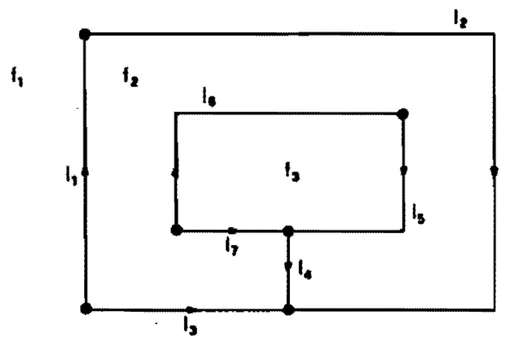


Fig. 13. Attaching edges to skeleton sides.

We show here a formulation of the bend minimization problem by means of interger linear programming.

First of all, we assign an arbitrary direction to edges in the planar representation P. Let F be the set of faces and



$A(f_1) = (-l_1, +l_3, -l_2)$

$A(f_2) = (+l_2, -l_4, -l_5, -l_6, +l_7, +l_4, -l_3, +l_1)$

$A(f_3) = (+l_5, -l_7, +l_6)$

Fig. 14. Planar representation described by edge lists.

q the external face of P. We describe P as a set of circularly ordered lists $A(f)$, one for each face f. List $A(f)$ contains the sequence of edges that are encountered when going around the contour of f in the positive direction, i.e., having f at one's right. Edges are marked positively if they are traversed in their direction, negatively otherwise. Notice that the two occurrences of an edge may appear in the same list (see Fig. 14).

In the following, we denote r the generic element of a list $A(f)$, $r.e$ and $r.s$ are the corresponding edge and sign.

There are two variables of the program associated to each element r of list $A(f)$: x_r represents the number of 90° angles in $r.e$ encountered at one's right when going around f in the positive direction; y_r represents the angle (measured in multiples of 90°) that edge $r.e$ forms in f with the next edge of $A(f)$. x_r and y_r are integer variables; x_r must be nonnegative, y_r must be positive. The global number of variables of the linear program is thus $4|E|$.

Two kinds of constraints must be imposed in order to obtain an orthogonal representation from the values of the variables.

1) The sum of angles around any vertex must be equal to 360°, that is,

$$\sum_{\text{all } r} a_{vr} y_r = 4 \qquad \text{for every vertex } v$$

where $a_{vr} = 1$ if both edge $r.e$ and the next edge in the list of r are incident to vertex v, $a_{vr} = 0$ otherwise.

2) Each face must have the shape of a rectilinear polygon whose sides are either horizontal or vertical. This is equivalent to impose that, denoted $N90(f)$, $N270(f)$, and $N360(f)$, the numbers of angles, respectively, of 90°, 270°, and 360° internal to face f, the following formula holds:

$$N90(f) - N270(f) - 2 * N360(f)$$

$$= \begin{cases} -4 & \text{if } f = q \text{ (external face)} \\ 4 & \text{otherwise.} \end{cases}$$

Step 2: ORTHOGONALIZATION (P, H)

1. *Precomputation* (P, H_k)

Produces a k-gonal representation H_k for P with minimum cost. k must be at least $\frac{1}{2}$ max deg (v); assuming a larger value allows more freedom in further phases of the algorithm.

2. *Expansion* (H_k, P')

Each vertex is expanded into a skeleton, first approximation of the final shape of the symbol. Edges are assigned to skeleton sides according to their slope in H_k.

3. *Normalization* (P', H)

Transforms the expanded graph P' into an orthogonal graph H with minimum number of bends. The algorithm for the Precomputation step is applied with the following assumptions:

a. $k = 2$
b. edges of skeleton are not allowed to be bent
c. in skeletons, corner nodes have internal angles of 90 degrees and remaining node angles of 180 degrees.

Fig. 15. The orthogonalization step.

3) Skeletons of symbols must be transformed into rectangles. That is,

• $x_r = 0$ for each element r such that $r.e$ is on the contour of the skeleton of some symbol;

• $y_r = 1$ for each corner angle of a symbol skeleton; and

• $y_r = 2$ for each non corner angle of a symbol skeleton.

The goal is to minimize the number of bends, that is,

$$\text{minimize } z = \sum_{\text{all } r} x_r.$$

Fig. 15 summarizes the orthogonalization step. Notice that the parameter k in the precomputation step can be choosen by the designer depending on the structure of the diagram and the required density of connections in the drawing. We show in Fig. 16 the behavior of the algorithm for three different choices of k.

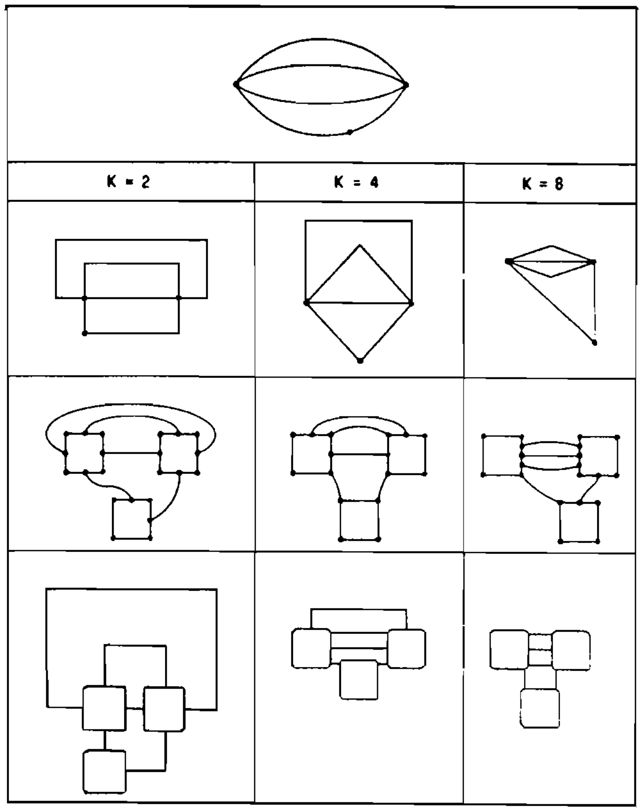


Fig. 16. Behavior of the algorithm for different choices of k.

C. Compaction

Step 3 receives as input an orthogonal representation H and produces a grid representation Q with minimum connections length.

The orthogonal representation is dimensionless: it defines the reciprocal disposition of a set of horizontal and vertical segments. In order to find a grid embedding, we have to compute the lengths of such segments. The following three constraints must be satisfied:

1) all segments are positive integers;

2) every circuit of the graph is mapped into a rectilinear polygon; and

3) two segments intersect each other only at their common endpoints.

A first description of the step is shown in Fig. 17. For further theoretical issues, see [14].

In order to build a normalized orthogonal representation, we perform a straightforward decomposition of faces

Step 3: COMPACTION (H, Q)

1. *Symbolic-Decomposition*

Construct a normalized orthogonal representation H' decomposing each face of H into rectangles by means of fictitious vertices and edges.

2. *Segment-Length-Computation*

Assign an integer length to segments of H', producing a grid representation Q for the diagram.

Fig. 17. The compaction step.

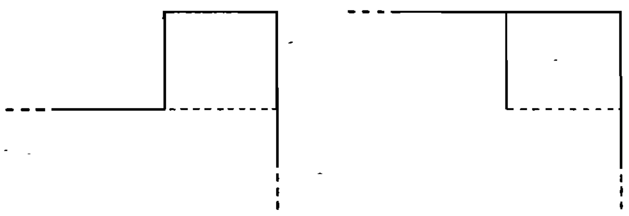


Fig. 18. Basic patterns of decomposition.

into rectangles: a rectangle is separated from a face each time one of the basic patterns shown in Fig. 18 is found. By iterating this procedure, we obtain a planar representation H' where each internal face is a rectangle, and the external face is the complement of a rectangle.

Then the segments of the orthogonal representation H' are partitioned into horizontal (set X) and vertical (set Y). For each face f of H', segments on the contour of f are then partitioned into sets $N(f)$, $E(f)$, $S(f)$, and $W(f)$, containing, respectively, segments on side North, East, South, and West of f.

Assigning lengths to segments can be expressed by means of integer linear programming as follows.

Let x_j and y_j be the lengths of segments $i \in X$ and $j \in Y$; since we want a grid representation we have the constraints

$$x_i >= 1, \quad \text{integer} \quad \text{for each } i \in X$$

$$y_j >= 1, \quad \text{integer} \quad \text{for each } j \in Y.$$

For each face (rectangle) of H', we impose that opposite sides have the same length

$$\sum_i a_{if} x_i = 0 \quad \text{for each face } f$$

$$\sum_j b_{jf} y_j = 0 \quad \text{for each face } f$$

where

$$a_{if} = \begin{cases} 1 & \text{if } i \in N(f) \\ -1 & \text{if } i \in S(f) \\ 0 & \text{otherwise} \end{cases}$$

$$b_{jf} = \begin{cases} 1 & \text{if } j \in E(f) \\ -1 & \text{if } j \in W(f) \\ 0 & \text{otherwise.} \end{cases}$$

The objective function to minimize is, for aesthetic $A3$,

$$\text{LENGTH} = \sum_i x_i + \sum_j y_j$$

and, for aesthetic $A4$,

$$\text{AREA} = \sum_{i \in N(q)} x_i + \sum_{j \in E(q)} y_j$$

where q is the external face.

The structure of the constraints allows the decomposition of the above linear program into two independent programs, one for the x_i variables and the other for the y_j variables, i.e., compaction can be performed independently in the two directions. This fact explains also why area minimization is equivalent to perimeter minimization of the external face, the latter value appearing in the objective function for aesthetic $A4$.

The above programs can be solved in polynomial time since the total unimodularity property holds for both matrix $\{a_{if}\}$ and $\{b_{jf}\}$. In [14] an algorithm is given for this step that makes use of network flow techniques. Its complexity is $O(n^2)$ for aesthetics $A3$ and $O(n)$ for aesthetics $A4$, where n is the number of vertices in the orthogonal representation H.

After a grid representation has been found, the diagram can be easily drawn by a plotting routine.

While describing the layout algorithm, we have not mentioned how we guarantee that no arrow enters a store symbol from the East or West side (see Section II). The trick is to impose that skeletons of store symbols have an horizontal band whose East and West sides are free of connections. A simple linear constraint is sufficient for this in the compaction step. The store symbol is drawn inside this band, and connections entering the skeleton from the vertical sides are bended.

V. Conclusions

We have shown an algorithm for the automatic layout of data flow diagrams that takes into account several aesthetics. Its basic strategy consists of computing successively the topology, the shape, and the dimensions of the diagram, according to a hierarchic layout model. As we have shown in Section IV, the computational complexity of the various steps is agreeable. A previous version of the algorithm (described in [2]) is presently operating on an IBM PC. The algorithm discussed in this paper is under implementation in Pascal on the same computer.

Furture research on the subject will be focused on the following topics.

1) Categorization of information systems diagrams with respect to their underlying graph structure and to the aesthetics usually followed for them.

2) Development of a parametric algorithm that can be interactively tailored to specific cathegories of ISD's.

3) Development of a tool for computer-aided layout of general ISD's. It should allow the designer to have the best advantage from the capabilities of both a sophisticated graphics editor and an automatic layout algorithm.

In this framework, an effective interaction scheme between the designer and the system will be investigated, in order to

1) establish a dialog during layout activities that allows effective exchange of information useful for a fast convergence of the algorithm;

2) provide the designer a large variety of controls, allowing him an incremental tuning of the system to his own aesthetics; and

3) make the system learn from previous experience.

References

[1] C. Batini, E. Nardelli, M. Talamo, and R. Tamassia, "A graph theoretic approach to aesthetic layout of information systems diagrams," in *Proc. 10th Int. Workshop Graphtheoretic Concepts in Comput. Sci.*, Berlin, Germany, 1984, pp. 125-137.

[2] C. Batini, M. Talamo, and R. Tamassia, "Computer aided layout of conceptual diagrams," *J. Syst. Software*, vol. 4, no. 1, pp. 163-173, 1984.

[3] M. Carpano, "Automatic display of hierarchized graphs for computer aided decision analysis," *IEEE Trans. Syst., Man, Cybern.*, vol. SMC-10, pp. 705-715, Nov. 1980.

[4] P. Chen, "The entity relationship model: Toward a unified view of data," *ACM Trans. Data Base Syst.*, vol. 1, no. 1, pp. 9-36, Mar. 1976.

[5] M. Delarche, "Quelques outils infographiques pour l'analyse structurelle des systemes," Doctoral dissertation, Univ. Grenoble, Grenoble, France, 1979.

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

[7] D. Johnson, "The NP-completeness column: An ongoing guide," *J. Algorithms*, vol. 3, no. 1, pp. 215-218, 1982.

[8] F. Harary, *Graph Theory*. Reading, MA: Addison-Wesley, 1979.

[9] J. Hopcroft and R. Tarjan, "Efficient planarity testing," *J. ACM*, vol. 21, no. 4, pp. 549-568, 1974.

[10] E. Nardelli and M. Talamo, "Fast algorithm for planarization of sparse diagram," Istituto per l'Analisi dei Sistemi e l'Informatica, CNR, Tech. Rep. R105, 1984.

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

[12] J. Soukup, "Circuit layout," *Proc. IEEE*, vol. 69, no. 10, pp. 197-213, 1972.

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

[14] R. Tamassia, "On embedding a graph in the grid with the minimum number of bends," Dip. Informatica e Sistemistica, Univ. Rome, Tech. Rep. 09.84, 1984.

[15] R. Tarjan, "Depth first search and linear graph algorithms," *SIAM J. Comput.*, vol. 1, no. 2, pp. 146-159, 1982.

[16] J. Vaucher, "Pretty printing of trees," *Software Practice and Experience*, vol. 10, pp. 553-561, 1980.

[17] C. Wetherell and A. Shannon, "Tidy drawing of trees," *IEEE Trans. Software Eng.*, vol. SE-5, pp. 514-520, 1979.

Carlo Batini has been an Associate Professor in the Department of Computer and System Sciences at the University of Rome "La Sapienza" since 1983. From 1975 to 1983 he served as an Assistant Professor at the University of Rome. His current research interests include methodologies and tools for conceptual design of databases.

Enrico Nardelli graduated in electrical engineering from the University of Rome, Rome, Italy, in 1983.

He is now a Research Associate in the Department of Computer and System Sciences at the University of Rome "La Sapienza." His current research interests include computer-aided design of information systems and layout algorithms. He is a consultant to Enidata SpA.

Roberto Tamassia graduated in electrical engineering from the University of Rome, Rome, Italy, in 1984.

He is now a Research Associate in the Department of Computer and System Sciences at the University of Rome "La Sapienza." His current research interests include computer-aided design of information systems, layout algorithms, and VLSI theory. During 1985 he visited the Coordinated Science Laboratory at the University of Illinois at Urbana-Champaign, supported by a Fulbright grant.

Mr. Tamassia is a member of the IEEE Computer Society.